Assessment of code, which aspects do teachers consider and how are they valued?

TOM NEUTENS, Ghent University- imec - IDLab - AIRO, Belgium
KRIS COOLSAET, Ghent University, Belgium
FRANCIS WYFFELS, Ghent University - imec - IDLab - AIRO, Belgium

In many countries, computer programming is becoming an integral part of the secondary school curriculum. However, many teachers, especially in the first years of Flemish secondary school, have limited experience with teaching programming. To improve their knowledge about programming, many different types of professional development programs have been proposed. Nevertheless, these programs mostly focus on technical skills and less on pedagogical skills. One aspect that is often overlooked in these programs is how teachers can assess code. To get insight into what teachers currently value when assessing code, we designed an experiment that analyzes the different aspects teachers consider during the assessment of code. During the experiment, the teachers (N=13) assess a set of programs from five different fictional learners. After the assessment, they participated in a structured interview giving us insight into the assessment process. We evaluated the transcripts of the interviews using deductive thematic analysis using a coding schema defining the different aspects of code that can be assessed. Additionally, we linked the assessment strategies of teachers to their teaching experience. Our results indicate that many teachers are unaware of the different concepts that can be part of the assessment of code which might lead to inaccurate or invalid feedback. Moreover, although our experimental group was too small to draw hard conclusions about the inter case results, our results indicate that the number of concepts considered by teachers seems to increase with experience. These results provide an initial insight into the code assessment practices of teachers and reveals interesting pathways for future research into the assessment of code.

CCS Concepts: • **Applied computing → Education**; **Computer-assisted instruction**.

Additional Key Words and Phrases: K12, Programming, assessment, Thematic analysis, Teachers

1 INTRODUCTION

In the 1980s, programming was an established part of the K-12 curriculum. However, as computers got more powerful, the primary use of a computer in the classroom shifted from being a device that had to be programmed into a tool that could be used without any programming knowledge [22]. In the last two decades there has been a resurgence of teaching programming in K-12 education. Many countries around the world are integrating programming into their national curricula in some shape or form [16]. Consequently, both the research on programming in K-12 as well as the tools which can be used to teach programming have grown substantially [5]. However, all these changes to the curriculum are mostly implemented in a top down approach. Governments, researchers, and industry are the main drivers in making the recent changes to curricula. In all these efforts to improve the programming and computational thinking skills of children one group that is often overlooked are

Authors' addresses: Tom Neutens, Tom.Neutens@UGent.be, Ghent University- imec - IDLab - AIRO, Technologiepark-Zwijnaarde 126, Gent, Oost-Vlaanderen, Belgium, 9052; Kris Coolsaet, Kris.Coolsaet@UGent.be, Ghent University, Krijgslaan 281, Gent, Oost-Vlaanderen, Belgium, 9000; Francis wyffels, Francis.wyffels@UGent.be, Ghent University - imec - IDLab - AIRO, Technologiepark-Zwijnaarde 126, Gent, Oost-Vlaanderen, Belgium, 9052.

1946-6226/2022/3-ART $15.00
https://doi.org/10.1145/3517133

the teachers. Previous research has shown that teachers encounter several hurdles when having to integrate programming into their teaching. Consequently, some researchers have looked into how these hurdles can be overcome through teacher professional development [10, 14, 39]. However, these professional development workshops mainly focus on supporting the teachers' technical knowledge about programming and related STEM content while the required pedagogical knowledge to effectively teach those subjects gets less attention. One specific example of that pedagogical knowledge is the assessment of code. Previous research has suggested different frameworks for the assessment of code [8, 40]. However, there is no knowledge about if and how these frameworks are applied in practice.

In this paper, we aim to couple the existing pedagogical frameworks for the assessment of code to the assessment teachers perform in practice. We believe that identifying the areas where theory and practice either do or do not meet is of significant interest and will allow us to formulate concrete steps for teachers to improve their assessment. To do this, we set up an experiment in which middle school teachers had to assess a set of five different graphicalvisual programs for five different fictional students. The participants were asked to provide both formative and summative feedback. After the teachers finished the assessment, we conducted a structured interview to determine how and why they made certain assessment decisions. In the following chapters, we first characterize the various assessment strategies for programming described in the literature. Thereafter, we define the educational context in which we are working. This is followed by an explanation of our experimental design, the experimental results, and a discussion section.

1.1 Assessment of code

Assessment is a broad subject. It refers to a diverse set of techniques that educators use to evaluate, measure, and document the learning progress, skill acquisition, or educational needs of students. Analyzing all of these aspects at once is impractical. Consequently, we chose to focus on the purpose of assessment (what can be assessed) and less on the different assessment formats which can be used (how it is assessed) [31]. Even though we believe that this is an important distinction, much of the literature about the assessment of code does not explicitly state this difference resulting in frameworks in which these two aspects are intertwined. In the following paragraphs, we give an overview of multiple assessment techniques and frameworks for programming described in the literature. Since literature about the assessment of code is limited, we looked at a broad range of assessment research covering different contexts and age groups.

In our search for assessment techniques, we started by looking at papers about the assessment of computational thinking. Since computational thinking is sometimes considered as an overarching term encompassing programming specific skills as well as other computing related skills, literature about the assessment of computational thinking often includes different assessment criteria for code without explicitly stating this distinction. In their review, Tang et al. clarify the difference between the types of computational thinking skills [42]. They explicitly put coding related concepts into their own category within the wide range of computational thinking concepts. Using this knowledge, we searched for the coding concepts which were present in previous work about the assessment of computational thinking. In [33] the authors describe an assessment tool for computational thinking based on the framework by Brennan & Resnick [8]. This framework contains multiple elements which are directly linked to the assessment of coding skills. For example, the authors describe concepts like sequences, loops, events, and parallelism. In [11] the authors created an assessment tool for computational thinking. They split up the concepts into five categories: syntax, data, algorithms, representing problems through a model or formula, and revising with the goal of efficiency and effectiveness. Their test does not explicitly include the assessment of code. However, a set of the questions they created were defined as coding problems. In [29], the authors suggest methods for automatically assessing seven computational thinking concepts using code features of Scratch programs. The concepts they assess are: abstraction and decomposition, parallelism, logical thinking, synchronization, flow control, user interactivity, and data representation. For each concept they define four competence levels, the competence level of a learner is determined by which code-blocks this learner uses in their programs. For example, the competence level for logical thinking is determined by the following criteria: 0) No if-statements used, 1) used if-statements, 2) used if-then-else statements, 3) used logical operations. The categorization is useful, however, it is limited to the context of scratch programs and does not take into account the context for the assessment.

Further exploration of the literature reveals that the criteria put forward by the different frameworks for computational thinking are not the only ones that can be used to assess code. Stegeman et al. [40] look at the assessment of code from a perspective of code quality. They define four categories for assessing the quality of a piece of code: documentation, presentation, algorithms, and structure. The category for documentation contains elements like clear variable naming and the correct use of comments. Presentation mainly deals with layout and formatting. Algorithms specifies criteria for how a program is solved. Finally, the structure category looks at how the code is split into sub-problems (decomposition) and how these sub-problems are grouped together (modularisation). Stegeman et al. further explored the assessment of code quality by defining a rubric to aid in the assessment [41]. This rubric associates an assessment scale with the different categories they defined before. In [6] the authors explore how different people value certain code quality metrics. They define three groups, students, educators, and developers and have them rate the importance of the following code quality metrics: readability, structure, comprehensibility, documentation, dynamic behavior, testability, correctness, maintainability and miscellaneous. In [3] the authors combine the assessment criteria for code associated with both computational thinking and code quality and extend them with assessment criteria for user experience. They propose a rubric for assessing a free choice Scratch project that has three main categories: overall proficiency, user experience, and coding and computer science concepts. Their rubric has a total of 19 dimensions for assessing student programs.

As recognized by [11], another important aspect that can be assessed when looking at code is creativity. It has been argued that writing code can foster creativity and can be used as an indicator for that creativity [37, 38]. The standard definition of creativity combines originality and usefulness however, [35] have argued that other criteria like *surprise* might need to be added to the definition. Consequently, there are different definitions used in literature. Treffinger et al. define four types of creativity. 1) Divergent thinking: the ability to generate ideas. 2) Convergent thinking, the ability to dive deeper into ideas. 3) Openness and courage to explore ideas. 4) Listening to one's inner voice [43]. Moreover, many consider creativity to be a multifaceted phenomenon involving cognitive, personality, and environmental components [36].

The previous paragraphs have listed a broad range of aspects that can be used for assessing code. These aspects are sometimes extended with other concepts that are indirectly linked to programming. One of those concepts is mathematics. In [15], the authors propose five mathematical reasoning principles associated with programming: Boolean logic, discrete math structures, precise specifications, modular reasoning, and correctness proofs. Others have taken a specific mathematical concept like Boolean logic and integrated it with other code assessment categories [19]. Another aspect related to coding is the assessment of testing and debugging. In [8] testing and debugging is described as an essential skill required when writing programs and that should be a part of the assessment process. Even though debugging and testing skills are closely intertwined with programming skills, some consider them to be skills that should be taught and assessed separately [24]. A final aspect which is not directly related to code but might be a part of the assessment of code are the learners' personal characteristics like punctuality and neatness. As described in [20], multiple empirical studies have revealed that personal aspects are often taken into account by teachers when assessing the cognitive ability of students.

1.2 Research questions

As described in the previous paragraph, there are many aspects that can be taken into account when assessing a program. The main question we try to answer in this paper is which of these aspects are used by teachers and how important they are in the assessment.The main question we try to answer in this paper is whether the assessment criteria that teachers use in practice align with the ones documented in literature. Specifically, we are interested in how many different aspects teachers consider in their assessment and how they are valued. For clarity we formulated twothe following research questions:

(1) Which criteria do teachers take into consideration when assessing code?
(2) What value do teachers attach to the different criteria they consider when assessing code?
(3) How does teaching experience influence which criteria are selected and how they are valued?

2 METHOD

Because little is known about how teachers assess code, the main aim of this study was to lay the groundwork for additional research in this field. Since we wanted to find new insights into the assessment of code with the goal of opening up directions for future research, we opted for a multi-case study. We collected qualitative data for each case using a semi-structured interview and analyzed the interviews using deductive thematic analysis.

2.1 Context

In the educational context we are working in, programming is slowly being introduced into the curriculum. For now it is only part of the curriculum in the first two years of Flemish secondary school (ages 12 to 14). Moreover, the Flemish government has decided that these skills are part of the core competences all students in the population have to master. Concretely, the government requires learners to acquire procedural knowledge about sequence, iteration, and selection [44]. It should be noted that, the government only defines the concepts students have to master and not how teachers should teach these concepts. [1]. As a result of the context we are working in, our research focuses on teachers who are active in the first two years of secondary school.

To get some more detailed information about the background of the participants, we sent out a survey with the following questions: (1) In which grades of secondary school do you teach? (2) Which courses do you teach? (3) How many years have you been teaching programming? (4) How many hours on average did you teach programming last school year. (5) On a scale of one to ten, what is your personal experience with programming? The responses to these questions provided insight into the background of the participants.

Since graphicalvisual programming environments are often used in the first year of secondary school, we chose to do the same for our experiment. Moreover, since physical computing is often used as a context for teaching programming to that age group and our research team has a lot of experience with physical computing exercises using an Arduino based microcontroller platform, we chose it as the platform for our exercises. To collect the data required to answer our research questions we set up an experiment consisting of two main parts. In the first part, the teachers get a set of 25 programs. These programs are solutions to five different programming assignments for five different fictional learners. The teachers are asked to give personal written feedback to the learner as well as a score for each question. In the second part of the experiment we conduct a semi-structured interview [23] with the teacher asking them to explain the reasoning behind the assessment as well as other aspects like how difficult they thought the assignment was. In the following paragraphs we explain these parts in more detail. In the following sections we first describe our experimental setup, thereafter, we explain the instruments used, finally, we give a detailed description about the data collection and analysis.

[1]https://onderwijsdoelen.be/

2.2 Instruments

To organize our experiment, we created the following research instruments: 1) A list of student personas describing the background of five fictional learners. 2) Five programming assignments the fictional learners had to solve. 3) A set of 25 solutions, one for each persona-assignment pair. 4) A blank feedback form the teachers had to fill out. 5) A set of questions for our semi-structured interview. 6) A small questionnaire to collect data about the teaching experience of the participants. Below, these instruments are discussed in more detail.

2.2.1 Personas. Since the variation of personal attributes for learners in the real world is large, we used a persona methodology to gather the information in a manageable format [12, 45]. These personas describe a learner and define the knowledge they have about programming. They were created by one of the members of the research team and were validated in a group discussion between three members of the team. Our aim was to create personas for which the programming skill varied from limited to very good. The choice of ability for each of the personas was mainly based on the teaching experience of the members of the team. The personas were used as a reference to define solutions linked to thetheir background knowledge of the persona. Table 1 defines these five personas in detail. In this table, we gave the personas a name to be able to use them as a reference in this paper. However, we did not provide these names to teachers to prevent any biases.

Table 1. Learner personas

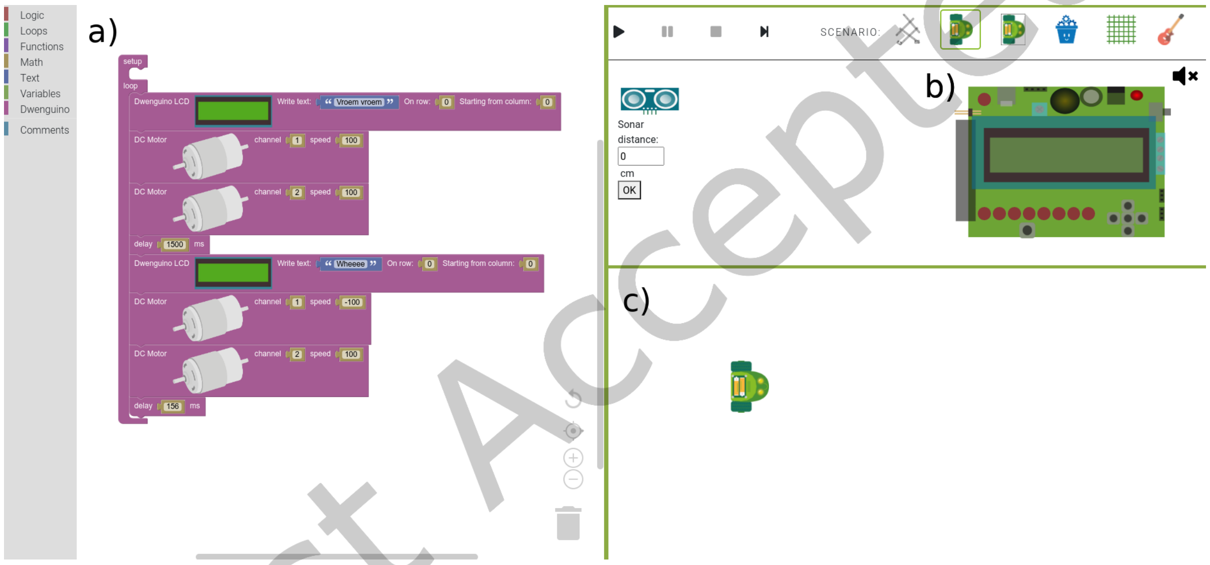
Name	Description
Lisa	Lisa likes STEM. Her father is an engineer and he often takes her to different STEM-workshops where she has come into contact with multiple programmable platforms. At home, she has written multiple programs in Scratch. At school, she finds that STEM classes go too slow for her. However, she always tries to solve the exercises to the best of her abilities. When she finishes all the assignments, she likes to add her own creative touch.
David	David likes programming. However, he has no experience apart from the exercises they do in class. He has a good understanding of the iteration concept. Consequently, he tries to apply it as much as possible. He does not fully understand how the if-then-else-block works and sometimes has issues evaluating conditions.
Sarah	Sarah usually pays attention during class. Programming is not her favorite subject but she tries her best to solve the exercises. She missed some of the previous classes because she was ill. During these classes the teacher explained the if-then-else-block. Because she missed that explanation, she still does not fully understand how the block works.
Kim	Kim has done multiple coding games at home like Blockly Maze, Code Combat and LightBot. She has mastered basic concepts like iteration, condition, and selection. However, these games do not contain any time based behaviour like the use of a delay command in an Arduino program. Consequently, she often has trouble with exercises that contain specific time behaviour.
Sep	Sep does not like programming. He tries to solve the exercises and he has a basic knowledge of the different programming constructs he can use. However, he only uses these constructs when he really has to. He thinks loops are stupid and prefers just copy and pasting blocks, which is a lot faster in his opinion.

2.2.2 Programming assignments. In addition to the five personas defined above, we defined five programming questions. We used the DwenguinoBlockly programming environment[2] and a simulator as a platform (figure

[2]https://www.dwengo.org/dwenguinoblockly/

1). Consequently, the questions were specifically made to be solved using this platform. The platform has a graphicalvisual programming environment based on Google Blockly and supports multiple robot simulations which can be executed inside the browser. For this experiment we used the simulation of the Dwenguino microcontroller board which includes an LCD-screen, buttons, and LEDs and the simulation of a driving robot with two wheels and a distance sensor. In this environment, programs are written using the structure of an Arduino program. The program always has a setup-loop-block into which all other code blocks are snapped. The environment supports different input and output blocks to, for example, read a sensor value or turn an LED on or off. It also includes blocks for the basic programming constructs like loops, if-statements, and variables. This programming environment was used as the context for the five questions we defined.

Fig. 1. Overview of the DwenguinoBlockly programming tool. a) The visual code editor (showing Lisa's solution to the first assignment). b) Simulation of the microcontroller board including an lcd-screen, 9 LEDs, a buzzer, and five push buttons. c) The simulation of the riding robot with two dc-motors.



```
setup
loop
  Dwenguino LCD   Write text: "Vroem vroem"  On row: 0  Starting from column: 0
  DC Motor   channel 1  speed 100
  DC Motor   channel 2  speed 100
  delay 1500 ms
  Dwenguino LCD   Write text: "Wheeee"  On row: 0  Starting from column: 0
  DC Motor   channel 1  speed -100
  DC Motor   channel 2  speed 100
  delay 156 ms
```

In table 2 you can read the five questions we defined. Using both the personas and the questions, we created a solution for each persona for each question resulting in 25 solutions. We aimed to make the questions open enough so teachers would be able to attach the assessment criteria they thought were relevant to the question but not too open, so the variety of possible answers would be too large. These types of open assignments are also common a physical computing context since, when working with physical systems, it is often impossible to create exact solutions as a result of influences from the real world.

2.2.3 Solutions. We created a solution for each persona-question pair resulting in 25 solutions. The set of solutions was created using an iterative review process. One researcher was responsible for creating and updating the set of questions. After each update, the set was reviewed by The set of solutions was reviewed by four experts from our research lab with different levels of teaching and programming experience. Two of the experts have professional experience teaching in secondary school while the others have experience in higher education. Three of the researchers are computer scientists while the fourth is a mathematician. We considered these different points of view to ensure the solutions we provide are on par with solutions real learners from the selected grade

Table 2. Questions that were solved by the personas

Nr.	Question
1	Write a program that makes the robot drive in a square shape on the ground.
2	Write a program that makes the robot drive straight and stop before it hits the wall.
3	Use the LEDs on the board to simulate a racing light. Have the lights count down one by one, after the countdown they have to blink three times.
4	Make up a yes or no question. Show this question on the lcd-screen. Let the player answer using the buttons on the board. When the player presses the wrong button, show a message on the screen informing the player of the mistake. If the answer is correct also inform the player using the lcd-screen
5	Make the robot drive forward. Make sure the robot stops and turns 90 degrees when it is less than 30cm from a wall. After the robot has nearly hit the wall four times, the robot should stop.

levels would produce. Since the solutions are written in a visual programming language, including screenshots of all of them here would be impractical. Consequently, we created two tables explaining the solutions. Figure 2 visualizes the solution to question three for each persona. Table 3 shows a descriptive overview of the solutions.

2.2.4 Assessment sheet. To add some structure to the assessment made by the participants, we created a basic assessment sheet. The sheet has five sections, one for each of the fictional learners. Each section has a table into which the participants can fill out the scores for each solution as well as a text box for writing down the personal feedback for that student. This sheet was also used as a guide during the interview.

2.2.5 Interview questions. To collect data about how teachers performed the assessment, we conducted a semi-structured interview. An overview of the main interview questions is given in table 4. Questions (1), (3), and (4) were designed to collect data about what teachers consider during the assessment as well as the value they attach to different criteria. Questions (2), (5), and (6) give us information about the teachers' experience and confidence allowing us to contextualize the assessments they make. All interviews were recorded for future reference and the teachers were asked to send us the papers or documents they used for the assessment to further document the process.

2.2.6 Experience questionnaire. To get some more detailed information about the background of the participants, we created a small survey with the following questions: (1) In which grades of secondary school do you teach? (2) Which courses do you teach? (3) How many years have you been teaching programming? (4) How many hours on average did you teach programming last school year. (5) On a scale of one to ten, what is your personal experience with programming? The responses to these questions should give us some insight into the background of the participants.

2.3 Data collection

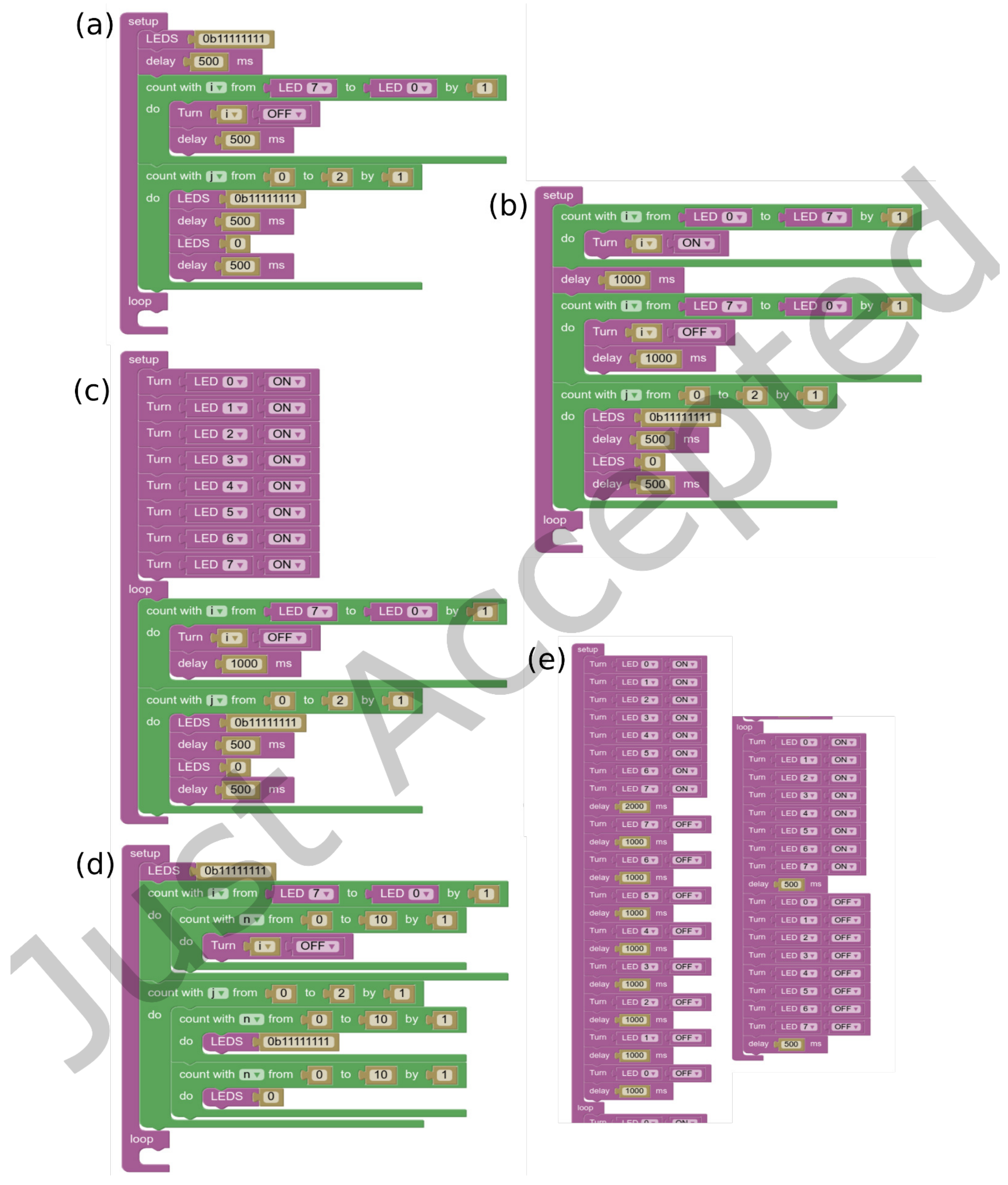
Using this list of questionsUsing the instruments described above, the rest of our experiment was set up as follows. To recruit participants for the experiment, we used snowball sampling. First, we sent out an invitation to teachers through multiple channels. These included contacts from a list of schools, school groups, and the government. These people were asked to forward the information to teachers in their contact lists. We reached out to teachers from across the Flanders to get a group of teachers with varying backgrounds. The invitation letter contained a detailed description of the experiment as well as the data we would be collecting. It also contained a link to a form the teachers could use to register to participate in the experiment, when registering, they had to consent to the data collection statements in the invitation letter which is in accordance with our university guidelines. Teachers who registered were contacted and provided the necessary information about the experiment. Initially,

Table 3. Summary of the solutions to be assessed by the participants.

Nr.	Lisa	David	Sarah	Kim	Sep
1	Uses the loop section of the setup-loop block to repeat going straight and turning 90 degrees. Added text to the lcd-screen when going straight and turning.	Uses a for-loop to repeat going straight and turning 90 degrees four times. Stops motors after for-loop.	Uses the loop section of the setup-loop block to repeat going straight and turning 90 degrees.	Uses the setup-loop block for infinite repetition. Uses loops to create an artificial delay. Delay block is not understood.	Does not use loops, iteration steps are copy and pasted after each other.
2	Basic program to stop at wall (as shown in figure 1). If measured distance < 100 then stop motors else turn both motors.	Same solution as Lisa but motor blocks are switched.	If measured distance < 100 stop motors and wait 10000ms. After the if statement turn both motors on. This is functionally correct for the first 10 seconds of execution.	Same as Lisa but without delay blocks.	Does not use sensor input or if-then-else block. Used trial and error to let the robot drive forward and stop after 3 seconds (close to the wall).
3	*Detailed overview is provided in figure 2.*				
4	Question in setup, large if-then-else structure in loop. If North button pressed show *correct* else if south button pressed show *not correct* else show button press options. Buttons have to be held down to see the answer.	Same structure as Lisa her solution, however, the second condition is inverted. Consequently, the solution shows *not correct* by default.	Uses three while loops instead of the if-statement in Lisa her solution. While no buttons pressed, show answer options. While north pressed, show *correct*. While middle pressed, show *not correct*	Same solution as Lisa but without delay. This causes lcd flickering.	All code in *setup*. No loops or if-statements. Uses special *wait until button pressed block* to show correct answer only when north button is pressed.
5	Combination of solution 1 and 2 with variable speed. Reduces speed by 25% of original speed on each turn.	All code in setup: for i from 1 to 4: { while distance from wall > 30 { go straight; } turn robot } stop robot	Similar to David his solution, however, without the for loop. Consequently, the robot only turns once and not four times.	Similar solution to Lisa, however, uses variable for the number of iterations instead of reducing the speed of the robot. Also uses a loop to make the 90° turn instead of using a delay block like in question one.	Similar solution to David, however, code is in loop and keeps repeating forever. Additionally, the angles are far from 90°.

24 teachers registered for the experiment using our form. However, in the end, just 13 responded to our request to schedule a date for the experiment, all but two came from different schools. Even though 13 participants seems low, since the data we collected was sufficiently rich, containing in depth information on the specific strategies each individual teacher used to assess code and it reached sufficient saturation by having participants with similar assessment strategies, we decided that 13 participants yielded sufficient data [2, 17]. Adding participants would

Fig. 2. Solutions to question 3 (racing light). (a) Lisa came up with the more or less optimal solution. (b) David used a loop to initialize the state of the lights instead of the LEDS block. (c) Sarah turned the LEDs on individually. (d) Kim used the execution time of a single block to delay the code. (e) Sep does not use loops.



(a)

```
setup
  LEDS 0b11111111
  delay 500 ms
  count with i from LED 7 to LED 0 by 1
  do
    Turn i OFF
    delay 500 ms
  count with j from 0 to 2 by 1
  do
    LEDS 0b11111111
    delay 500 ms
    LEDS 0
    delay 500 ms
loop
```

(b)

```
setup
  count with i from LED 0 to LED 7 by 1
  do
    Turn i ON
  delay 1000 ms
  count with i from LED 7 to LED 0 by 1
  do
    Turn i OFF
    delay 1000 ms
  count with j from 0 to 2 by 1
  do
    LEDS 0b11111111
    delay 500 ms
    LEDS 0
    delay 500 ms
loop
```

(c)

```
setup
  Turn LED 0 ON
  Turn LED 1 ON
  Turn LED 2 ON
  Turn LED 3 ON
  Turn LED 4 ON
  Turn LED 5 ON
  Turn LED 6 ON
  Turn LED 7 ON
loop
  count with i from LED 7 to LED 0 by 1
  do
    Turn i OFF
    delay 1000 ms
  count with j from 0 to 2 by 1
  do
    LEDS 0b11111111
    delay 500 ms
    LEDS 0
    delay 500 ms
```

(d)

```
setup
  LEDS 0b11111111
  count with i from LED 7 to LED 0 by 1
  do
    count with n from 0 to 10 by 1
    do
      Turn i OFF
  count with j from 0 to 2 by 1
  do
    count with n from 0 to 10 by 1
    do LEDS 0b11111111
    count with n from 0 to 10 by 1
    do LEDS 0
loop
```

(e)

```
setup
  Turn LED 0 ON
  Turn LED 1 ON
  Turn LED 2 ON
  Turn LED 3 ON
  Turn LED 4 ON
  Turn LED 5 ON
  Turn LED 6 ON
  Turn LED 7 ON
  delay 2000 ms
  Turn LED 7 OFF
  delay 1000 ms
  Turn LED 6 OFF
  delay 1000 ms
  Turn LED 5 OFF
  delay 1000 ms
  Turn LED 4 OFF
  delay 1000 ms
  Turn LED 3 OFF
  delay 1000 ms
  Turn LED 2 OFF
  delay 1000 ms
  Turn LED 1 OFF
  delay 1000 ms
  Turn LED 0 OFF
  delay 1000 ms
loop
  Turn LED 0 ON
  Turn LED 1 ON
  Turn LED 2 ON
  Turn LED 3 ON
  Turn LED 4 ON
  Turn LED 5 ON
  Turn LED 6 ON
  Turn LED 7 ON
  delay 500 ms
  Turn LED 0 OFF
  Turn LED 1 OFF
  Turn LED 2 OFF
  Turn LED 3 OFF
  Turn LED 4 OFF
  Turn LED 5 OFF
  Turn LED 6 OFF
  Turn LED 7 OFF
  delay 500 ms
```

have added little additional insight into the different criteria that are considered during the assessment and would not immediately invalidate the trend observed between teaching experience and assessment strategies. Moreover,

Table 4. Interview questions

Nr.	Interview question
1	How did you proceed with the assessment?
2	How hard/easy was this assignment for you?
3	Could you give a description of the different students you assessed?
4	Could you point out some of the strengths/weaknesses of each of the students?
5	Do you think your assessment is a better representation of the capabilities of the learners than the assessment another teacher would make?
6	Is there something you think could help you with the assessment you just performed?

these small scale experiments allow us to identify where possible shortcomings in the assessment lie and allow us to formulate hypotheses for future research. These hypotheses can than be tested on a larger group of teachers from the first two years of Flemish secondary education.

The initial information we sent to the teachers contained a short tutorial explaining the details of the programming environment we would use during the experiment. The teachers were asked to go through this tutorial and make sure they understood the concepts that would be used in the experiment. Once the teachers had gone through the tutorial, we scheduled a time and date for the experiment itself. The teachers were not compensated for their participation, however, they did receive a certificate of participation which they can include in their professional development plan at their school. Initially, we were planning to invite the teachers into our lab to conduct the experiment. However, due to governmental restrictions, we decided to do the experiments using one-on-one online meetings between a teacher and a researcher. Each experiment required about three hours. We started a video call with the teachers using the platform of their preference. At the start, we explained the structure of the experiment, sent them the solutions to the questions they had to assess in a digital executable format and explained what was expected of them. After the introduction, the teachers got about 1 hour and 15 minutes to perform the assessment. During this period, the teachers turned off their cameras and muted their microphones to work independently. When they finished the assessment the camera and microphone were turned back on and the interview started. The researcher doing the call, remained online to answer any questions the participant had during the assessment. The interview started by asking the teacher to go over each exercise and explain how they performed the assessment, what score they gave, and what feedback they wrote down. This part of the interview was guided by what the teachers wrote down on the assessment sheet described in section 2.2.4. In some cases the interviewer asked to specify, clarify, or explain certain elements of the assessment. In the second part of the interview, the researcher continued by asking the rest of the questions defined in section 2.2.5. Sometimes, depending on the course of the interview, these questions were asked during the first part of the interview. After the interview, the participants were asked to fill out the experience questionnaire described in section 2.2.6.

2.4 Data analysis

Once the interviews were collected we used deductive thematic analysis to further analyze our data [7]. To extract the necessary information from the interviews, we manually transcribed the recordings (because automatic transcription tools yielded insufficient results for our language) and applied the coding schema on the text. The coding was performed using classic word processing software by highlighting the text in accordance with the categories in the coding schema. The highlights for each schema category were then grouped in a new document for further analysis. Finally, we summarized how each teacher assessed each concept in our coding schema. The coding was performed by one of the researchers in our team and was reviewed by another member of the research team, both with extensive background knowledge in computing education. The reviewer assessed if the coding was consistent with the description in the schema. We chose this reviewed single coder strategy described above because: 1) It was feasible for one person to code the transcripts. 2) It avoids the application of special techniques for enhancing inter-coder reliability[32]. 3) Reviewing the coding enhances the reliability when depending on a single coder [9, 13]. To analyze the transcripts we constructed a coding schema containing the different aspects of code that can be assessed. This coding schema was constructed by first listing all aspects that literature describes as possible assessment criteria for code (section 1.1), supplemented with personal aspects of the learner, and then grouping and filtering them. We mainly considered practical reasons for not including certain aspects into our coding schema. For example, concepts like parallelism or events are not present in the pieces of code we presented to the teachers. Consequently, there is little value in including them in our coding schema. The final version of our coding schema is shown below.The final version of our coding schema is shown in table 5, it lists all categories and subcategories together with a sentence to contextualize the category. In the following paragraphs we provide a more in depth discussion about each concept in the schema.

Function: This category contains assessment criteria related to the functionality of the code [3, 18]. Under the category of function we included the concepts: 1) Correctness: Does the program do what the assignment said it should do? Does it meet the requirements set forward by the teacher? 2) Efficiency: How much computational power is required for executing the program? Are there more computationally efficient ways of writing the program? 3) Usability: Is the application easy to use by the teacher? Is it clear how the program can be controlled?

Readability: All aspects of the program that contribute to more readable code. We limited ourselves to the following four indicators of readability since the others presented in the literature are less relevant for graphicalvisual programs [6]: 1) Variable names: Does the variable name communicate the meaning of the variable? Is the name relevant to the program context? 2) Comments: Does the learner use comments in his or her code? Are the comments easy to read? Do they communicate the meaning of the program? 3) Layout: Is the layout of the blocks clear? Are there any overlapping blocks? Are there unused blocks in the program? 4) Conciseness: Is the code DRY (Do Not Repeat Yourself)? Is there any copied and pasted code? Does the code contain blocks which are not executed? Conciseness is especially important in graphicalvisual programs since these programs quickly get too large to fit onto the screen.

Application of programming concepts: There are many possible programming concepts to include into our analysis [3, 4, 8, 33] . We limited the concepts we selected to those which are used in the solutions presented to the teachers. These are sequence, iteration, selection, condition, variables, time, and inputs/output.

Application of algorithmic thinking: Algorithmic thinking has been defined as: "*mental orientation to formulating problems as conversions of some input to output and looking for algorithms to perform the conversions*"[26]. Since this is a broad definition we defined the following aspects, which we will analyze: 1) Analyze a given problem. 2) Specify a problem precisely. 3) Construct a correct algorithm to a given problem using the basic actions. 4) The ability to think about all possible special cases [18].

Execution of testing and debugging: Are learners able to locate faults in their code? Are learners able to correct these faults? Do learners sufficiently test their code to be able to locate faults? [24, 27, 30]

Mathematical concepts: Many mathematical concepts can be linked to programming [15]. However, simple graphicalvisual programs often only contain a limited number of mathematical concepts. The concepts we decided to include for our analysis are Boolean logic [19] and basic arithmetic [3] since these were the only concepts present in our exercises.

Creativity: Since creativity has many definitions, we mainly focus on the category of divergent thinking [43]. We limit our coding to this category since other categories like convergent thinking and the ability to generate ideas are less relevant for the experiment we set up because we do not provide enough artifacts to teachers to be able to assess them. Divergent thinking itself can be characterized in multiple ways like fluency, originality, and flexibility [34]. Our work will mainly focus on originality since both fluency and flexibility are impossible to measure using a single artefact.

Personal aspects: This category encompasses all personal traits the learners show which are not directly linked to programming or computational thinking. We are looking for aspects like helpfulness in the classroom, neatness, disruptive behavior, student effort [20]. We define this broadly since it is not directly related to the assessment of code. However, we do want to analyse what value teachers attach to these aspects compared to the ones more directly related to code.

We are aware that these concepts are extracted from a broad literature exploration for many different educational contexts. Consequently, we realize that not all of these concepts might be applied/be applicable in certain educational contexts. However, our goal is to identify if and how these concepts are used to assess learners. Using the proposed coding schema, we annotated the interview transcripts. Each time a teacher talked about a specific concept, we marked it in the text. After marking all elementshighlighting and grouping the concepts from our coding schema in the interview of each teacher, we made a summary for each concept. These summaries contain the different aspects teacher consider when assessing each concept. These summaries are listed in the results section. Finally, after analyzing if and how the different assessment criteria in our coding schema are used, we explored the relation between teaching experience and assessment strategy. We grouped the participants into the groups: teachers without any prior experience teaching programming, teachers with between one and five years of experience teaching programming, and teachers with more than five years of experience teaching programming. Using these groups, we performed a qualitative intra- and inter-group analysis of similarities and differences.

3 RESULTS

The interviews we conducted (N=13) resulted in 408 minutes of video recordings. We manually transcribed the interviews to have a textual reference for our analysis. Using the coding schema defined in the previous section, we annotated the transcripts by highlighting the parts where the interviewed teacher was talking about one of the concepts in our coding schema. After this annotation process, we grouped the highlighted parts of the text according to their respective coding concept. Next, we summarized how the different teachers assess each concept in our coding schema.These were transcribed and coded using the techniques described in section 2.4. Table 6 shows an overview of how many teachers mentioned a certain aspect. It is clear that some aspects like functionality are considered more often than others. Additionally, based on the results of our survey, we divided the teachers into four groups: (1) Teachers without previous programming teaching experience ($N = 2$). (2) Teachers with one to five years of programming teaching experience ($N = 2$). (3) Teachers with more than five years of programming teaching experience ($N = 5$). (4) Teachers who did not wish to disclose their experience for processing in our experiment ($N = 4$). We performed an inter- and intra-group qualitative analysis of the differences and similarities in the assessment strategies. The following sections first elucidate the results of our deductive thematic analysis thereafter, we describe how teaching experience affects the assessment strategies.

3.1 Function

Our coding schema defines three aspects related to the functionality of the program. The first element is functional correctness. All of the thirteen participating teachers included functional correctness as an element in their assessment. For many of the teachers, executing the program in the simulator and checking if it has the desired function is the first part of their assessment process. For most teachers, the result of this execution strongly determines the score and feedback learners get. Some teachers explicitly state the criteria they want the solutions to adhere to. When more of these criteria are satisfied, the learners get a higher score. When we analyze the criteria, the teachers propose it is clear that most of them are related to the functionality of the program. One

Table 5. Coding schema summary

Category	Sub categories	Contextualization
Function	Correctness	Does the program do what is required?
	Efficiency	How computationally efficient is the program?
	Usability	Is it clear how the program can be used?
Readability	Comments	Are comments used and are they understandable?
	Layout	Does the code layout follow a certain structure?
	Conciseness	Is there repeated code?
	Variable names	Do variable names give a clear indication of the types of values it will contain?
Programming concepts	Sequence	Are commands executed in the correct order?
	Iteration	Are loops used correctly?
	Condition	Is a certain statement true or false?
	Selection	Is it clear how the control flow can be changed based on a condition?
	Variables	How can values be stored during execution?
	Time	How does time influence the execution of the application?
	Input/output	How can external (sensor) values be read and how can external actuators be controlled?
Algorithmic thinking	Analyze given problem	Which sub-problems exist?
	Specify precisely	Which solution strategies can be used to solve the problem?
	Construct algorithm	How can the solution strategy be translated to a program?
	Think of special cases	How can we be sure our program works in all cases?
Testing and debugging	/	Why does our program not do what we want it to do?
Mathematical concepts	Boolean logic	Is this true or false?
	arithmetic	How do I add, subtract, divide, or multiply these numbers?
Creativity	additional elements	How can I add my own touch?
	novel solutions	Are there different ways of accomplishing the same result?
Personal aspects	/	How does he or she behave in the classroom?

teacher described his assessment method as follows: "*For exercise 5, the one where the robot tries to escape. There the students had to fulfill multiple criteria. For example, the robot had to drive. It had to stop at 30cm from the wall. It should be able to turn 90 degrees and should stop after four attempts.*" Other teachers did not go into as much detail

Table 6. An overview of the concepts extracted from literature with the number of teachers in each experience group that mentioned elements of these concepts in their interview.

Aspect of assessment	**Number of teachers**				
	Less then one year experience (N = 2)	**One to five years experience** (N = 2)	**More then five years experience** (N = 5)	**No experience data** (N = 4)	**Total** (N = 13)
Function	2	2	5	4	13
Readability	1	2	4	2	9
Programming concepts	2	2	5	4	13
Algorithmic thinking	0	1	2	1	4
Testing and debugging	0	1	4	1	6
Mathematical concepts	1	1	2	1	5
Creativity	1	1	5	3	10
Personal aspects	2	1	4	4	11
Average number of concepts per teacher	4.5	5.5	6.2	5	5.46

when defining criteria. For example, one of the teachers used the following criteria: "*I mainly looked at: Does the program work? Did they achieve their goal? And did they do any unnecessary steps?*" Even though functionality is the most important factor in determining a score, some teachers do make exceptions. For example, in one of the solutions they have to assess, the student reversed a condition by using a greater than symbol instead of a less than symbol. Consequently, the code has no visible output. For some teachers this results in a very low score. However, some teachers recognize that the reversal of the symbol is only a small error resulting in a high score and a small remark in the written feedback.

Notwithstanding that most teachers attach a high value to the functional correctness of the programs, some teachers (N=2) do not consider this as a primary element for their assessment. These teachers do take function into account, but only as a last step in the assessment process. They define criteria based on the coding constructs they think the learners should be able to apply in that specific exercise. Some of these criteria are: does the student correctly use sensors and actuators, and does the student apply the principle of iteration correctly? Once these questions are answered, the teachers look at how the program executes to determine their final judgement.

The second aspect related to function we defined in our coding schema is efficiency. We specifically define this as computational efficiency, which is the computational power required for executing a program. Since we are working with simple programs, computational efficiency is mainly determined by the number of blocks in the program that get executed but do not contribute to the function of the program. Ten of our thirteen participants mentioned criteria related to efficiency during the interview. However, it is not always clear if they are considering computational efficiency or program conciseness. For example, one of the teachers stated the following: "*If they used too many blocks, then I don't consider the code to be efficient.*" Using more blocks can result in computational inefficiency, verbose code, or both. Consequently, only considering the number of blocks when looking at a solution might not be a valid way of assessing computational efficiency. Other teachers take a different approach to assessing efficiency, one of them explained it as follows: "*Student 2 always reaches his goal. However, sometimes he has unnecessary steps. When I removed those parts from the program, it still functioned correctly. So, in my opinion, those were unnecessary.*"

The final aspect related to function we analysed is usability. None of the participants explicitly mentioned program usability as a criterion for their assessment. However, some teachers do include it as a general assessment of function. For example, one of the teachers said the following about a solution to question four: "*For student 4, I noted, the question is correctly shown on the screen. However, it was quickly replaced by the possible answers. Very quickly, I was not able to read the question before the possible answers were shown.* " This shows that usability is a criterion for some; however, it is not explicitly defined as usability.

Previous paragraphs explained that teachers attach a lot of value to the function of a program. Many teachers consider functional correctness as the main aspect of their assessment. Nevertheless, some teachers attach less value to functional correctness and focus more on the correct application of certain programming concepts. Functional usability is never explicitly used as an assessment criterion. However, some teachers include it as part of the assessment of program functionality without being aware it is a separate aspect related to function. Most of the participating teachers are aware that efficiency should be part of the assessment. Nonetheless, the concept of computational efficiency is not clearly defined by any of the participants. Often, computational efficiency and code compactness are assessed together, possibly leading to an inaccurate and invalid assessment of the code.

3.2 Readability

Our coding schema defined the following aspects of readability: 1) variable names, 2) code comments, 3) layout, and 4) conciseness. The aspects readability, variable names, and layout were not mentioned explicitly by any of the participants. When the participants talked about readability their descriptions remained very general not specifying which aspects were not clear. Some of the statements about readability include: "*When writing a program the learner should try to make the code as simple as possible*", "*This piece of code was not structured and unclear*", and "*I mainly looked at how compact the program was.*" From the transcripts it is clear that teachers do prefer code that is easily readable and understandable. However, the notion of what readable and understandable code is seems dependent on the preference of the individual teacher. Nevertheless, some teachers do try to quantify how readable the code is by looking at its conciseness. Nine of the participants used the number of blocks in the program as a metric for the assessment. As explained in the previous paragraph about functionality, we believe that the number of blocks used in a program can influence both computational efficiency and readability. From the interviews, it is clear that teachers are not aware of this difference. We realize that looking at the number of blocks is an easy way for teachers to get a sense of a student's competence. However, using this metric raises concerns about the validity of the assessment. Giving students feedback based on the number of blocks they use might result in an incorrect representation of the learner's programming skills. The following quote clearly shows an example of how the number of blocks is used as a metric. "*You sometimes have students who use less blocks, in that case I would give them a perfect score (blue). If they use the number of blocks they were allowed to, they get green (represents a score for meeting the requirements), and if they use more blocks, they get orange (failing grade).*"

To summarize, the teachers in our experiment do realize that readability is an important component of the assessment of code. However, most of the assessment of readability is done based on the opinion of the teacher. Concrete aspects like variable names, code comments, and layout were not mentioned by any of the teachers. However, most teachers realize that conciseness is an important factor when writing readable code. Nevertheless, teachers often struggle to quantify this conciseness resulting in an assessment purely based on the number of blocks that are present in the program.

3.3 Application of programming concepts

For the analysis of which programming concepts are used by teachers in their assessment, we only looked at concepts that were present in the programs we presented to them. Those concepts were: sequence, iteration, selection, condition, variables, time delay, input/output. Since the solutions we presented to the teachers did not always contain all concepts and not all concepts were used equally across the different solutions, there is little value in comparing the absolute number of times each concept is mentioned by the teachers. However, it is clear that some concepts are considered more often than others. For example, all but two of the teachers mentioned the concept of iteration, either by explicitly stating that the learners should use iteration or saying they should add a loop to the program. One of the teachers who did not mention iteration mainly assessed the programs based on their function. The only reference they made was: "*If the program does what it should, I think the student should get a maximum score. If it doesn't, then I will look at the intermediary steps he has.*" With these intermediary steps, she refers to the program itself. From the other programming concepts in our coding schema, only selection was mentioned by a majority of the teachers. The rest of the concepts were rarely mentioned explicitly, they were mostly grouped together under the term "*special blocks*" or "*special functions*" The following quote demonstrates that some concepts are more clearly defined for teachers than others: "*For some of the students I wrote down that they know what is happening but still have issues with loops and special functions of programming.*" This shows that all concepts except iteration are grouped into one category.

In summary, most teachers included at least some programming concepts in their assessment. Iteration and selection are concepts most teachers are familiar with and look for during the assessment. However, when giving an assessment of a learner, teachers often fail to identify the specific concept that the learner has not mastered yet. The feedback often only contains references to terms like "*special blocks*" indicating that teachers tend to group concepts together, often resulting in unspecific feedback not tailored to the student's needs.

3.4 Application of algorithmic thinking

From our interviews, it is clear that only looking at the learner's code is insufficient when assessing the application of algorithmic thinking. Teachers rarely mention the skills related to algorithmic thinking we defined in our coding schema. When some of these concepts are mentioned, they are not directly linked to the assessment of the programs we gave them but to their own classroom assessment experience where they have more information for the assessment. The following quote conveys how algorithmic thinking is mentioned by the different teachers: "*When I teach programming to beginners, we first talk about the problem. What is the main problem? What are we looking for? Which blocks do we need? Then we often write down the steps on a piece of paper, step, arrow, step, arrow, ... Do we need a loop somewhere? That really teaches them how to think.*" When algorithmic thinking is mentioned it is related to the process and not the product. Indicating that assessing algorithmic thinking by only looking at code is not considered by the teachers in our experiment.

3.5 Mathematical concepts

Mathematical concepts rarely appear in the transcripts. This might be the nature of the solutions we gave the teachers. These solutions only contain a limited number of mathematical concepts like addition, less/greater than, and Boolean logic. Addition is never mentioned by any of the teachers. Boolean logic is sometimes mentioned implicitly: "*I also look at if the student is able to think logically.*" However, most of the time the teachers refer to one specific condition in one of the solutions where the greater-than symbol should be reversed by the smaller-than symbol without relating this to Boolean logic.

3.6 Execution of testing and debugging

None of the participants talked about debugging during the interview. However, six of the teachers included the concept of testing in their assessment. None of them directly linked it to the score they gave but did include it in the written feedback. This feedback is often related to the functional correctness of the program. When a program does not execute as it should, the teachers think it should be tested more thoroughly. The following quote demonstrates this clearly: "*When I execute the code and it does not work, or the robot only goes forward once, I always wonder if the student actually tested his or her solution before submitting it.*" This link between function and testing often results in written feedback like: "*If you test your program, you can avoid these errors.*" or "*Always check your work!*" This shows that most of the teachers are aware of the importance of testing. However, teachers have to be aware that remarks about more testing are necessary but not sufficient feedback. Just saying a program should be tested more because it does not have the desired function is not a replacement for providing explicit feedback about why it does not have the desired function. The difference is illustrated by the following quotes: "*As feedback I wrote down, you made some careless mistakes however, if you test your programs more those errors will go away.* " and "*Definitely test your programs, try to change the parameters of the delay blocks and make sure everything works correctly.*" The first quote is more general than the second quote which might not be enough to help the student with his or her specific problems.

3.7 Creativity

ElevenTen of the thirteen teachers included creativity in their assessment. The teachers consider two types of coding behaviour as creative. The first type is the addition of an extra element to the program which does not affect the main goal of the exercise. For example, the learner writes his or her name on the lcd-screen while making the robot drive in a square pattern. The extra text on the lcd-screen does not affect whether the robot drives in a square pattern or not. The second type is when the learner solves the problem in an unexpected way, surprising the teacher with his or her solution. For example, for the final question shown in table 2 one of the fictional learners solved the problem by reducing the speed of the robot by a quarter of the original speed instead of limiting the number of collisions explicitly using a counter. All of the ten teachers who took creativity into account in their assessment mentioned examples of the first type of creativity. However, only two of the teachers described examples of the second type of creativity. Moreover, those two teachers don't differentiate between the two types and assess them both in the same way. This seems to indicate that teachers are either unaware of the second type or do not consider them to be sufficiently different to assess them in a different way.

Not only do teachers differ in the types of creativity they assess, they also differ in the value they think creativity has. Some teachers explicitly state that creativity is one of the criteria they always assess when looking at the types of solutions we presented to them. This means that their assessment includes a category for creativity. This category is then one of the aspects determining the students' score. Other teachers do value the creative touch learners add to their program, they reward it by writing a positive note in their formative feedback text to the learner. However, these teachers do not consider it as an element that should influence the student's score. The main argument teachers give for not including creativity is that the questions the learners got do not explicitly ask them to be creative or add their own creative touch. When asking if they would consider increasing a student's score when he or she added a creative touch to the program, one of the teachers answered: "*No, I don't think a student should get a lower score just because he or she did not add anything extra to the program.*" Besides the teachers previously described who generally consider the creative elements added by learners as a positive aspect of the learner's abilities, some other teachers consider these aspects as detrimental to the overall quality of the program. Consequently, these teachers subtract from a student's score when there are elements in the program which are not explicitly asked for in the assignment. One of these teachers put forward the following argument for his method of assessment: "*Yeah, well, its cool to add those things but when you work for a company and you put a bunch of Easter eggs into the code, the customer will not understand. They have to learn that they don't program for themselves but for someone else.*" Interestingly, this same teacher negatively scores both types of creativity described in the previous paragraph. The argument he gives for negatively scoring the first type seems sound. However, he applies the same reasoning to the second type of creativity. When students solve a problem in a different way than the other students did while still reaching a correct result, it is punished in the same way as when they add extra elements which were not stated in the assignment. This strengthens the idea that teachers are not aware of different types of creativity.

To summarize, while many teachers consider creativity when assessing programming, they do not seem to be aware of the different ways creativity can manifest itself in a program. Additionally, even though many teachers take creativity into account, the value assigned to creativity varies a lot from one teacher to the next.

3.8 Personal aspects

Even though we do not provide any information about the personality of the learners during the experiment, many teachers (N=11) do try to imagine a personality that fits the students. Some teachers explicitly state that they need this information to make an accurate assessment of the code: "*I think it is hard to write down feedback for these students. I would have to have more information about them. Is it a stronger or weaker student? If it is a weaker student and he has 6/10 I would write that he did a good job but if it is a stronger student I would write that he should have done better.*" Other teachers go even further and attach personal aspects to the learner based on the solutions they handed in: *Student 5 is someone, I would not say he has autism however, he likes to see everything written out in steps.*" Some teachers say there are certain aspects they usually use as criteria for the assessment of programming which are impossible in this context. These criteria are related to what learners do during programming class and not what they produce. Some examples are: the amount of effort they put in a solution, if they help other students, and how motivated they are.

3.9 Link between assessment strategy and teaching experience.

To elucidate the link between teaching experience and assessment strategy, we divided the teachers into four groups: (1) Teachers without previous programming teaching experience ($N = 2$). (2) Teachers with one to five years of programming teaching experience ($N = 2$). (3) Teachers with more than five years of programming teaching experience ($N = 5$). (4) Teachers who did not wish to disclose their experience for processing in our experiment ($N = 4$). We first describe the background of the teachers in groups one to three as well as the intra-group similarities and differences To assess the relation between teaching experience and assessment strategy, we summarized the intra-group similarities and differences for the experience groups described in section 3 . Thereafter, we look at the inter-group similarities and differences to get a sense of how these teaching strategies might evolve with experience.

No experience teaching programming. Two of the participants indicated that they had no prior experience teaching programming. Additionally, they both indicated to have little experience with programming in general, scoring themselves one out of ten and three out of ten. Up till now, these teachers primarily taught mathematics. However, they participated in the experiment because they might bewould be required to teach programming in the future as a result of changes to the curriculum. Even though their experience with teaching programming was the same, one had a lot more teaching experience than the other, 25 years as opposed to 3 years. Both teachers considered functional correctness as the main criterium for their assessment. Nevertheless, they are aware that other aspects like programming concepts and efficiency might be important for the assessment. However, they have a hard time defining these aspects and refer to them as "*intermediate steps*"or "*unnecessary steps*". It is clear from the interactions that these teachers are aware that their knowledge about programming is insufficient to accurately assess these *intermediate or unnecessary steps*. Interestingly, they cope with this differently. One of the teachers (with three years of teaching experience) decided to only assessscore the programs based on their functional correctness using a list of criteria like the length of time a text is shown on the screen or the angles the riding robot turns when driving a square pattern on the floor. The other teacher (with 25 years of teaching experience) tries to identify unnecessary blocks by comparing the solutions of all five learners. This information is then used to identify redundant blocks in certain solutions. This teacher also considers other aspects like creativity and mathematical concepts when determining a score. By using the strategy of comparing solutions, this teacher reaches a fairly accurate ranking of the different students. However, the personal feedback learners receive remains vague, for example, *Sometimes your programs can be a bit simpler* or *Be aware of the details*.

These results indicate that having no experience with teaching programming as well as programming in general results in assessment mainly based on functional criteria which are sometimes arbitrarily chosen. However, having limited programming experience does not necessarily result in a purely functional assessment. The second teacher tries to get more insight into the solutions by using more general assessment strategies. The difference between these two teachers might be a result of their general teaching and assessment experience, allowing the second teacher to apply more general assessment techniques to programming problems.

One to five years of experience teaching programming. Two teachers had been teaching programming for less than five years (three years and four years), both less than two hours a week. The main subject they teach is "techniek", which can be loosely translated to "technology". "Techniek"is a STEM subject in the first two years of secondary school covering multiple STEM-related topics. The two teachers had similar teaching experience in general (20 and 21 years). Moreover, they rated their general programming experience as six out of ten and seven out of ten.

These two teachers used similar strategies when assessing the solutions presented to them. Like the teachers without any programming experience, these teachers start by looking at functional criteria like *Does the riding robot stop after driving one square pattern on the ground or does it keep going*. However, both teachers elaborate more on other aspects, mainly conciseness and programming concepts. They explicitly state that they look at loops and if-statements when judging a solution. The use of these concepts is often related to conciseness, however, sometimes they know a program isn't concise based on the number of blocks but are unable to explain why.

One of the teachers does not go beyond the strategy discussed above except for briefly mentioning that extra creative elements added to the program were not considered in the assessment. The other teacher does elaborate on other aspects like computational thinking and personal aspects. This teacher attaches a lot of value to the programming process and less to the result. This teacher also wants to know if the learners enjoy programming since this is one of the main criteria they use to formulate advice for future study choices.

Having one to five years of programming teaching experience seems sufficient to be able to assess some basic programming concepts. This allows teachers to give more accurate feedback about the concepts the students have not mastered yet. Nevertheless, these teachers also focus on a set of semi-arbitrarily functional correctness criteria, mainly focus on the number of blocks to determine the compactness of a program, and mostly ignore criteria like creativity, efficiency, and testing and debugging.

More than five years of experience teaching programming. Five teachers indicated they had more than five years of experience teaching programming. However, the number of hours a week they teach programming varies. Three participants teach programming less than two hours a week, one participant teaches programming between two and five hours a week, and one teacher teaches programming more than ten hours a week. All participants in this group gave themselves a score of seven or eight out of ten for general programming experience. The number of years they have been teaching varies from 5 to 33 years. Three teachers explicitly teach a programming course, the other two teach STEM.

The transcripts of these teachers reveal that these teachers have a much broader view on the assessment than the teachers in the first two groups. Almost all teachers in this group consider functionality, readability, programming concepts, testing , and creativity during their assessment. Most of them look at personal aspects and creativity as well. Mathematical concepts and algorithmic thinking were only mentioned oncetwiceand none of the teachers said anything about computational thinking. Overall, their assessment is more balanced than the teachers in the previous two groups. Instead of mainly looking if the functional requirements of the program are correct and afterward adapting their valuation based on another concept like compactness, these teachers first consider the different aspects and then value the solution based on these different criteria. Even though these teachers mostly consider the same concepts, how each of these concepts is valued varies between the different teachers. One example is creativity, some teachers do not consider creativity in their assessment, others want to encourage it by scoring creative solutions higher, and one of the teachers deducts points for creative solutions because they do not conform to the standard way of solving the problem. Another concept with different valuations by the teachers in this group is functional correctness. Some teachers deduct a significant amount of points from the final score when the program does not function as requested while others deduct only one or two points and focus more on other aspects like the correct use of programming concepts. From our interviews, it is not clear why some teachers attach a different value to these concepts. However, the transcriptsinterviews indicate that the specific context in which the teachers works impacts their valuation. This context can vary significantly across the educational system. In our experimental group, we could identify two variations in the context that influence the way teachers assess code. The first is if the teacher is used to teach in an open or closed problem context. Teachers working in an open problem context (usually linked to STEM education) often attach less value to the functional requirements of the program than teachers mostly working with closed-ended problems. This is illustrated by the following statement made by one of the teachers who assessed the solutions in our experiment with a focus on functionality: "*If I had to assess these programs in a STEM context, I would look at if they analyzed the problem correctly, do they use the correct control structures, did they use variables, is the solution readable, can you immediately understand the program, how much time did it take, and did they add improvements.*"Another aspect that seems to influence the context is the cohort these teachers usually work with. In theory, the first two years of secondary school in Flanders should have similar student cohorts. However, the average socioeconomic status [1] of learners varies across different schools and can have an influence on the motivation of learners. This might encourage teachers to focus more on what learners do well, like using the correct programming concepts, add additional creative elements and correct functionality of specific elements in the program. In contrast, teachers who are confronted less with these issues might focus more on adhering to strict assessment criteria. These observations are interesting, however, they require a lot more research in order to be confirmed.

Inter group comparison. By analyzing these three groups, a pattern towards a more broad assessment seems to appear. Inexperienced teachers mainly focus on functional correctness. However, using general assessment techniques like comparing solutions, they are able to more-or-less rank the students according to their abilities. Nevertheless, their personal feedback written is superficial rendering it less useful for the learners. Teachers with moderate experience use a similar technique to the teachers without experience but do show a better understanding of some programming concepts like loops and if-statements. Consequently, their feedback is more concrete. However, they pay less attention to the other concepts linked to programming. Finally, experienced teachers have a much broader view of the assessment of code. They consider a lot more concepts during the assessment and do not necessarily see function as the main criterium for their assessment. However, the strategies for valuing the different aspects differ a lot between these teachers. A first analysis indicates that this valuation is influenced by the context in which these teachers work. This trend towards a broader assessment seems to be supported by the average concepts considered in each group shown in table 6. However, we do not believe this quantitative comparison has much (if any) value since our experimental groups were small. Nevertheless, during the interviews, it became clear that more experienced teachers seem to have a deeper insight into the assessment criteria for programming.

4 DISCUSSION

4.1 Primary assessment strategies

The first two research questions we tried to answer with this experiment are which criteria teachers consider when assessing code and how they value these different aspects when assessing code. Our results have shown that the answers to these questions are closely intertwined. Consequently, discussing them separately is of little value. From our analysis it is clear that most teachers consider functionality as one of the most important aspects to base their assessment on. Specifically, functional correctness seems to be the most important facet related to functionality, especially teachers with limited confidence and experience seem to focus on functional correctness. Other aspects related to function like efficiency and usability are considered less often and when they are considered, teachers are not aware that they are assessing a specific concept related to programming. Even though most teachers attach a lot of value to program functionality, some teachers primarily focus on the application of programming concepts. They define which concepts should be used in each exercise and assess if that concept was understood by looking at the code the learner wrote. If the concepts were applied correctly, the students will get a higher score independent of whether the complete program was functionally correct or not. The teachers in this group indicated that they valued the assessment of the process over the assessment of the product which would explain why they attach less value to the functional correctness of the programs.

4.2 Secondary assessment strategies

Despite that most teachers predominantly consider the functionality of the program for their assessment, some of the teachers also look at the correct usage of programming concepts in a second stage of the assessment. Confident and experienced teachers seem to prefer this method of assessment. However, these teachers are still unaware of some of the aspects they assess and attach a different value to them. From the aspects related to programming concepts we defined in our coding schema, iteration and selection were mentioned by many but not all teachers. Those concepts seem to be the ones that most teachers are familiar with, other concepts are often grouped together into one category. Besides functionality and programming concepts, most teachers also consider readability in some form during their assessment. However, readability is never deliberately cited as a criterion for the assessment. It is mostly assessed implicitly as part of the clarity of the program. If the teacher can easily understand the program, it is considered readable. Specific indicators of readable code like clear variable names, code comments, and layout are never explicitly mentioned by the teachers.

4.3 Creativity

Other aspects like creativity would also benefit of a clear description of how it can manifest itself in a program. Our results have shown that many teachers see creativity as a part of the assessment of programming. However, not all of them define and value creativity in the same way. Some only look at the extra elements students added to the program while others go further and also attribute creativity to the way a certain problem is solved.

4.4 Personal aspects

The final assessment criterion we defined is the learner's personal aspects. We recognize that our experiment did not provide sufficient information about the learners to get an accurate image of the personalities of the different learners. However, our interviews have shown that these personal aspects are an important element teachers use when assessing programming exercises. Many teachers either construct a personality in their head based on the solutions they have, others explicitly state that they need this information to make an accurate assessment. We believe it is valuable that teachers try to use a more personalized approach when assessing programming problems. However, we believe that solutions to programming problems are a weak indicator of the personality of the learner. Consequently, constructing an image of a learner personality based on solutions to programming problems will lead to inaccurate and invalid assessment. Moreover, since our data indicates that many teachers only consider a subset of all possible aspect of programming for their assessment, using this subset of criteria to construct a personality will result in an inaccurate representation of a learner's personality.

4.5 Lesser-used concepts

In the previous paragraphs we discussed the main aspects teachers consider when assessing code. However, some other concepts get less attention. Based on our interviews, we believe that the application of algorithmic thinking and the execution of testing an debugging are less prevalent in the assessment teachers make since these concepts are more related to the process of programming and less to the end result. Since our experiment was set up focusing on the assessment of the end result, these concepts were considered less frequently. Nevertheless, some teachers do try to include them as part of their assessment. References to mathematical concepts rarely appear in our transcripts. We assume this is because the exercises contained only a limited number of these concepts and the concepts that were used were relatively simple. Additionally, mathematical concepts like addition are mostly part of the learners' prior knowledge which is probably why the teachers in our experiment did not explicitly state it as an assessment criterion.

4.6 Assessment pitfalls

Our analysis revealed some pitfalls in the assessment some teachers use. For example, one One specific aspect related to readability that teachers often include in the assessment is the number of blocks learners use. However, teachers are not aware of which underlying concepts this number of blocks can represent. It can either impact the readability or the computational efficiency. We suspect teachers take the number of blocks a program has as a criterion because online platforms like Google Blockly or code.org use the same metric to determine if the learners successfully solved one of the programming problems. However, the problems used on these platforms are created by experts in a narrowly defined environment. Consequently, they can design the challenges in such a way that the best solution is also the solution with the least amount of blocks. However, in more open contexts like the one in our experiment this is not the case. Others have also shown that using certain heuristics during assessment, like the number of code blocks or the number of lines in a function, can have a negative effect on the quality of the assessment. Applying these metrics does not necessarily result in a correct assessment [25]. It is important that teachers are made aware of this to improve the validity of their assessment and facilitate the assessment of exercises they create themselves.

From the analysis for the criteria functionality, readability, and programming concepts, it is clear that teachers need a better understanding of the different aspects of programming in order to improve the accuracy and validity of their assessments. Some concepts like usability, efficiency, variable naming, code comments, layout, and certain programming concepts are either never considered or not specified sufficiently. From our interviews it is clear that many teachers want to improve their assessments but are unable to do so because they lack the required framework to reason about why certain ways of programming are good or bad. The last question in the interview asked teachers what would help them to facilitate the assessment of code. Some teachers said they needed more content knowledge, however, most teachers said they wanted a checklist, rubric, or clear set of criteria to base their assessment on. Nevertheless, this list of criteria should not be too limiting and allow them to create an assessment strategy suited to their needs. Consequently, we believe that improving the awareness of the different programming concepts described in this paper is required to improve the quality and validity of the assessment of code.

4.7 Effect of teaching experience

The third research question we tried to answer was about the relation between teaching experience and assessment strategy. Our results show the first indications of the relation between teaching experience and assessment strategies. It appears that more experience results in a broader range of assessment criteria. However, these results are only an indication of a possible trend. Further research is required to see if this trend is present in a larger population. The idea that experience and the number of concepts considered during the assessment are linked, strengthens the idea that teaching teachers about the different criteria can have a positive effect on their assessments.

4.8 Final thoughts

Previous work has shown that teacher educators struggle to improve the way teachers grade and report student learning progress [20]. Nevertheless, the application assessment strategies for programming remain under-explored [28]. Additionally, previous efforts to create professional development programs for programming teachers often have limited attention for assessment [21]. Our results indicate that the assessment of code should be an integral part of both teacher education as well as teacher professional development programs. Teachers should be made aware of the difference between the purpose and the format of the assessment [31] and understand which formats are useful for certain purposes. Our results show that some aspects of programming like function, readability, programming concepts, and creativity are often considered when assessing code. However, teachers are often not aware that they are assessing these concepts. This leads to inaccurate and invalid feedback.This can lead to inaccurate or invalid feedback. For example, teachers might say a learner's code deservers a perfect score because it is functionally correct while it is not compact because no control structures are used. Our results also showseem to indicate that more experienced teachers consider a lot more aspects related to code than teachers with less experience. Additionally, the number of concepts considered seems to increase with experience. This indicates that teachers aremight be discovering these assessment criteria over the years and adding them to their assessment portfolio. The framework we propose could help teachers to get a better understanding of the different assessment dimensions related to code right away. This understanding should guide the assessment decisions teachers make, leading to improved accuracy and validity.

4.9 Future work

This small scale case study shows many different pathways for future research. A larger scale quantitative assessment of the relation between teaching experience and assessment concepts is a straightforward next step. Other topics, like if explaining the categories in our assessment schema to teachers has an effect on their assessment strategies or if the division between assessing based on the programming process versus the end result can be identified on a larger scale. Additionally, adapting and validating our coding schema in different contexts can be of value for teacher educators.

5 CONCERNS TO VALIDITY

Our results reveal multiple issues with the way teachers assess code. However, even though this study contained an in depth analysis of the assessment techniques of different teachers, because our sample size is relatively small and the teachers were self-selected, our results are not necessarily generalizable to all teachers. All teachers who participated in our experiment teach programming in the first two years of secondary school which requires them to reach the same educational goals. However, they have different backgrounds. Some have more experience teaching practical subjects while others have more experience with theoretical subjects. The target audience the teachers usually have in their classes also differs depending on the cultural and social characteristics of the school where they teach. For example, some teachers have more experience with students learning disabilities while others have more experience of whom Dutch is not their native language. Moreover, the way we presented the assessment to the teachers and the specific tools used during the experiment might not align with the assessment techniques these teachers usually use in practice. In their own classroom, they have more freedom to perform the assessment which might result in more assessment criteria being covered. Furthermore, the participants might have a different level of familiarity with the specific type of programming problems we presented in this work. Having more experience with visual programming languages for physical systems might influence the assessment results. Despite these variations in our experimental group, the group does not cover all the possible experience teachers can have, limiting the generalizability of our results. Exploring the assessment methods for specific teacher profiles requires additional research. Nevertheless, our results give an indication of the strategies that are used for assessing code in the first two years of secondary school of Flemish education.

6 CONCLUSION

Overall, considering the narrowly defined context for our experiment, teachers do consider multiple different aspects related to programming during their assessment. Nevertheless, most of the teachers mainly look at functional correctness. Additionally, many teachers are not aware of the different concepts they assess, the concepts they are aware of are often not clearly defined. This lack of insight into the different aspects of programming and how these aspects can be assessed results in threats to the accuracy and validity of the assessment and may have a negative effect on the different learning outcomes related to programming. Ideally, the feedback provided by a teacher should help learners get a better understanding of how to write high quality code. This is not the case for some of the feedback teachers gave during the experiment. For example, teachers who only assess the code based on the functional result often miss other issues with the code resulting in similar feedback for code of differing quality. Moreover, other teachers make assumptions about the understanding of certain programming concepts. For example, one teacher said the following about a learner who, in the exercise where the learners had to make a robot drive a square pattern on the floor, used the infinite loop in the setup-loop structure as the basis for the repetition in their application: *The code is in an infinite loop so the robot doesn't stop. They should have used a loop that counts down. They clearly have not mastered that type of loop yet.* This feedback is inaccurate because: 1) The assignment does not specify the number of repetitions. 2) Not using a certain coding construct does not necessarily mean the learner does not understand it. Not having a mental overview of the different aspects related to programming also prevents teachers from reasoning about which assessment criteria are best used in certain situations. The ability to reason about assessment criteria would be especially beneficial for beginner teachers, helping them to provide better feedback to their students. In our opinion, teachers have to be made more aware of all possible aspects of programming and how these aspects can be assessed.

This paper defines a framework for the assessment of programming by combining the different aspects previous work has shown to be relevant when assessing code. Our framework covers a wide range of topics, however, it is limited to the concepts which were relevant to the context of our experiment. In this paper, we identified how the assessment teachers make in practice adheres to the proposed framework. Moreover, using this framework we revealed some inconsistencies and threats to validity in the assessment practice like when teachers give negative feedback to learners with an alternative solution strategy with the argumentation they should not add easter eggs to ther programs. However, assessment practice has also given insight into the relevance of certain assessment criteria within our framework. Additionally, we have showour results indicate that a teachers' understanding of different concepts grows over time with teaching experience. We are convinced the proposed framework can be used as a reference, however, it should be adapted to the specific learning context for the assessment. Consequently, future work should explore how the proposed framework should be extended or adapted to fit different learning contexts. Moreover, additional research is required to confirm that creating awareness about the different aspects of programming actually leads to improved assessment. Nevertheless, our framework can already be used as a reference by teachers and teacher educators to help improve insight into the accuracy and validity of their assessments. Moreover, the specific problems we have identified in this paper can be used to inform teachers about possible good or bad practices when assessing code.

REFERENCES

[1] Elizabeth H Baker. 2014. Socioeconomic status, definition. *The Wiley Blackwell encyclopedia of health, illness, behavior, and society* (2014), 2210–2214.

[2] Sarah Elsie Baker and Rosalind Edwards. 2012. How many qualitative interviews is enough. (2012).

[3] Satabdi Basu. 2019. Using Rubrics Integrating Design and Coding to Assess Middle School Students' Open-ended Block-based Programming Projects. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1211–1217.

[4] Satabdi Basu, Daisy Rutstein, Yuning Xu, and Linda Shear. 2020. A principled approach to designing a computational thinking practices assessment for early grades. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 912–918.

[5] Rosemary Pessoa Borges, Pablo Roberto Fernandes Oliveira, Romulo Galdino Rocha Lima, and Rommel Wladimir Lima. 2018. A systematic review of literature on methodologies, practices, and tools for programming teaching. *IEEE Latin America Transactions* 16, 5 (2018), 1468–1475.

[6] Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. 2018. "I know it when I see it" Perceptions of Code Quality: ITiCSE'17 Working Group Report. In *Proceedings of the 2017 ITiCSE Conference on Working Group Reports*. 70–85.

[7] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[8] Karen Brennan and Mitchel Resnick. 2012. New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, Vol. 1. 25.

[9] John L Campbell, Charles Quincy, Jordan Osserman, and Ove K Pedersen. 2013. Coding in-depth semistructured interviews: Problems of unitization and intercoder reliability and agreement. *Sociological methods & research* 42, 3 (2013), 294–320.

[10] Mehmet Celepkolu, Erin O'Halloran, and Kristy Elizabeth Boyer. 2020. Upper Elementary and Middle Grade Teachers' Perceptions, Concerns, and Goals for Integrating CS into Classrooms. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 965–970.

[11] Guanhua Chen, Ji Shen, Lauren Barth-Cohen, Shiyan Jiang, Xiaoting Huang, and Moataz Eltoukhy. 2017. Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & Education* 109 (2017), 162–175.

[12] Alan Cooper et al. 2004. *The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity*. Vol. 2. Sams Indianapolis.

[13] Nicole M Deterding and Mary C Waters. 2021. Flexible coding of in-depth interviews: A twenty-first-century approach. *Sociological methods & research* 50, 2 (2021), 708–739.

[14] Yihuan Dong, Veronica Cateté, Nicholas Lytle, Amy Isvik, Tiffany Barnes, Robin Jocius, Jennifer Albert, Deepti Joshi, Richard Robinson, and Ashley Andrews. 2019. Infusing computing: Analyzing teacher programming products in K-12 computational thinking professional development. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. 278–284.

[15] Svetlana V Drachova, Jason O Hallstrom, Joseph E Hollingsworth, Joan Krone, Rich Pak, and Murali Sitaraman. 2015. Teaching mathematical reasoning principles for software correctness and its assessment. *ACM Transactions on Computing Education (TOCE)* 15, 3 (2015), 1–22.

[16] Katrina Falkner, Sue Sentance, Rebecca Vivian, Sarah Barksdale, Leonard Busuttil, Elizabeth Cole, Christine Liebe, Francesco Maiorana, Monica M McGill, and Keith Quille. 2019. An international comparison of k-12 computer science education intended and enacted curricula. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research*. 1–10.

[17] Patricia I Fusch and Lawrence R Ness. 2015. Are we there yet? Data saturation in qualitative research. *The qualitative report* 20, 9 (2015), 1408.

[18] Gerald Futschek. 2006. Algorithmic thinking: the key for understanding computer science. In *International conference on informatics in secondary schools-evolution and perspectives*. Springer, 159–168.

[19] Shuchi Grover and Satabdi Basu. 2017. Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and boolean logic. In *Proceedings of the 2017 ACM SIGCSE technical symposium on computer science education*. 267–272.

[20] Thomas R Guskey and Laura J Link. 2019. Exploring the factors teachers consider in determining students' grades. *Assessment in Education: Principles, Policy & Practice* 26, 3 (2019), 303–320.

[21] Emily Hestness, Diane Jass Ketelhut, J Randy McGinnis, Jandelyn Plane, Bonnie Razler, Kelly Mills, Lautaro Cabrera, and Elias Gonzalez. 2018. Computational thinking professional development for elementary science educators: Examining the design process. In *Society for Information Technology & Teacher Education International Conference*. Association for the Advancement of Computing in Education (AACE), 1904–1912.

[22] Ken Kahn and Harriette L Spiegel. 1999. The role of computer programming in education. *Journal of Educational Technology & Society* 2, 4 (1999), 6–9.

[23] Hanna Kallio, Anna-Maija Pietilä, Martin Johnson, and Mari Kangasniemi. 2016. Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. *Journal of advanced nursing* 72, 12 (2016), 2954–2965.

[24] ChanMin Kim, Jiangmei Yuan, Lucas Vasconcelos, Minyoung Shin, and Roger B Hill. 2018. Debugging during block-based programming. *Instructional Science* 46, 5 (2018), 767–787.

[25] Diana Kirk, Ewan Tempero, Andrew Luxton-Reilly, and Tyne Crow. 2020. High School Teachers' Understanding of Code Style. In *Koli Calling'20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research*. 1–10.

[26] Özgen Korkmaz, Recep Çakir, and M Yaşar Özden. 2017. A validity and reliability study of the computational thinking scales (CTS). *Computers in human behavior* 72 (2017), 558–569.

[27] Michael J Lee, Faezeh Bahmani, Irwin Kwan, Jilian LaFerte, Polina Charters, Amber Horvath, Fanny Luor, Jill Cao, Catherine Law, Michael Beswetherick, et al. 2014. Principles of a debugging-first puzzle game for computing education. In *2014 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, 57–64.

[28] Linda Mannila, Fredrik Heintz, Susanne Kjällander, and Anna Åkerfeldt. 2020. Programming in primary education: towards a research based assessment framework. In *Proceedings of the 15th Workshop on Primary and Secondary Computing Education*. 1–10.

[29] Jesús Moreno-León, Gregorio Robles, and Marcos Román-González. 2015. Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista de Educación a Distancia* 46 (2015), 1–23.

[30] Laurie Murphy, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. Debugging: the good, the bad, and the quirky–a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin* 40, 1 (2008), 163–167.

[31] Guri A Nortvedt and Nils Buchholtz. 2018. Assessment in mathematics education: responding to issues regarding methodology, policy, and equity. *ZDM* 50, 4 (2018), 555–570.

[32] Joel D Olson, Chad McAllister, Lynn D Grinnell, Kimberly Gehrke Walters, and Frank Appunn. 2016. Applying Constant Comparative Method with Multiple Investigators and Inter-Coder Reliability. *Qualitative Report* 21, 1 (2016).

[33] Marcos Román-González, Juan-Carlos Pérez-González, and Carmen Jiménez-Fernández. 2017. Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. *Computers in Human Behavior* 72 (2017), 678–691.

[34] Mark A Runco and Selcuk Acar. 2012. Divergent thinking as an indicator of creative potential. *Creativity research journal* 24, 1 (2012), 66–75.

[35] Mark A Runco and Garrett J Jaeger. 2012. The standard definition of creativity. *Creativity research journal* 24, 1 (2012), 92–96.

[36] Sameh Said-Metwaly, Wim Van den Noortgate, and Eva Kyndt. 2017. Approaches to measuring creativity: A systematic literature review. *Creativity. Theories–Research-Applications* 4, 2 (2017), 238–275.

[37] Ronny Scherer, Fazilat Siddiq, and Bárbara Sánchez Viveros. 2019. The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology* 111, 5 (2019), 764.

[38] Young-Ho Seo and Jong-Hoon Kim. 2016. Analyzing the effects of coding education through pair programming for the computational thinking and creativity of elementary school students. *Indian Journal of Science and Technology* 9, 46 (2016), 1–5.

[39] Jocelyn Simmonds, Francisco J Gutierrez, Cecilia Casanova, Cecilia Sotomayor, and Nancy Hitschfeld. 2019. A Teacher Workshop for Introducing Computational Thinking in Rural and Vulnerable Environments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1143–1149.

[40] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2014. Towards an empirically validated model for assessment of code quality. In *Proceedings of the 14th Koli Calling international conference on computing education research*. 99–108.

[41] Martijn Stegeman, Erik Barendsen, and Sjaak Smetsers. 2016. Designing a rubric for feedback on code quality in programming courses. In *Proceedings of the 16th Koli Calling International Conference on Computing Education Research*. 160–164.

[42] Xiaodan Tang, Yue Yin, Qiao Lin, Roxana Hadad, and Xiaoming Zhai. 2020. Assessing computational thinking: A systematic review of empirical studies. *Computers & Education* 148 (2020), 103798.

[43] Donald J Treffinger, Grover C Young, Edwin C Selby, and Cindy Shepardson. 2002. Assessing Creativity: A Guide for Educators. *National Research Center on the Gifted and Talented* (2002).

[44] Onderwijs Vlaanderen. 1999. *Onderwijsdoelen - Resultaten.* https://onderwijsdoelen.be/uitgangspunten/4814

[45] Anna Yström, Lena Peterson, Björn von Sydow, and Johan Malmqvist. 2010. Using Personas to Guide Education Needs Analysis and Program Design. In *Proceedings of 6th International CDIO Conference, Montreal, Canada.*