



UvA-DARE (Digital Academic Repository)

Computerized adaptive assessment of understanding of programming concepts in primary school children

Hogenboom, S.A.M.; Hermans, F.F.J.; Van der Maas, H.L.J.

DOI

[10.1080/08993408.2021.1914461](https://doi.org/10.1080/08993408.2021.1914461)

Publication date

2022

Document Version

Final published version

Published in

Computer Science Education

License

CC BY-NC-ND

[Link to publication](#)

Citation for published version (APA):

Hogenboom, S. A. M., Hermans, F. F. J., & Van der Maas, H. L. J. (2022). Computerized adaptive assessment of understanding of programming concepts in primary school children. *Computer Science Education*, 32(4), 418-448.
<https://doi.org/10.1080/08993408.2021.1914461>

General rights




It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (<https://dare.uva.nl>)

Computerized adaptive assessment of understanding of programming concepts in primary school children

Sally A. M. Hogenboom ^a, Felienne F. J. Hermans ^b and Han L. J. Van der Maas ^c

^aInstitute for Logic, Language, and Computation, Faculty of Humanities, University of Amsterdam, Amsterdam, Amsterdam, The Netherlands; ^bLeiden Institute of Advance Computer Sciences, Faculty of Science, University of Leiden, Leiden, CA, The Netherlands; ^cDepartment of Psychological Methods, Faculty of Social and Behavioral Sciences, University of Amsterdam, Amsterdam, Amsterdam, The Netherlands

ABSTRACT

Background and Context: Valid assessment of understanding of programming concepts in primary school children is essential to implement and improve programming education.

Objective: We developed and validated the Computerized Adaptive Programming Concepts Test (CAPCT) with a novel application of Item Response Theory. The CAPCT is a web-based and resource-efficient adaptive assessment of 4489 questions measuring: the understanding of basic sequences, loops, conditions (if & if-else statements), debugging, multiple agents, procedures, and the ability to generalize to a new syntax.

Method: Data was collected through an existing online adaptive practice and monitoring system called Math Garden. We collected 14 million responses from 93,341 Dutch children (ages 4 - 13).

Findings: The CAPCT demonstrated good psychometric qualities because 75% of the variance in question difficulty was explained by differences in item characteristics. The CAPCT demonstrated robustness against adding new participants and adding new items. Differences in player ability (i.e., understanding of CS concepts) were due to differences in age, gender, the number of items played, and prior mathematical ability.

Implications: The CAPCT may be used by teachers to identify the level of programming concept understanding of their pupils, while researchers may use the CAPCT to construct and validate effective teaching resources.

ARTICLE HISTORY


Received 18 September 2020
Accepted 6 April 2021

KEYWORDS

Computational thinking; programming; elementary education; computerized assessment

1. Introduction

Companies, educators, and policymakers alike call for the inclusion of Computer Science (CS), Programming, and Computational Thinking (CT) education in (primary) school curricula (Google for Education, 2020; SLO, 2015; Wing, 2006). Teaching children to program and solve problems computationally prepares them for the future job market (World Economic Forum, 2016; sec. Skill Stability), and increases logical and abstract thinking, problem-solving ability, and creativity (Cao et al., 2015; Durak & Saritepeci,

CONTACT Sally A. M. Hogenboom  sally.hogenboom@gmail.com  Institute for Logic, Language and Computation (ILLC), University of Amsterdam, Science Park 107, 1098 XG Amsterdam, The Netherlands

This article has been republished with minor changes. These changes do not impact the academic content of the article.

© 2021 The Author(s). Published by Informa UK Limited, trading as Taylor & Francis Group.

This is an Open Access article distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives License (<http://creativecommons.org/licenses/by-nc-nd/4.0/>), which permits non-commercial re-use, distribution, and reproduction in any medium, provided the original work is properly cited, and is not altered, transformed, or built upon in any way.

2018). CS and CT both require the ability to implement common programming concepts such as loops and conditionals. CS and CT education promote use of functioning algorithms, but also more conceptual use without computers (e.g., CS Unplugged, 2020). Both fields therefore share the need for teaching and evaluating the extent to which pupils understand common programming concepts.

Although efforts have been made to assess programming comprehension in primary school children; valid and easy to implement assessments are currently lacking (Grover & Pea, 2013; International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA), 2011; Tang et al., 2020; Voogt et al., 2017). The lack of valid assessments prevents researchers from validating and comparing the vast range of education tools and curricula currently being created (e.g., Chakarov et al., 2019).

This paper covers the construction, data-collection, and internal validation of a new adaptive assessment of programming comprehension suited for administration at (Dutch) primary schools (ages 4–13). We developed the Computerized Adaptive Programming Concepts Test (CAPCT) to measure comprehension of basic sequences, loops, if-statements, if-else statements, procedures, multiple agents, debugging, and generalization to a different programming syntax. The CAPCT is part of an existing adaptive online measurement environment, called Math Garden, described in more detail in [section 1.2](#)

The Math Garden environment has great practical value as an educational tool (simultaneously practicing and measuring ability), and scientific value through the collection of high quantities of rich data on children's cognitive development (for overviews see Brinkhuis et al., 2018; Hofman et al., 2018). Over 4000 items were created and internally validated, advancing the field of CS and CT education in multiple ways. First, teachers may use the student-reports in Math Garden to gain insight in the individual abilities of their pupils. The reports indicate where pupils stand compared to the content and their peers. Math Garden also solves the issues of longitudinal measurement, which is an area that needs further development in (primary school) CS education (Vivian et al., 2020). Second, the freely available itemset includes the difficulty estimates for all items.¹ This may allow future research to validate non-adaptive (paper-and-pencil) tests that can measure ability of students that do not have access to Math Garden. These tests could then be used for informal testing in classrooms (which CS concept requires further instruction?) or as pre- and post-tests during research (e.g., the effectivity of lesson plans).

1.1. Assessment of programming concepts comprehension

A wide range of methods have been used to assess Computer Science (CS) comprehension, programming concept understanding, or Computational Thinking (CT; for overviews see: Román-González et al., 2019; Tang et al., 2020). However, three main measurement characteristics prevent extensive use across the field (Grover & Pea, 2013; Voogt et al., 2017).

First, assessments are often highly resource intensive. For example, behavioral observations (Bers, 2010; Sáez-López et al., 2019), interviews (Zhong et al., 2016), peer-assessment (Basogain et al., 2018), and think-out-loud procedures (Atmatzidou & Demetriadis, 2016) all require one-to-one contact or extensive scoring procedures. For researchers, this may merely be an inconvenience, yet for teachers it makes monitoring student ability rather problematic. Automated assessments of code are less resource

intensive (e.g., Dr. Scratch), but focus on the product – which costs time to produce – rather than directly evaluating concept understanding (Moreno León et al., 2015; Román-González et al., 2019).

The second problem with current assessments is the lack of validation studies and specific evaluation criteria (Tang et al., 2020). Tang and colleagues found that 49% of the 96 assessments in their systematic review included reliability evidence, with only 18% including validity evidence. The lack of internal validation studies, let alone convergent or divergent validity studies, makes it difficult to estimate how existing measurements of CS/CT understanding are related.

Finally, assessments were often created for specific software (e.g., Scratch; Franklin et al., 2013), programming languages, teaching methods (Chang, 2014), or age-groups (Tang et al., 2020); limiting the use across curricula and educational tools or generalization between studies.

Some notable exceptions to the issues mentioned above are the Computational Thinking test (CTt; Román-González, 2015; Román-González et al., 2017, 2018), the assessment created by Grover and Basu (2017), the Middle Grades Computer Science Concept Inventory (MG-CSCI; Rachmatullah et al., 2020), the Lean Computational Thinking Abilities Assessment (LCTAA; Wiebe et al., 2019), the Beginners Computational Thinking Test (BCCT; Zapata-Caceres et al., 2020), and the PESS framework (Mannila et al., 2020).

First, these tests are short and consist of multiple-choice items – reducing the resources needed for assessment and scoring. The LCTAA – partially based on the Bebras competition (Dagienè & Sentance, 2016)– contains some open-answer items, but clearly defined correct answers still allow easy scoring.

The abovementioned tests are also exceptions to the lack of validated assessments (Tang et al., 2020). The tests include validity evidence through Classical Test Theory (e.g., factor analyses) and/or Item Response Theory (e.g., Rasch analyses) procedures. The PESS framework was not yet validated but is promising with regards to the generalizability problem; it was designed to suit the whole range of Swedish primary school children (ages 7– 15), refrains from using specific coding languages and/or software (items based on the Computational Thinking test), and combines both self-efficacy and skill-based questions.

Finally, the abovementioned tests form exceptions to the generalizability issues because they rely on block-based and/or textual representations of code for which the participants do not need prior experience. This has the benefit that measurements can easily be compared to determine the convergent validity of different measurements (Román-González et al., 2019).

The abovementioned tests have many positive characteristics that will benefit researchers and teachers. However, they also have shortcomings that we seek to address within the Computerized Adaptive Programming Concepts Test (CAPCT). We expanded the scope of the measurement both in size (4000+ items vs. \pm 30 items) and content (different CS/CT concepts). The large item set provides researchers the opportunity to create independent pre- and post-tests of equal difficulty (see the discussion in [section 4.3](#)). We collected response data across the range of Dutch primary school children (ages 4 – 13), showing that very young children were able to correctly answer questions about

common programming concepts. Finally, Math Garden allows for adaptive and longitudinal measurement of understanding of programming concepts.

When seeking to extend the content of the Computerized Adaptive Programming Concepts Test (CAPCT) beyond the concepts included in the already available assessments, we encountered a broad range of concepts and abilities associated with CT and CS. To illustrate, the International Society for Technology in Education (ISTE) & Computer Science Teachers Association (CSTA) (2011) described CT as – among other skills – the ability to persist when working with difficult problems. Measuring such an ability is not only extremely complicated, the ISTE & CSTA were also, to the best of our knowledge, the only ones to explicitly associate this ability with CT. In contrast, dealing with algorithms was included as a critical ability in most sources (e.g., Atmatzidou & Demetriadis, 2016, p. 664; “Algorithm is a practice of writing step-by-step specific and explicit instructions for carrying out a process. [...] selection of appropriate algorithmic techniques is a crucial part of CT.”). An extensive review of the differences in CT definitions can be found in Tang et al. (2020).

Selecting which CS/CT concepts to assess was further complicated by the realization that some concepts were found challenging by undergraduate students, let alone primary school children (e.g., recursion; Wu et al., 1998). We, therefore, conferred with experts in the fields of test development, software development, and cognitive development to compile a list of 19 programming concepts appropriate for assessment in children (see Appendix A for the operationalization of all concepts), out of which 7 were included in the CAPCT due to the applied design constraints (section 2.1.2).

1.2. Math garden: web-based adaptive data collection

Assessment of CT and CS ability typically occurs by collecting data in face-to-face settings; researchers go to schools or have participants visit their lab. Consequently, constraints are placed on the number of participants included in studies, the scope of the assessment (large item sets cost more valuable resources), and the representativeness of the sample. The current study bypasses these issues by collecting data through a web-based platform; Math Garden (Klinkenberg et al., 2011). Math Garden is an online practice and monitoring environment that allows children to automate arithmetic and general abilities (e.g., multiplication, pattern recognition). Over 2000 Dutch primary schools hold licenses for their pupils and at times even have mandatory practice sessions scheduled in their weekly routines.

Math Garden operates on an Elo-based adaptive algorithm (for an introduction see, Klinkenberg et al., 2011; for the statistical foundation see, Maris & van der Maas, 2012). Elo rating-systems are common in chess competitions, where each player has a unique rating, with higher ratings indicating higher levels of ability. When players compete against each other, based on their respective ratings, each player has an expected probability of winning the game. After the match, the difference in expected and observed outcome is used to update the ability estimate (i.e., Elo rating) of each player. The measurement scale produced by the Elo-rating system is a Rasch-scale, the basic model of item response theory (van der Linden & Hambleton, 1997). This means that Elo based scales are akin to the best measurement scales available in psychometrics. In Math Garden the adaptive algorithm is constructed in such a way that players “compete” against items (i.e.,

questions). If a player answers a question correctly, their ability rating will increase, while at the same time the difficulty rating of the item will decrease (and vice versa).

Computer Adaptive Testing (CAT) systems often aim for a success rate of 50% to maximize the evidence collected about a persons' true ability level (van der Maas & Nyamsuren, 2017; Wainer et al., 2000). Instead, the Math Garden algorithm strives for success rates of 60%, 75%, or 90% – depending on the players' chosen difficulty setting – to ensure a positive learning experience through maximizing success (cf., Jansen et al., 2016). To compensate for the loss of information from these higher percentages correct – the system is less aware of what a person doesn't know – the Math Garden algorithm applies an explicit scoring rule for the trade-off between response time and accuracy (correct/incorrect). All items in Math Garden have a time limit, for some games this is 8 seconds for other 15 or 60 seconds. The score achieved on an item is equal to the remaining time for an item if the response is correct, and minus the remaining time for incorrect responses. Player ability, or "winning/losing" from an item is thus determined by both the speed and the accuracy of the response. Over time this automated scoring rule ensures that ratings develop to values that indicate the true ability ratings of players and the difficulty ratings of items, holding information on both accuracy and speed (Coomans et al., 2016; Maris & van der Maas, 2012).

The adaptive algorithm as described above, is a self-organizing system with multiple benefits that surpass classic test development procedures. First, the system does not require pilot testing to establish the difficulty of the item-set, or to gather reference group data. The item difficulties and reference group data are established on the fly, with each response leading to a more accurate estimate of the item's true difficulty and the reference group ability parameters (Klinkenberg et al., 2011). Second, the system creates measurement scales that meet the requirements of modern test theory (e.g., Maris & van der Maas, 2012). Therefore, estimations of test validity and reliability are not constrained by keeping the item set and pool of participants constant. Finally, as is demonstrated in section 3.1.1, the adaptive algorithm was robust against system changes such as adding a large set of new items and continuously adding new users. Note, however, that using an adaptive system in itself does not resolve issues that are common in classical and modern test theory. For example, floor and ceiling effects may still arise when the content of the test does not match the abilities of the children. This may happen when children do not understand task demands due to a lack of instruction being provided, or when children demonstrate higher levels of ability than anticipated.

Adding the CAPCT as part of the Math Garden platform has benefits that go beyond the psychometric benefits discussed above. First, the adaptive algorithm and intelligent item selection allows children to play at their own level of ability. This means that depending on their responses, children may progress and regress through the items at their own pace, even encountering the same item multiple times before being able to answer it correctly (Hofman et al., 2018). Given the adaptive nature of the system, the large set of children that use Math Garden, and the popularity of the game, we were able to internally validate a more extensive and diverse set of items than each child would have been expected to complete in a classic setting. In Phase 1 we analyzed 520 items, in Phase 2 we increased the set to 4486 items (see section 3.1). Finally, as the game was made available to all Math Garden users, no active form of participant recruitment was

required, thereby allowing the participant pool to grow up until the moment of data-analysis.

1.3. Current study

The primary purpose of the current study was to address the lack of validated and widely applicable assessments of comprehension of programming concepts in primary school children (ages 4 – 13). We have created a set of 4486 items that aimed to measure comprehension of sequences, loops, if-statements, if-else statements, procedures, multiple-agents, debugging, and generalization to a different programming syntax. Following a two-step procedure, the items ($N_1 = 520$, $N_2 = 3966$) were included as a game in the adaptive online environment Math Garden, allowing for large-scale data collection across Dutch primary schools. The real-time estimations of player ability and item difficulty were used to validate the Computerized Adaptive Programming Concepts Test (CAPCT) following Embretson's Cognitive Design System Approach (Embretson, 1998). In short, this entails using the cognitive constructs underlying each item as predictors of item difficulty rather than applying conventional analyses based on external validation tests.

2. Method

In this method section we will first discuss how the Computerized Adaptive Programming Concepts Test (CAPCT) was created (section 2.1). Then we will discuss how the data was collected and which exclusion criteria were applied (section 2.2 & 2.3). Finally, the method section will conclude by describing how the data was analyzed (section 2.4).

2.1. Assessment design

2.1.1. Graphical design

A number of considerations were taken into account while developing the CAPCT. The target population of primary school children (ages 4 – 13) differs considerably in the ability to comprehend written instructions. Written instructions are often used for task descriptions or block based representations (e.g., the Bebras competition; Dagienė & Sentance, 2016). As this would negatively affect the performance of those with poor reading abilities, we (a) created a visual syntax style based primarily on symbols, and (b) kept task instructions as short and consistent across items as possible. During informal user testing, we learned that layout changes between items negatively affected children to such an extent that they were no longer evaluating a CS concept, but rather differences in user interface characteristics. We thus (c) created one format to which all items adhered. Each item consisted of a question, an image depicting the current situation (a grid, an algorithm, or a combination of grid and algorithm), and four answer-options (for an example see Figure 1). Where possible, (d) the incorrect answer-options (i.e., distractors) were constructed to evaluate common misconceptions. For example, in an if-else statement, the set of instructions that is executed depends on the evaluation of the conditional statement, executing some instructions but not others. In this case, a misconception would be to execute all instructions

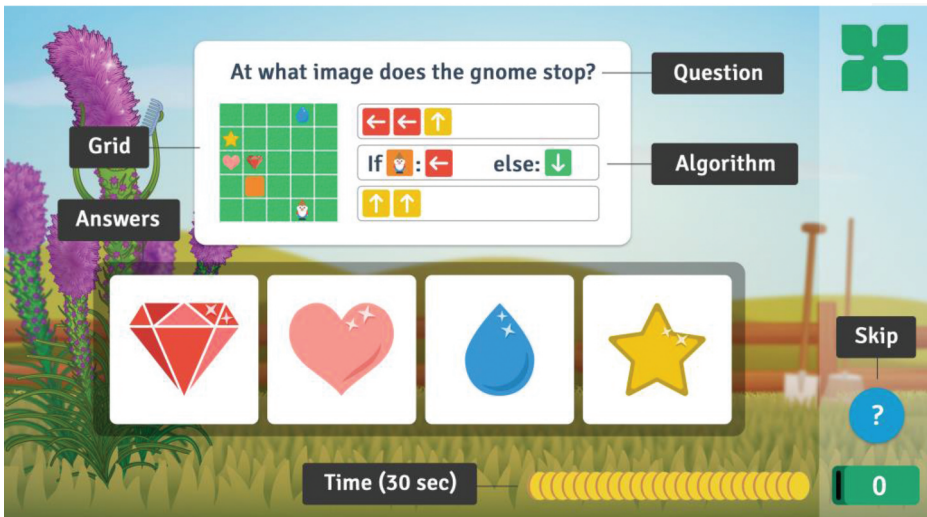


Figure 1. An example of an *if-else-if-met* item (see Table 1 for other item types). In this item, children execute the statements on the first row (left-left-up), and then decide whether the gnome is standing on the orange square or not. If the statement is true the gnome will go left, if not the gnome will go down. The statements on the last row are always executed (up-up). Children indicate the correct answer (the *star*) by pressing one of the answers (random order for each response). In this case, the *heart* evaluated the misconception that all instructions were executed, where the *diamond* evaluated the misconception that the “else” component of the algorithm was executed. The remaining *time* is represented by coins, with one coin representing one second. Children were allowed to *skip* one question each session by choosing the “?” button. Examples of all items are available online via <https://osf.io/bmf47/>.

irrespective of the conditional statement. Finally, we sought to make the game appealing for all children and thus (e) chose a gender-neutral gnome as the primary agent.

2.1.2. Content design

The operationalization of all concepts are listed in Appendix A. From this list we excluded 12 concepts they could not be assessed within the specified item format (e.g., data structures). The CAPCT therefore includes 7 representative programming concepts, which were assessed by 13 unique item types (see Table 1). In total 4486 items were generated through custom created R scripts. The design guide provides detailed information on the premises of each item type, how the items were created, and how answer-options were selected (available via <https://osf.io/bmf47/>). In short, following Embretson’s Cognitive Design System Approach (Embretson, 1998) each item measured a distinct cognitive process (e.g., evaluating if-statements), with prior expectations (see Table 1) of which cognitive processes were easier/harder. The answer-options were created as random mistakes, or as evaluations of known misconceptions (e.g., executing all visible code, rather than evaluating a branch of a conditional statement). For each item type between 44 and 600 unique items were created.

Table 1. An overview of the programming concepts included in the Computerized Adaptive Programming Concepts Test (CAPCT). Each concept includes one or more *item types*, which measure distinct cognitive constructs. The *number (N) of instructions* served as a proxy for cognitive load. The combination of cognitive construct (item type) and cognitive load (N instructions) is classed as the *item level*. We established prior rankings after consultation with multiple experts in the field of software development and education. Posterior ranks were calculated from median difficulty ratings, the development of which is portrayed in Figure 3. Visual examples of all items, as well as detailed descriptions of the way answer-options were selected, are available via <https://osf.io/bmf47/>.

Programming Concept	Item Type	Description	N Instructions	Prior Ranking	Posterior Ranking
Sequences	Sequences	Given a grid with a gnome and rainbow depicted; which set of up, down, right, and left instructions gets the gnome to the rainbow?	1 – 8 instr.	1	2
Debugging	Debugging	Identification	Find which one of four presented answers is incorrect in the sense that it does <i>not</i> cause the gnome to reach the rainbow.	4 instr.	2
Debugging	Modification	The child is presented with an incorrect set of instructions which can be modified by changing one pre-selected instruction to a different orientation.	5 instr.	3	6
Loops	For Loops: Known	Matching of for-loop algorithms (i.e., repeat the enclosed instruction x number of times) to one of four grids.	2 – 4 instr.	4	1
	For Loops: Unknown	How many times should a set of given instructions be repeated for the gnome to reach the rainbow?	2 – 4 instr.	5	4
Conditions (If Statement)	If At: Met	Where does the gnome end up? The if-statements equals true; the gnome is standing on the orange square at the time of evaluation – the instructions depicted are executed.	3 – 9 instr.	6	5
	If At: Unmet	Where does the gnome end up? The if-statement equals false; the gnome is <i>not</i> standing on the orange square during evaluation – the instructions depicted are <i>not</i> executed.	2 – 8 instr.	7	8
Procedures	Procedures	Where does the gnome end up? Included in the set of instructions is a procedure call. This means that the instructions included in the procedure (e.g., up, up) are “inserted” when the procedure was called.	6 – 8 instr.	8	13

(Continued)

Table 1. (Continued).

Programming Concept	Item Type	Description	N Instructions	Prior Ranking	Posterior Ranking
Conditions (If-Else Statement)	If Else: If Met	Where does the gnome end up? The If statement equals true; the gnome is standing on the orange square during evaluation – the instructions depicted after the <i>if</i> are executed, but the instructions after <i>else</i> are <i>not</i> .	4 – 12 instr.	9	12
	If Else: Else Met	Where does the gnome end up? The if statement equals false; the gnome is <i>not</i> standing on the orange square during evaluation – the instructions depicted after the <i>if</i> are <i>not</i> executed, but the instructions after <i>else</i> are.	4 – 12 instr.	10	11
Multiple Agents	Multiple Agents	Two agents are placed in a single grid, where can they meet each other in the least number of steps?	2 – 8 instr.	11	3
Generalization	Generalization-Regular	The ability of a child to utilize relative directions (i.e., take x steps forward, turn left/right) to solve the same grids as were included in the simple algorithm items.	3 – 8 instr.	12	7
	Generalization-Reversed	Given a solution algorithm of relative directions – which grid shows the corresponding problem?	3 – 7 instr.	13	10

2.2 Implementation and data collection²

The items were implemented in Math Garden as a game called “Codetaal” with translates as “Codelanguage”. As a minimum level of reading ability was required to understand task demands, the game was made available to children who were able to add and subtract at a grade three level (± 6 years old, US grade 1). On the premises that Dutch children will have learned to read at a fundamental level during this phase of their primary school education. We should note that the adaptive nature of Math Garden allows children to progress to ability levels well beyond their age or grade– thus allowing younger children who have reached the specified entry requirements to also play Codetaal. Children were free to start and stop playing Codetaal at their own discretion. It is Math Garden policy to return players to the home screen after completing ten items: discouraging children from over-practicing or neglecting singular abilities. After returning to the home screen, children were free to reselect the game and continue playing.

The planned analyses rely on stable data (discussed in more detail in [section 3.1](#)), requiring a large number of responses per item. As the popularity of games within Math Garden vary from time to time, we opted to release the game with a subset of items (N = 520; Phase 1), ensuring sufficient data per item to validate the CAPCT. Phase 1 was conducted over a period of 101 days until the 2nd of August (2018) and comprised of school days (N = 55), weekends (N = 20 days), and national- and school-holidays (N = 26). Due to the high volume of responses in Phase 1 (N = 3,744,355), with item ratings stabilizing after two days ([Figure 2](#)), we felt confident that we could collect the responses required to validate the remaining items that had been created (N = 3966; Phase 2). The complete item set (N = 4486) will continue to be available to children for the time-being, however, the response data was analyzed up until the 1st of June (2019), comprising of school days (N = 160), weekends (N = 55 days), and national- and school-holidays (N = 88).

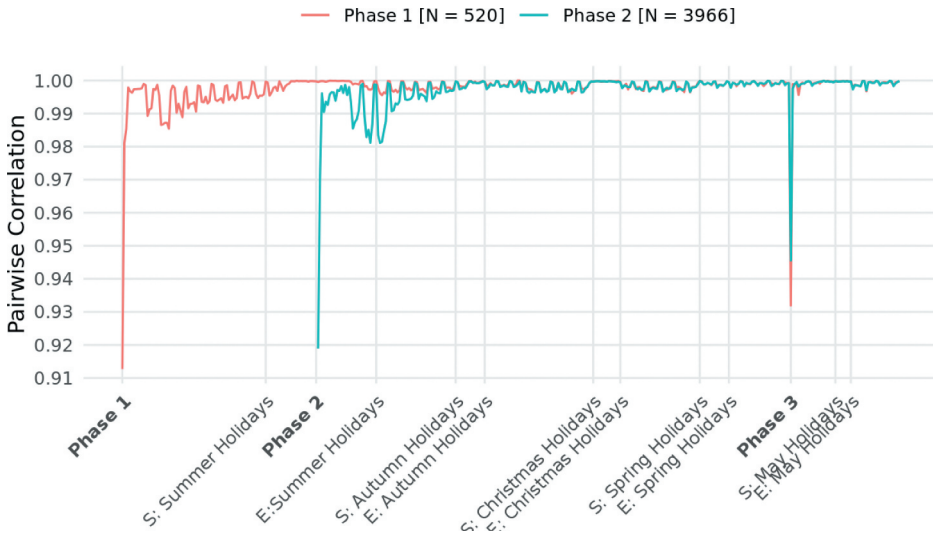


Figure 2. Pairwise correlations between item difficulties on consecutive days (Lag 1). The magnitude of the pairwise correlations was plotted over days, with significant dates marked on the x-axis (e.g., “S: Summer Holidays” = Start of the Summer Holidays). The red line represents the set of 520 items that were launched for Phase 1, the blue line represents the remaining 3966 items of the total item set (N = 4486) that were launched for Phase 2. Higher pairwise correlations indicate higher levels of stability over days.



Figure 3. The relative difficulty of cognitive constructs (i.e., item types; Table 1) over weeks. The ranks were determined based on the median item difficulty per item type. In Phase 1, the total item set consisted of 520 items with 20 to 60 items per item type. In Phase 2 we added 3966 items, making a total item set of 4486 items, with 44 to 600 items per item type. During Phase 3 (discussed in more detail in Section 3.4), we manually reset 52 item difficulties.

2.3. Exclusion criteria

Math Garden relies on demographic data, especially grade information, to ensure that appropriate results are presented to the players. Schools who provide their pupils with accounts often use the automated connection between Math Garden and the students' administrative system. This ensures that grade information remains reliable even when children move classes, or the next school year starts. Some players (1.7% of the included players in this paper) have so-called family accounts; privately owned accounts administered by their parents. Even with regular reminders, the family accounts tend to show more missing and/or inconsistent data (e.g., failing to update the players' grade at the start of a new school year). These anonymous player data showed that some players warranted exclusion from further analyses that relied on *demographical data*.

First, the personal data of 10,039 players were missing. Second, 1,816 players fell outside the age range of Dutch primary schools (below 4 or above 13; e.g., teachers) and 885 were registered as secondary school students or above (grade above 8; US grade 6). Consequently, the data of 12,740 players were excluded from analyses that rely on *demographic* information (e.g., predictors of player ability). Note that the responses of these players (8.4% of the total amount of responses) were *not* removed from analyses that relied on *response* data (e.g., differences in item difficulty), as response data are updated on-the-fly (Klinkenberg et al., 2011), and removal of these responses would create data irregularities that, for example, prevent accurate analysis of rating stabilization (section 3.1.1).

2.4 Data analyses

The primary purpose of the data analyses was to establish the internal validity and reliability of the CAPCT. However, before being able to conduct these analyses we first established stability of the data. As the adaptive nature of Math Garden allows the outcome variables (e.g., item difficulty) to change continuously, we needed to ensure that the variance between days was no longer of a conclusion changing magnitude. We explored the stability of item difficulties in two ways. First, we computed pairwise correlations with Lag 1; a procedure common in time series analysis (e.g., Shumway & Stoffer, 2017, sec. 3.1.1). Second, we showed that the relative difficulty of underlying cognitive concepts was stable over time in a rank-based analyses (section 3.1.2).

After we established that the data was stable, we explored the construct validity of the CAPCT. Embretson discussed that construct validity for cognitive assessments can be established by "... the overall fit of a mathematical model. The dependent variable of the mathematical model is item performance (i.e., accuracy or response time) and the independent variables are the item stimulus features." (Embretson, 1998, p. 383). In the case of the CAPCT the performance of an item was captured in item difficulties as they were an aggregation of both accuracy and response times (Klinkenberg et al., 2011), and the independent variables were CS concept, Item Type (i.e., cognitive construct), and/or the Number of Instructions. The number of instructions that required evaluation was included as a proxy for cognitive load. For example, a sequence (easy concept) with eight arrows may put a larger strain on the cognitive load than an if-else statement (hard concept) with four arrows. Following Embretson's logic we created four linear regression

models with above mentioned parameters (section 3.2). We have computed these models on a weekly basis to show that even though the performance measure (item difficulty) fluctuates slightly over time, our ability to explain the variance does not.

From these analyses it became evident that we were unable to explain a small portion of the variance in item difficulty. We explored this by conducting visual analyses of item clusters and response patterns to the items (section 3.3). These analyses then led us to conduct an exploratory analysis; the manual reset and follow-up analyses of 52 item difficulties (section 3.4).

Our final set of analyses explored the variance in player ability, or in other words how well children were able to apply CS and CT concepts in a multiple-choice format. We computed seven linear regressions, again on a weekly basis, with the predictors age, gender, the number of items completed, and prior mathematical ability (section 3.5). We conclude with an analyses of potential gender bias in over- and underperforming on clusters of items.

3. Results

Codetaal was played by 93,341 Dutch children who were between 4 and 13 years old ($M = 10.2$, $SD = 1.6$, $Median = 10$) in grades one through eight ($M = 5.9$, $SD = 1.5$, $Median = 6$). The players were registered at 3401 different schools, with a minimum of 1 and maximum of 383 children listed per school ($M = 27$, $SD = 46$, $Median = 4$). The analyzed sample (a) included only three of the youngest children (4 years old) generally enrolled in Dutch primary schools, (b) spanned across all grades (1 – 8), and (c) included children from approximately 50% of all Dutch primary schools (3401 out of 6739; Centraal Bureau der Statistiek, 2018). We, therefore, believe that the degree of representation achieved through our data collection was an accurate representation of all Dutch primary schools.

Relatively speaking a limited number of children from the lower grades played the game. This was most likely due to the set entry requirement which made the game available only to children who were able to perform additions and subtractions at a grade three level (US grade 1; see section 2.2). Therefore, when viewing the results, one should take into account that the children from grades one and two are likely performing at a level well beyond their peers.

The children provided 14,154,189 responses. Each unique item ($N_1 = 520$, $N_2 = 3966$) was responded to between 506 and 109,516 times ($M = 3155$, $SD = 6296$, $Median = 1848$). We observed a ceiling effect for the best players in our sample. For 7.2% of the children, the content was too simple as indicated by them performing above the difficulty level of 99% of the items. This ceiling effect has had minor effects on the results, which we will discuss in section 3.2.

The code and outcome of all analyses below are available via the Open Science Framework (<https://osf.io/bmf47/>). These files include colored and zoomable figures; descriptives; additional visualizations, general descriptions of data structures, and examples of all item types.

3.1. Stabilization of data

3.1.1 Item difficulties

In order to establish the stability over time of the primary outcome variable 'item difficulty' we conducted a lagged pairwise correlation analysis (Shumway & Stoffer, 2017). For each day an item was played, a new item difficulty was saved. Consequently, on days where few responses were logged (e.g., holidays), a range of items may not have had updated item difficulties saved. These missing data were excluded in a pairwise manner, with the correlations between item difficulties on consecutive days (Lag 1) plotted in Figure 2.

The pairwise correlations show multiple interesting patterns. First, as is common in Math Garden data, the repetitive fluctuations in correlations are due to a weekend effect, with weekends showing higher pairwise correlations than schooldays. This effect, which is similar to the increase in correlations seen during holidays, is due to a relatively small number of children playing a small number of items. Although the repetitive nature of this effect is apparent, it is also evident that the difference between schooldays and holidays/weekends becomes smaller over time declining to less than 0.1 difference.

Second, Figure 2 shows that after a single day of collecting data (122,793 responses) the item difficulties have already converged to stable ratings, as indicated by the high pairwise correlation (.98) with the item difficulties logged on the second day (137,221 responses). In addition, the system proved to "bounce back" from changes to the system. The first change we made was expand the content of the CAPCT from 520 items (Phase 1) to 4486 items (Phase 2). This caused the pairwise correlation to drop, however the self-organizing nature of the system returned to stable ratings after the summer holidays were over and sufficient numbers of responses were collected. The second change to the system was to manually reset a subset ($N = 52$) item difficulties in Phase 3 (see section 3.4 for further explanation). Again, the pairwise correlation dropped, but quickly returned to highly stable ratings.

3.1.2 The relative difficulty of concepts

For policy makers and educators, it is especially relevant to know which concepts are easier or harder, as it may provide useful guidelines towards creating curricula guidelines for teaching Computer Science (CS) concepts to primary school children. However, at times a CS concept can require different levels of understanding and cognitive processes. For example, an If-statement that evaluates to true, requires the same cognitive procedures as a sequence without the conditional statement (executing all depicted instructions). In contrast, and If-statement that evaluates to false, requires the understanding that some instructions are depicted, yet not executed by the agent. We therefore conducted the analyses based on the relative difficulty of the cognitive constructs in the different item types (see Table 1). We computed ranks based on the median item difficulty of each item type. As the sample size between, and the variance within item types varies greatly (e.g., Procedures; $N = 44$, $SD = 2.03$, Debugging-Identification; $N = 100$, $SD = 0.25$) we have refrained from conducting significance analysis (Brunner et al., 2018). Rather, we have plotted the ranks for each day that data was collected ($N = 404$) to show that the relative difficulty of item types is stable, though interchangeable in some cases (e.g., For-Loop-Known, Sequences, Multiple-Agents; Figure 3). The interchangeability of

ranks between item types is discussed in more detail in [section 3.3](#) where we show how items cluster together within and across item types.

3.2. Internal validity of the CAPCT

In [section 3.1](#) we showed that the data was stable over time – thus allowing for internal validity analysis of the CAPCT. Following Embretson’s logic for validating cognitive assessments, we fitted four different linear regressions. All models seek to explain the variance in item difficulty, as this aggregates the performance parameters „accuracy„ and “response time” (Embretson, 1998; Klinkenberg et al., 2011).

As the CAPCT measures understanding of CS concepts, the first model explores the predictive value of Computer Science (CS) concepts as a whole (see [Table 1](#)). However, as discussed above, we consider the CS concepts too broad a clustering. The second model therefore explored the predictive value of the different cognitive constructs (i.e., item type). In the third model, we have included the number of instructions depicted in the algorithm as a proxy for cognitive load. All arrows, conditional-, and loop-statements count as an instruction. This third model thereby accounts for the possibility that an easy construct with eight arrows may be more difficult than a difficult construct with 2 arrows. The fourth model additionally includes the number of item responses (NIR) of each item. We noticed that at times items are more difficult than others for no apparent reason other than that they have been responded to a significant amount more (see [section 3.4](#) for further explanation).

The results of the linear regression analyses of the last day are listed in [Table 2](#). In addition, we have plotted the explained variance of each of the models over time in [Figure 4](#).

The Figure and Table show that the fourth model – taking into account cognitive construct (i.e., item type), cognitive load (Number of Instructions), and the Number of Item responses – always outperforms the other models. Interestingly, the difference between Model 3 and 4 disappears after a manual reset of item difficulties in Phase 3. Note that the lower explained variance in phase 2 and 3, compared to phase 1, is due to the increase in the number of items and thus to be expected.

3.3 Cluster analysis

In the previous section we saw that the variance in item difficulty was explained for $\pm 75\%$ by the cognitive construct (item type), Number of Instructions, and the Number of Item Responses (NIR). However, in [section 3.1](#) we also saw that the relative difficulty of item

Table 2. An overview of the four linear regressions at the time of data-analysis. The model with the lowest AIC and BIC is preferred (model 4; bold). For an overview of the different CS concepts, item types, and number of instructions please see [Table 1](#).

Model	df	R ²	Adj. R ²	AIC	BIC
CS Concepts	9	0.60	0.59	20,395	20,453
Item Type (IT)	14	0.64	0.63	19,940	20,030
IT + N Instructions (NI)	15	0.74	0.74	18,466	18,562
IT + NI + Number of Item Responses (NIR)	16	0.75	0.75	18,295	18,398

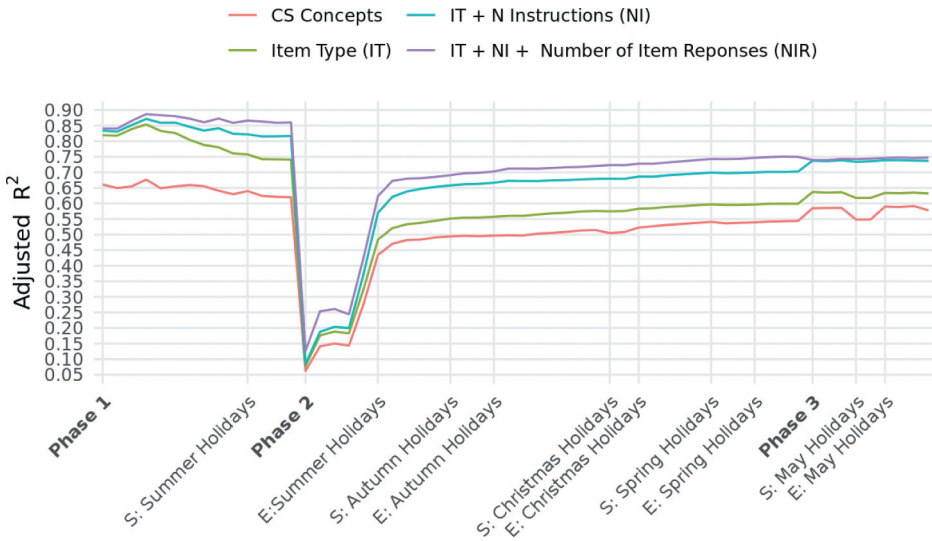


Figure 4. The explained variance (Adjusted R^2) for each of the four linear regressions on a weekly basis. Significant dates were marked on the x-axis (e.g., “S: Summer Holidays” = Start of the Summer Holidays).

types were at times interchangeable. We, therefore, explored the variance of item difficulty within and between items that measure the same cognitive construct (i.e., item type) and have the same number of instructions depicted (our proxy for cognitive load; Figure 5).

The item difficulty scale, which is on the same scale as player ability, is centered at 0 but unconstrained in its' limits. The scale is directly influenced by the number and ability/difficulty of players and items. Consequently, an absolute score of „5,, will provide little information. Which is, in fact, similar again to Chess ratings; a score of 2600 offers little information about ability, unless you know the mean and standard deviation of chess ratings. Similar, an item with score of 5 is interpreted as difficult only when you know that 1.7% of the 4000+ items were more difficult than that. We thus discuss item difficulty and player ability from a relative perspective rather than absolute.

Figure 5 provides valuable insight into what makes questions easy or difficult. When comparing these results to those in Figure 3 (the stability of item type ranks over time), we should note that the clusters are not directly comparable. In Figure 3, all item types were grouped together, regardless of the number of instructions in each question. The current figure does show however that the medians of clusters lie very close together, and thus are likely to be interchangeable given the minor variations of the adaptive algorithm. From the presented visualization additional effects become noteworthy.

First, some clusters exhibit bimodality (e.g., Sequence_2– the third row from the bottom). We explored the difference between these modes by visualizing each item, including the response patterns made by the children as a percentage of the given answers. For the Sequence items it was quickly evident what made a question easy or hard; did the gnome make a turn? To illustrate; the easy items were the items where

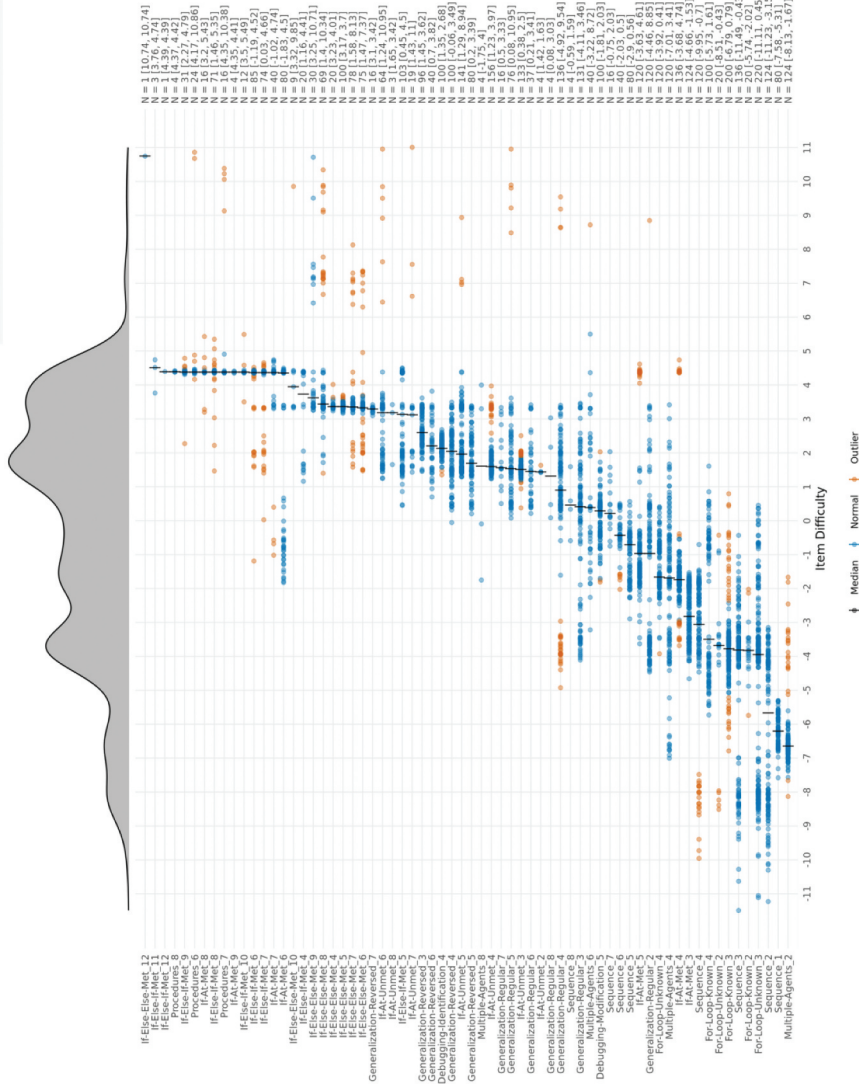


Figure 5. The distribution of items grouped across cognitive constructs (i.e., item type; see Table 1) and the number of instructions that need to be processed (e.g., Multiple-Agents_2 = item_type_2 instructions). Each item included in the CAPCT (N = 4486) is depicted in blue (normal item) or red (outlier). Outliers were indicated as such when the item difficulty differed more than 1.5 times the IQR from Q1 or Q3 of the cluster. The median is indicated by the black vertical line. The top-margin shows the distribution of items across the item difficulty range, the right-margin lists the number of items and the minimum and maximum rating of each cluster. A zoomable plot is available via <https://osf.io/bmf47>.

the gnome travelled in a single direction (e.g., “up-up”), while the harder items were those where the gnome travelled in two directions (e.g., “up-right”). However, there are other clusters, such as the “Generalization-Regular”, where not only did we not find an explanation for the bimodality, nor were we able to explain the patterns of (singular) outliers. The inability to explain why (singular) outliers and bimodalities occurred led us to initiate Phase 3; manually resetting item difficulties for outlying items.

3.4. Exploratory analysis: phase 3

The two primary reasons for conducting Phase 3 were the significant predictive value of Number of Item Responses on item difficulty (section 3.2), and our inability to explain patterns of (singular) outliers (section 3.3). The aim of Phase 3 was therefore to explore whether these items were outliers due to their content or whether other parameters were influencing item difficulties. We reset 52 items to the median item difficulty of the respective CS Concept clusters (3 x Multiple-Agents, 4 x If-At-Met, 5 x Debugging-Identification, 5 x If-Else-If-Met, and 35 x Generalization-Reversed). If items were true outliers, after a manual reset, they would migrate again to their phase 2 difficulty (i.e., an outlier position). This would happen, for example, if an item was tagged with an incorrect answer, or had duplicate answer options. We determined extreme outliers based on a visual analysis of Figure 6, where the orange X’s represented the old situation.

From the data it becomes apparent that the manual reset was effective in the sense that the outlying items remained stable among their peers. Consequently, after Phase 3, the unexplained bimodality of Generalization-Reversed no longer existed. However, an undesirable consequence of Phase 3 was that other items started to drift away from their clusters. This effect was most visible among the more complicated items, such as the Procedures.

The primary reason for this drift is the ceiling effect with 7.2% of the players outperforming 99% of the items. Such a mismatch between subjects and test content in IRT and computerized testing is always problematic. The manual reset of Phase 3 demonstrated that the items were outliers due to other effects than design mistakes (e.g., all answers being incorrect). However, resetting items unfortunately is only a temporary solution. Permanent solutions include the exclusion of outperforming players from the game (a reasonable approach seeing that they “passed” the test) or adding more difficult items. In a future version of the CAPCT we will opt for the latter solution by converting the CAPCT to an (interactive) open-answer format.

3.5. Reliability: what predicts player ability?

The main purpose of the CAPCT was to measure the extent to which children understand common programming concepts. Our final analyses were therefore concerned with the extent to which we can predict the ability of children from the available player characteristics.

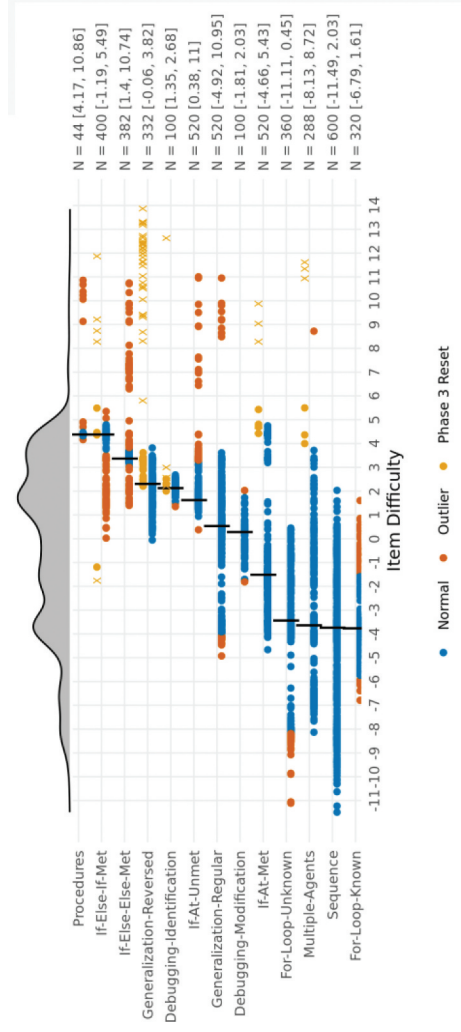


Figure 6. The distribution of items per cognitive constructs (i.e., item type; see Table 1). The items that were manually reset during Phase 3 are depicted in orange, with the X marking the situation before the reset, and the circles the current situation. Items that were not moved are marked as normal (blue) or outliers (red) based on a 1.5 IQR distance from Q1 or Q3. The median is indicated by the black vertical line. The top-margin shows the distribution of items across the item difficulty range, the right-margin lists the number of items and the minimum and maximum rating of each cluster.

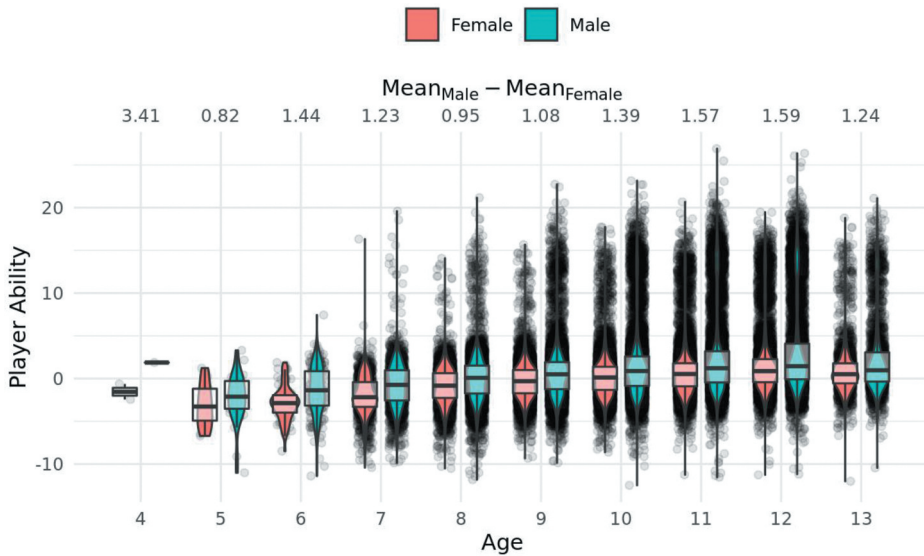


Figure 7. An overview of the achieved player abilities as differentiated by age and gender. Each player is represented by a data-point, with group statistics (median, and 1st and 3rd quartiles) displayed by the violin- and boxplots. The top x-axis describes the difference in mean abilities of gender per age-group. Note; there are only three players of age 4.

3.5.1. Age x gender

We first explored the effects of age (5 – 13) and gender (male/female) on player ability (continuous) with a Wilcox non-parametric independent two-way ANOVA with 20% trimmed means (Mair & Wilcox, 2019). The distributions of players across these groups are displayed in Figure 7. We excluded the four-year old users from the analysis because the number of users in each group (two females, one male), did not allow us to compare means. The analysis showed evidence for main and interaction effects ($F(1, 93,340) = 51, p < .001$). On average, males ($M = 1.7, SD = 4.5$) outperformed females ($M = 0.46, SD = 3.1$; $F(1, 93,340) = 90, p < .001$), and children in the higher grades generally attained higher levels of ability ($F(7, 93,340) = 6028, p < .001$). There is a small interaction effect indicating an increasing gender difference with age.

3.5.2. Prior ability

The ANOVA showed that gender and age predicted player ability, but prior levels of ability may also play a role. Ideally, prior ability would be measured by pre-testing CT or CS skills (e.g., the Computational Thinking test; Román-González, 2015), or via self-report of programming experience. Such estimates of prior ability were not possible within the Math Garden environment – but we do have access to players' scores on other games. As a proxy for prior ability, we chose the games Addition and Subtraction, which also serve as the entry criteria for new players (section 2.2). Prior ability was established as the last standardized score attained in the two months prior to first playing Codetaal. With a maximum of ± 25 games available to a child, data sparsity arises from allowing children to freely choose which games to play (see the drop in N's in Table 3).

Table 3. An overview of the linear regressions of player ability. The model with the lowest AIC and BIC is preferred (model 7; bold). Mean and SD adjusted R^2 were computed from the weekly analyses available via <https://osf.io/bmf47>.

Model	df	R^2	Adj R^2	AIC	BIC	N	Mean Adj. R^2	SD Adj. R^2
1: Age	2	0.10	0.10	518,310	518,328	93,341	0.18	0.09
2: Gender	3	0.10	0.10	518,006	518,034	93,341	0.18	0.07
3: Age x Gender	5	0.16	0.16	511,039	511,086	93,341	0.26	0.10
4: Number of Responses (NoR)	2	0.21	0.21	506,210	506,229	93,341	0.29	0.08
5: NoR + Age x Gender	6	0.49	0.49	463,970	464,026	93,341	0.58	0.05
6: Addition + Subtraction (Prior Ability)	3	0.17	0.17	349,955	349,982	63,802	0.28	0.11
7: NoR + Age x Gender + Prior Ability	8	0.57	0.57	308,815	308,888	63,802	0.64	0.05

Note: the Number of Players (N) included in Models 1 – 5 differs from Models 6 – 7. The data sparsity results from free playing choice – some players will not have played Addition and Subtraction in the two months prior to playing Codetaal.

In order to explore the effect of prior ability beyond gender and age effects, we computed seven linear regression models of player ability. We first explored the effects of age (continuous) and gender (male/female). We then added the Number of Responses (NoR) made by a player to the predictive model – which was also a predictor of item difficulty (section 3.2). The predictor NoR was log transformed to normalize the positively skewed distribution. Model 6 and 7 then take prior ability into account (Table 3).

The number of players per week varies greatly. For example, during the Christmas holiday ± 1.000 players were included in the regression models, which is in sharp contrast to other weeks ($M = 8.344$, $SD = 5,874$). Differences in the number of players per week were expected, as children were free to choose which games to play and may only have played during school hours. For this reason, when talking about the predictors of player ability, we will discuss average explained variance. The longitudinal analysis, which was similar to the item analyses in section 2.3, is available in the online Appendix (<https://osf.io/bmf47/>).

3.5.3. Gender bias in item responses

Math Garden collects a range of data, including whether the players' accuracy (correct/incorrect) and speed when answering an item were above or below expectation. This expectation is based on the current ability of the player and the difficulty of the item (for the formula for expected score see Maris & van der Maas, 2012). In general, the difference between scores and expected scores should be zero, but it is possible that a player or a group of players systematically over- or underperform on specific items or sets of items. The 14 million records collected for Codetaal thus allow us to explore whether different groups of players consistently perform above or below expectation on specific item clusters.

In the following analysis we explore under- and overperformance of gender (Male, Female) on clusters of items. We chose to conduct two different clusters: items that measure the same underlying constructions (item types; see Table 1) and items that require the same cognitive load (N Instructions). We computed the under-/overperformance for each record with known gender information ($N > 12,000,000$). Some of these players, most of whom are outperformers, have responded extremely often to a single item (e.g., > 400 times). By averaging their responses, we reduce the impact these players may have on the analysis. The gender bias was computed from differences in average

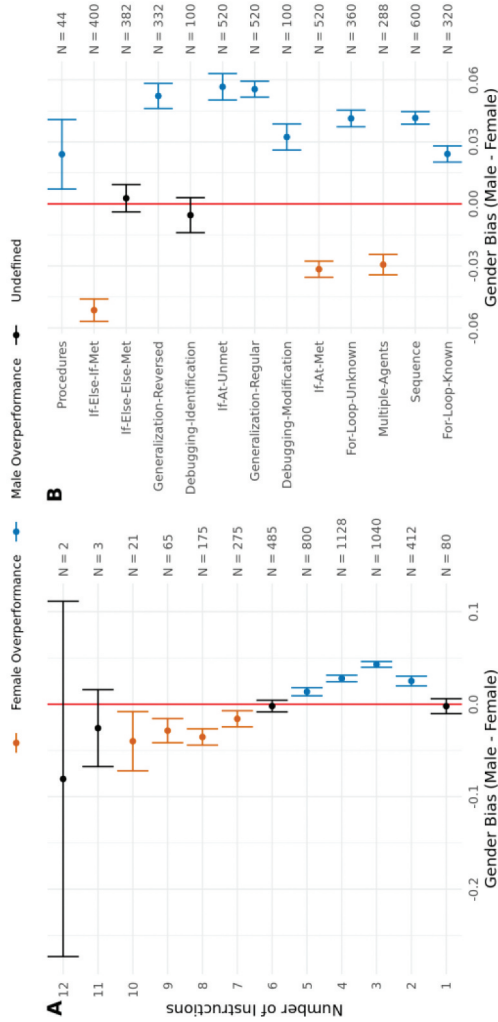


Figure 8. The mean gender bias in “score – expected score” for items in distinct clusters. (A) items were clustered by the number of instructions that required evaluation. This measure served as a proxy for cognitive load. (B) items were clustered by the cognitive construct (item type; Table 1) they measured. The item types were ordered by median item difficulty, showing the easier constructs at the bottom of the figure. Error bars represent the 95% Confidence Interval of the mean. Colors indicate whether, on average, male or female players tended to over-perform on this cluster of items. A bias is classed as “undefined” when the 95% CI includes 0. The number of items in each cluster is listed on the right-hand side of each plot. Note; plot A and B have differently scaled x-axes.

score per gender per item. [Figure 8](#) shows the mean gender bias for all the items in a cluster, with the error bars indicating the 95% Confidence Interval of the mean.

The scoring scale, upon which [Figure 8](#) is based, reaches from -1 to 1 ; the found gender biases were thus relatively small. The results, however, are based on large samples sizes (N 's per cluster) with aggregated scores from more than 12 million responses. The magnitude of the data therefore allows for some preliminary evidence of subtle gender bias effects that appear to be related to the difficulty of items/clusters.

[Table 2](#) ([section 3.2](#)) shows that the combination of Number of Instruction ([Figure 8A](#)) and Item Type ([Figure 8B](#)) explains 74% of the variance in item difficulty. This makes the subtle gender effects even more interesting. The Number of Instructions shows a clear pattern where easier items (low cognitive load) see more over-performing male players than female players ([Figure 8A](#)). At the same time, items with 7 – 10 instructions (high cognitive load) show more over-performing female players. [Figure 8B](#), however, shows that the relative difficulty of the measured CS concept shows a less pronounced and more random relationship with gender bias. It appears as if most concepts see over-performing male players, while three distinct concepts show predominantly over-performing female players. A possible explanation is provided in the discussion.

4. General discussion

In the current study, we sought to create and validate a new assessment of programming concepts comprehension suited for (Dutch) primary school children (ages 4 – 13); the Computerized Adaptive Programming Concepts Test (CAPCT). Overall, the CAPCT (4486 items) showed good psychometric qualities. Since across the item set, 75% of the variance in item difficulty could be explained by differences in item characteristics, internal validity appears to be high (Embretson, 1998). On average, we were able to explain 64% of the variance in player ability, taking into account the age, gender, number of responses, and prior mathematical ability of players.

4.1. Items

Our data showed that the CAPCT was highly reliable with item difficulties remaining stable over time. The CAPCT may thus reliably be used by researchers and educators to estimate the understanding of programming concepts in (Dutch) primary school children. We should note, however, that the construct validity of certain Computer Science (CS) Concepts warrants further exploration. For example, the prior difficulty estimations of our experts, did not match the data-based posterior rank of the concept “Multiple-Agents”. When looking at the items post-hoc,³ it becomes evident that the intended concept Multiple-Agents (*“The ability to identify that two or more conditions must be true, so that task demands can be met. For example; a scale can only be balanced when two weights of equal weight are placed at either end of the scale”*; see [Appendix A](#)), may not have been measured by these items. The data, however, also showed that the items themselves were not the problem; they were highly stable and clustered well together ([Figure 6](#)). Therefore, rather than dismissing these items, we propose that these items were in fact a different way of measuring the understanding of basic Sequences.

From the differences in expert estimated prior-ranks, and data based posterior-ranks we also learned that adult experts may not be the best judges of what the primary school children understand of programming concepts. This calls for more data driven approaches such as the current study, and the need for more complicated content. We, for example, did not include concepts such as recursion because we believed them to be too complicated for primary school children to understand. Further research will have to demonstrate, if we were correct in our assumptions, or whether children are more capable than we expected.

In conclusion, even though some constructs warrant redefinition, the items themselves are highly reliable and retain a stable difficulty over time.

4.2 Users

Our data showed that, on average, we could explain 64% of the variance in user ability based on age, gender, the number of responses, and prior mathematical ability of players. With an average of 28% of the variance explained by differences in the number of responses made. The fact that more responses lead to higher levels of ability may provide evidence for a (self) learning trajectory. Such evidence is of value to the educators and policymakers seeking to implement programming as part of (formal) education, as it provides concrete guidelines about how one could build up a programming-oriented lesson plan. However, external validity studies are required to verify whether the found trajectory was merely a byproduct of the assessment design, or whether the found trajectory replicates when utilizing different assessment measures (e.g., the Computational Thinking test; Román-González, 2015).

In creating the new assessment, we aimed to reduce the resources required to measure understanding of programming concepts. To do so, we opted for a multiple-choice format which allows easy administration and scoring. However, programming ability is often referred to in a problem-solving context (Chang, 2014), where one actively searches for a solution, without being provided with explicit guidelines as to what the solution should be. The adopted multiple-choice format provided players with answer possibilities, thereby considerably reducing the search-space available for finding the correct solution. Future research might thus explore whether open-ended questions (e.g., creating the algorithms by dragging arrows to an answer box) will result in similar item difficulties and CS concept ranking as we found in the current study. Adopting open-answer formats with increased search-spaces may also resolve the issue that approximately 7.5% of the users outperformed 99% of the items (a ceiling-effect), thus creating a measurement that is suited for a larger range of abilities.

In [section 3.5.3](#) we explored the gender bias of different item clusters; the Number of Instructions (proxy for cognitive load) and the item type (CS construct). [Figure 8A](#) showed a clear pattern where items with fewer steps to process (low cognitive load), on average, saw more over-performing male players. In contrast, items with more steps to process (high cognitive load) showed more over-performing female players. A possible explanation for these subtle gender effects is a gender difference in the applied speed-accuracy trade-off.

The algorithm behind Math Garden computes ability as a function of accuracy (correct/incorrect) and response time (Klinkenberg, 2014). A player thus receives the optimum score by responding both accurate and fast. However, to discourage guessing, the scoring algorithm also penalizes players for giving the wrong answer by retracting the number of coins earned (see Figure 1). The speed-accuracy trade-off necessary to navigate the Math Garden algorithm may cause differences in response patterns (fast/slow). In fact, prior research of multiplication items demonstrated that male players were more likely to give a fast answer than female players (Hofman et al., 2018). If items with fewer instructions require less cognitive processing, faster speed-accuracy trade-offs may be rewarded as “hasty” and “slow” thinking may lead to the same accuracy. However, items with more instructions to process may see higher accuracy in those that are “slow” rather than “hasty” players. Research will have to determine whether the speed-accuracy trade-off in Codetaal and/or Math Garden is indeed different across genders and may explain systematic but subtle under- and overperformance.

In Figure 8B we saw that Multiple-Agents, If-At-Met, and If-Else-If-Met clusters saw relatively more female over-performance than male over-performance. We are not sure why this may be the case. The difficulty of the Multiple-Agents and If-At-Met items falls neatly together with male over-performing constructs (e.g., Sequences; Figure 6). And in fact, if the same cognitive strategy is applied to Sequence and If-At-Met items; they would both result in the correct answer. Future research will have to uncover whether male and female players adopt the same strategies to solve the items. This could be done, for example, by using mouse tracking technologies to study underlying thought processes (e.g., de Mooij et al., 2020).

4.3 Future research

In this paper we have presented the first steps towards validating a new assessment of programming concepts understanding; the Computerized Adaptive Programming Concepts Test (CAPCT). The potential of the CAPCT is great, but future research has to determine when the CAPCT is/isn't suited to determine understanding of programming concepts.

We created the CAPCT item set with the intention of allowing researchers and teachers to create their own paper-and-pencil tests (materials available via <https://osf.io/bmf47/>). However, paper-and-pencil tests are often non-adaptive and include only a limited number of items (e.g., 20). Extra validation studies are required to determine the validity of a non-adaptive CAPCT, including the effects of non-randomized answer options. After all, in Math Garden the answer options are randomized so that the difficulty of the item is not determined by the position (1 – 4) of the correct answer option. It stands to reason that paper-and-pencil tests will carefully have to balance the position of the correct answer options among items that measure the same cognitive construct.

The size of the CAPCT item bank further has the potential to create non-adaptive paper-and-pencil tests that measure understanding of single CS concepts (e.g., If-At). Because the answer options include common misconceptions (e.g., executing all visible arrows) as the incorrect answers, repeated errors may signal teachers which instructions their pupils may benefit from most. Research could explore whether teachers could

actually benefit from informal testing of single CS concept assessments and determine the consequential validity of the CAPCT for everyday use.

A third focus should be on establishing the convergent and divergent validity of the CAPCT versus other available assessments. Román-González and colleagues already conducted a convergent validity study of the Computational Thinking test, the Bebras, and Dr. Scratch (Román-González et al., 2019). However, a large-scale international study across all six existing and CAPCT assessments would greatly benefit the field. Comparing results across different nationalities may also shed light on cross-cultural and language-based differences in the understanding of block-based programming languages.

Convergent and divergent validity studies should be considered from the perspective of multiple-choice assessments, but also from the perspective of natural programming contexts. For example, the multiple-choice questions of the CAPCT can easily be converted to an open-answer environment (i.e., drag-and-drop). In an open-answer environment it is easy to constrict the search-space for the correct answer by allowing players to use only a limited set of building blocks. Comparing the results of multiple-choice and (un)restricted open-answer items may shed light on how multiple-choice assessments compare to real-life programming contexts.

The method of data collection did not lend itself for analysis of the effect of prior programming and CS experience. Nor were we able to determine whether children with high levels of ability on the Codetaal game would also demonstrate high levels of ability in other programming contexts (e.g., creating a game of pong with Scratch; Resnick et al., 2009). The significance of prior Mathematical ability on CAPCT ability ratings may indicate the need for external validity research of prior CT/CS or programming ability on ability.

5. Conclusion

Altogether the current study and the created assessment take a significant step towards programming concepts understanding assessments that are generic, easy to implement, and low in the required resources. The Computerized Adaptive Programming Concepts Test (CAPCT) showed good psychometric qualities with 75% of the variance in item difficulty explained by item characteristics, and an average of 64% of the variance in player ability explained by demographical data, number of responses, and prior mathematical ability. As children proved more able than we expected, future versions of the CAPCT will include more complicated concepts and open-answer questions.

Notes

1. Available via the Open Science Framework: <https://osf.io/bmf47>.
2. Before conducting the study, ethical approval was granted by the Ethics Review Board from the Psychological Methods department at the University of Amsterdam (2018-PML-8835). Users of Math Garden provide informed consent prior to using the products and have the option to opt-out of their data being used for research.
3. We have created overview images of each item, together with response percentages, to allow for post-hoc exploration. These overviews are available via <https://osf.io/bmf47>.

Acknowledgments

We would like to acknowledge the contributions of prof. dr. J.M. Voogt (University of Amsterdam, the Netherlands) for sharing her expertise on Computational Thinking. In addition, we thank Oefenweb by Prowise – owners of Math Garden – for their contributions to hosting & collecting data on the Computerized Adaptive Programming Concepts Test.

Disclosure statement

Funding. This research was conducted as a Thesis project for the Research Master Psychology at the University of Amsterdam. Consequently, no external funding was received for conducting this research.

Employment. H. L. J. van der Maas was employed as a senior advisor at the company who owns Math Garden, **while** the study was conducted. S. A. M. Hogenboom was employed by the same company for a year and a half **after** the study was conducted. The company has not affected the outcome of this research in any way and has agreed to make the materials of the Computerized Adaptive Programming Concepts Test freely available for researchers and teachers. Please note that both authors are **no longer** employed by the company.

Notes on contributors

Sally A.M. Hogenboom is a PhD candidate with the Institute for Logic, Language and Computation at the University of Amsterdam. Her work focusses on the impact of exposure to (online) media and stereotypical beliefs and actions. The present work is, however, part of her Research Master Psychology thesis – also at the University of Amsterdam – which focused on current developments in Computational Thinking / Programming Education.

Felienne F. J. Hermans is an associate professor at the Leiden Institute of Advanced Computer Science at Leiden University. She specializes in the instructional strategies and mental models of programming, and the creation and evaluation of educational programming languages.

Han L. J. Van der Maas is a professor at the department of Psychological Methods at the University of Amsterdam. He specializes in formalizing and testing psychological theories in areas such as cognition, expertise, development, attitudes and intelligence.

ORCID

Sally A. M. Hogenboom  <http://orcid.org/0000-0003-3222-0019>

Felienne F. J. Hermans  <http://orcid.org/0000-0003-0722-0156>

Han L. J. Van der Maas  <http://orcid.org/0000-0001-8278-319X>

Data availability statement

We have made an overview of the contents of the Computerized Adaptive Programming Concepts Test (CAPCT) available via the Open Science Framework (<https://osf.io/bmf47>). This includes a Design Guide and Item Overviews. The Design Guide allows readers to understand how/why certain item types were created, and importantly how the (in)correct answer options were selected. The Item Overviews (10 items per item type) allow for a brief exploration of what makes an item easy/difficult. The complete set of materials of the CAPCT item bank are available upon request.

References

- Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, 75(part B), 661–670. <https://doi.org/10.1016/j.robot.2015.10.008>
- Basogain, X., Olabe, M. Á., Olabe, J. C., & Rico, M. J. (2018). Computational thinking in pre-university blended learning classrooms. *Computers in Human Behavior*, 80, 412–419. <https://doi.org/10.1016/j.chb.2017.04.058>
- Bers, M. U. (2010). The TangibleK robotics program: Applied computational thinking for young children. *Early Childhood Research and Practice*, 12(2), 1–20. <https://eric.ed.gov/?id=EJ910910>
- Brinkhuis, M., Savi, A., Hofman, A., Coomans, F., van der Maas, H., & Maris, G. (2018). Learning as it happens: a decade of analyzing and shaping a large-scale online learning system. *Journal of Learning Analytics*, 5(2), 29–46. <https://doi.org/10.18608/jla.2018.52.3>
- Brunner, E., Bathke, A. C., & Konietzschke, F. (2018). *Rank and pseudo-rank procedures for independent observations in factorial designs – Using R and SAS*. Springer Series in Statistics. Springer International Publishing. <https://www.springer.com/gp/book/9783030029128>
- Cao, J., Fleming, S. D., Burnett, M., & Scaffidi, C. (2015). Idea garden: Situated support for problem solving by end-user programmers. *Interacting with Computers*, 27(6), 640–660. <https://doi.org/10.1093/iwc/iwu022>
- Centraal Bureau der Statistiek. (2018). *Aantal scholen in het primair onderwijs*. Centraal Bureau der Statistiek. <https://www.onderwijsincijfers.nl/kengetallen/po/instellingen/aantallen-instellingen-po>
- Chakarov, A. G., Recker, M., Jacobs, J., Horne, K. V., & Sumner, T. (2019). Designing a middle school science curriculum that integrates computational thinking and sensor technology. *SIGCSE 2019 – Proceedings of the 50th ACM Technical Symposium on Computer Science Education, USA*, 818–824. <https://doi.org/10.1145/3287324.3287476>
- Chang, C.-K. (2014). Effects of using alice and scratch in an introductory programming course for corrective instruction. *Journal of Educational Computing Research*, 51(2), 185–204. <https://doi.org/10.2190/EC.51.2.c>
- Coomans, F., Hofman, A., Brinkhuis, M., van der Maas, H. L. J., & Maris, G. (2016). Distinguishing fast and slow processes in accuracy – Response time data. *PLoS ONE*, 11(5), e0155149. <https://doi.org/10.1371/journal.pone.0155149>
- Dagienè, V., & Sentance, S. (2016). It's computational thinking! Bebras tasks in the curriculum. In A. Brodnik & F. Tort (Eds.), *Informatics in schools: Improvement of informatics knowledge and perception* (pp. 28–39). Springer International Publishing.
- de Mooij, S. M. M., Raijmakers, M. E. J., Dumontheil, I., Kirkham, N. Z., & van der Maas, H. L. J. (2020). Error detection through mouse movement in an online adaptive learning environment. *Journal of Computer Assisted Learning*, July, 37(1), 242–252. <https://doi.org/10.1111/jcal.12483>
- Durak, H. Y., & Saritepeci, M. (2018). Analysis of the relation between computational thinking skills and various variables with the structural equation model. *Computers & Education*, 116, 191–202. <https://doi.org/10.1016/j.compedu.2017.09.004>
- Embretson, S. E. (1998). A cognitive design system approach to generating valid tests: Application to abstract reasoning. *Psychological Methods*, 3(3), 380–396. <https://doi.org/10.1037/1082-989X.3.3.380>
- Franklin, D., Conrad, P., Boe, B., Nilsen, K., Hill, C., Len, M., Dreschler, G., Aldana, G., Almeida-Tanaka, P., Kiefer, B., Laird, C., Lopez, F., Pham, C., Suarez, J., & Waite, R. (2013). Assessment of computer science learning in a scratch-based outreach program. *SIGCSE 2013 – Proceedings of the 44th ACM Technical Symposium on Computer Science Education, USA*, 371–376. <https://doi.org/10.1145/2445196.2445304>
- Google for Education. (2020). *Code with Google*. https://edu.google.com/code-with-google/?modal_active=none&story-card_activeEl=enhance-any-subject
- Grover, S., & Basu, S. (2017). Measuring student learning in introductory block-based programming: Examining misconceptions of loops, variables, and Boolean logic. *SIGCSE 2017 - Proceedings of*

the 2017 ACM SIGCSE Technical Symposium on Computer Science Education, USA, 267–272. <https://doi.org/10.1145/3017680.3017723>

- Grover, S., & Pea, R. (2013). Computational Thinking in K-12: A Review of the State of the Field. *Educational Researcher*, 42(1), 38–43. <https://doi.org/10.3102/0013189X12463051>
- Hofman, A. D., Jansen, B. R. J., de Mooij, S. M. M., Stevenson, C. E., & van der Maas, H. L. J. (2018). A solution to the measurement problem in the idiographic approach using computer adaptive practicing. *Journal of Intelligence*, 6(1), 14. <https://doi.org/10.3390/jintelligence6010014>
- International Society for Technology in Education (ISTE), & Computer Science Teachers Association (CSTA). (2011). *Computational thinking. teacher resources* (2nd ed.). <http://csta.hosting.acm.org/csta/csta/Curriculum/sub/CompThinking.html>
- Jansen, B. R. J., Hofman, A. D., Savi, A., Visser, I., & van der Maas, H. L. J. (2016). Self-adapting the success rate when practicing math. *Learning and Individual Differences*, 51, 1–10. <https://doi.org/10.1016/j.lindif.2016.08.027>
- Klinkenberg, S. (2014). High speed high stakes scoring rule. In M. Kalz & E. Ras (Eds.), *Computer assisted assessment. Research into E-assessment* (pp. 114–126). Springer International Publishing.
- Klinkenberg, S., Straatemeier, M., & van der Maas, H. L. J. (2011). Computer adaptive practice of Maths ability using a new item response model for on the fly ability and difficulty estimation. *Computers and Education*, 57(2), 1813–1824. <https://doi.org/10.1016/j.compedu.2011.02.003>
- Mair, P., & Wilcox, R. (2019). Robust statistical methods in R using the WRS2 package. *Behavior Research Methods*, 52(2), 464–488. <https://doi.org/10.3758/s13428-019-01246-w>
- Mannila, L., Heintz, F., Kjällander, S., & Åkerfeldt, A. (2020). Programming in primary education: Towards a research based assessment framework. *WiPSCE '20: Proceedings of the 15th Workshop on Primary and Secondary Computing Education, USA*, 1–10. <https://doi.org/10.1145/3421590.3421598>
- Maris, G., & van der Maas, H. (2012). Speed-accuracy response models: Scoring rules based on response time and accuracy. *Psychometrika*, 77(4), 615–633. <https://doi.org/10.1007/s11336-012-9288-y>
- Moreno León, J., Robles, G., & Román González, M. (2015). Dr. Scratch: Automatic analysis of scratch projects to assess and foster computational thinking. *RED. Revista De Educación a Distancia*, 46, 1–23. <https://www.um.es/ead/red/46/>
- Rachmatullah, A., Akram, B., Boulden, D., Mott, B., Boyer, K., Lester, J., & Wiebe, E. (2020). Development and validation of the middle grades computer science concept inventory (MG-CSCI) assessment. *Eurasia Journal of Mathematics, Science and Technology Education*, 16(5), em1841. <https://doi.org/10.29333/ejmste/116600>
- Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. (2009). Scratch: Programming for all. *Communications of the ACM*, 52(11), 60–67. <https://doi.org/10.1145/1592761.1592779>
- Román-González, M. (2015). Computational thinking test: Design guidelines and content validation. *Proceedings of the 7th Annual International Conference on Education and New Learning Technologies (EDULEARN 2015), Barcelona*, 2436–2444. <https://doi.org/10.13140/RG.2.1.4203.4329>
- Román-González, M., Moreno-León, J., & Robles, G. (2019). Combining assessment tools for a comprehensive evaluation of computational thinking interventions. In S.-C. Kong & H. Abelson (Eds.), *Computational thinking education* (pp. 79–98). Springer. https://doi.org/10.1007/978-981-13-6528-7_6
- Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the computational thinking test. *Computers in Human Behavior*, 72, 678–691. <https://doi.org/10.1016/j.chb.2016.08.047>
- Román-González, M., Pérez-González, J. C., Moreno-León, J., & Robles, G. (2018). Extending the nomological network of computational thinking with non-cognitive factors. *Computers in Human Behavior*, 80, 441–459. <https://doi.org/10.1016/j.chb.2017.09.030>
- Sáez-López, J. M., Sevillano-García, M. L., & Vazquez-Cano, E. (2019). The effect of programming on primary school students' mathematical and scientific understanding: Educational use of mBot. *Educational Technology Research and Development*, 67(6), 1405–1425. <https://doi.org/10.1007/s11423-019-09648-5>

- Shumway, R. H., & Stoffer, D. S. (2017). Time series regression and exploratory data analysis. In: Time Series Analysis and Its Applications. Springer Texts in Statistics. Springer, Cham. https://doi.org/10.1007/978-3-319-52452-8_2
- SLO. (2015). *Voorbeeld leerdoelen computational thinking*. <https://slo.nl/vakportalen/vakportaal-digitale-geletterdheid/computational-thinking/voorbeeld-leerdoelen>
- Tang, X., Yin, Y., Lin, Q., Hadad, R., & Zhai, X. (2020). Assessing computational thinking: A systematic review of empirical studies. *Computers and Education*, 148, 103798. <https://doi.org/10.1016/j.compedu.2019.103798>
- CS Unplugged (2020). *Computer Science without a computer*. <https://csunplugged.org/en>
- van der Linden, W. J., & Hambleton, R. K. (1997). *Handbook of modern item response theory*. Springer, New York, NY. <https://doi.org/10.1007/978-1-4757-2691-6>
- van der Maas, H. L. J., & Nyamsuren, E. (2017). Cognitive analysis of educational games: The number game. *Topics in Cognitive Science*, 9(2), 395–412. <https://doi.org/10.1111/tops.12231>
- Vivian, R., Franklin, D., Frye, D., Peterfreund, A., Ravitz, J., Sullivan, F., Zeitz, M., & Mcgill, M. M. (2020). Evaluation and assessment needs of computing education in primary grades. *ITICSE '20: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education, USA*, 124-130. <https://doi.org/10.1145/3341525.3387371>
- Voogt, J., Brand-Gruwel, S., & Van Strien, J. (2017). *Effecten van programmeeronderwijs op computationale denken : Reviewstudie*. Open Universiteit, Heerlen, the Netherlands. <https://dare.uva.nl/search?identifer=14732dea-6d83-410d-8548-20ff8eef8e0f>
- Wainer, H., Dorans, N. J., Eignor, D., Flaugher, R., Green, B. F., Mislevy, R. J., Steinberg, L., & Thissen, D. (2000). *Computerized adaptive testing: A primer*. Routledge, NY. <https://www.routledge.com/Computerized-Adaptive-Testing-A-Primer/author/p/book/9781138866621>
- Wiebe, E., Mott, B. W., London, J., Boyer, K. E., Aksit, O., & Lester, J. C. (2019). Development of a lean computational thinking abilities assessment for middle grades students. *SIGCSE 2019 – Proceedings of the 50th ACM Technical Symposium on Computer Science Education, USA*, 456-461. <https://doi.org/10.1145/3287324.3287390>
- Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35. <https://doi.org/10.1145/1118178.1118215>
- World Economic Forum. (2016). *The future of jobs – Reports – World economic forum*. <http://reports.weforum.org/future-of-jobs-2016>
- Wu, C. C., Dale, N. B., & Bethel, L. J. (1998). Conceptual models and cognitive learning styles in teaching recursion. *SIGCSE 1998 - Proceedings of the 29 SIGCSE technical symposium on Computer science education, USA*, 292–296. <https://doi.org/10.1145/273133.274315>
- Zapata-Caceres, M., Martin-Barroso, E., & Roman-Gonzalez, M. (2020). Computational thinking test for beginners: Design and content validation. *IEEE Global Engineering Education Conference, EDUCON, Porto (Portugal)*, 1905–1914. <https://doi.org/10.1109/EDUCON45650.2020.9125368>
- Zhong, B., Wang, Q., Chen, J., & Li, Y. (2016). An Exploration of three-dimensional integrated assessment for computational thinking. *Journal of Educational Computing Research*, 53(4), 562–590. <https://doi.org/10.1177/0735633115608444>

Appendix A

We compiled a list of 19 programming concepts that were deemed appropriate for evaluation in primary school children (ages 4 – 13). The list was compiled after an extensive literature study and consultations with experts in the field of Computer Science, Education, Cognitive Development, and Game Design. From these concepts, seven (denoted with an *) were included in the Computerized Adaptive Programming Concepts Test (CAPCT). The other concepts were excluded based on their inability to be assessed within the design constraints (section 2.1).

ABILITY	OPERATIONAL DEFINITION
Abstraction	<i>The ability to recognize when a specific algorithm may be altered – by the inclusion of parameters – to meet a wider range of task demands. For example; one can walk a square of size 10 by executing “Do 4 times; walk 10 steps forward, turn 90 degrees left.”. Abstraction occurs when the specific indices are swapped for general parameters: “Do x times, walk y steps forward, turn z degrees left.” By entering values for x, y, and z, a wider range of task demands (e.g., walk a triangle) can be met by the same algorithm.</i>
Algorithm*	<i>In its simplest form, algorithms are a sequence of instructions which – when executed – will resolve the problem at hand. As of such, all items that require the execution of two or more instructions are considered algorithms.</i>
Conditionals*	[Included as “Sequences”] <i>If-conditions: a block of code is executed only when a condition is met. For example; if age is below 18 say “You are a child” -> consequently nothing will be said if the age is above 18. If-else conditions: a block is executed when a condition is met, whereas a different block is executed in all other cases. For example, if age is below 18 say “You are a child”, else say “You are an adult”-> consequently depending on age the message will be “You are a child/adult”. Note: in contrast to the if-condition a message will always be said.</i>
Data	<i>Data Structures: The ability to identify the logical data structures for a given context. For example, storing one’s birthday wishes in a list, as opposed to separate variables named “wish_1 = . . . , wish_2 = . . . ”.</i> <ul style="list-style-type: none"> - strings [e.g., a name, a place, a color] - numbers [e.g., birthdates, ages, house-numbers] - unordered lists [e.g., groceries, all names in a class, all streets in a town] - ordered lists [e.g., length of family members, finish positions in a competition] <i>Data Properties: The ability to understand inherent properties of data types. For example; one cannot add a number (1) to a string (“a”).</i> <i>Data Probability: The ability to differentiate between probable and improbable data. For example, the data for all the names in a class are improbable to contain 20 x “Gregg”.</i>
Debugging*	<i>The ability to identify incorrect algorithms (i.e., algorithms that do not meet task demands), and modify them so that task demands are met.</i>
Effectivity	<i>The ability to distinguish between correct solutions (i.e., task demands are met) and incorrect solutions (i.e., task demands are not met).</i>
Efficiency	[Similar to: “Debugging” with the exclusion of modification abilities] <i>The ability to identify the most efficient solutions, in the sense that efficiency constitutes the number of instructions required to meet task demands (e.g., use of loops and functions to reduce the number of instructions), or in the sense that real-life resources (e.g., time or energy) are reduced to a minimum.</i>
Events	<i>The ability to account for trigger-based behaviors; conditionals are specified which – when true – will elicit distinct sets of behaviors. For example; ““walk” [event] until “apple” [event], then execute “eat apple””.</i>
Functions	<i>The ability to identify the output of a function given the input (i.e., parameters). For example; the function “add 2” -> [output = x + 2] will return 4 for x = 2, and 12 for x = 10.</i>
Generalization*	<i>The ability to recognize when problems require the same solutions, or when the application of the same algorithm in different situations will result in equivalent outcomes.</i>
Loops*	[Similar to “Abstraction” with the exception that problems are concrete instead of abstract during “Generalization”] <i>Use of infinite-loops (i.e., repeat a behavior continuously). For example; an automatic clock repeats “add one minute” continuously. Use of for-loops (i.e., repeat the execution of a block of code). For example; repeat 4 times “draw one stroke forward, turn left” (drawing a square). Use of while-loops (i.e., repeat the execution of a block of code while a condition is met). For example; walk forward while no obstacles encountered.</i>
Modularization	<i>The ability to identify segments of code/instructions that are repeated at multiple phases of the algorithm. For example; “stopping at a traffic light” is a behavior that may occur at multiple phases of cycling through a town.</i>
Multiple Agents*	[Similar to “Procedures” with the exception that procedures require an understanding of the behaviors included, whereas “Modularization” is simply the recognition of repeated components.] <i>The ability to identify that two or more conditions must be true, so that task demands can be met. For example; a scale can only be balanced when two weights of equal weight are placed at either end of the scale.</i>
Ordering	<i>The ability to order given data according to a specified criterion. For example; ordering everyone in your class by birthdate.</i>

(Continued)



ABILITY	OPERATIONAL DEFINITION
Parallelization	<i>The ability to recognize that a process may be executed more efficiently (time and resources) when multiple processes are executed simultaneously. For example; two people wish to do laundry (wash & dry). Although both can wait their turn, the process will be faster if the second person starts washing while the first uses the drier.</i>
Pattern Recognition	<i>The ability to identify patterns of meaningful data amongst distractors. For example; identifying the sequence 123 in the data 835-476-123-895</i>
Procedures*	<i>The ability to identify a sub-routine – a section of stand-alone instructions in a larger algorithm. For example; “move left, move right” as dancing [procedure] in an algorithm modelling dancefloor behaviors. [Similar to: “Functions” with the exception that although “Procedures” take input, they do not return output.]</i>
Sorting	<i>The ability to sort given data into separate categories according to a specified criterion. For example; sorting the class into boys and girls.</i>
Variables	<i>Variable Assignment: The ability to understand (multiple) assignments to variables. For example; if the variable “x” is assigned the value 5, and then is assigned the value 8, what is the value of “x”? Answer: 8 (the last assignment to the variable is stored). Variable Names: The ability to differentiate between informative and uninformative variable names. For example; the variable name “right” could indicate properties of a direction (i.e., turning right) and correctness.</i>