Journal of Cognitive Psychology

ISSN: (Print) (Online) Journal homepage: https://www.tandfonline.com/loi/pecp21

The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course

Irene L. Graafsma, Serje Robidoux, Lyndsey Nickels, Matthew Roberts, Vince Polito, Judy D. Zhu & Eva Marinus

To cite this article: Irene L. Graafsma, Serje Robidoux, Lyndsey Nickels, Matthew Roberts, Vince Polito, Judy D. Zhu & Eva Marinus (2023): The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course, Journal of Cognitive Psychology, DOI: 10.1080/20445911.2023.2166054

To link to this article: https://doi.org/10.1080/20445911.2023.2166054

Published online: 18 Jan 2023.

JOURNAL OF COGNITIVE PSYCHOLOGY
https://doi.org/10.1080/20445911.2023.2166054

The cognition of programming: logical reasoning, algebra and vocabulary skills predict programming performance following an introductory computing course

Irene L. Graafsma [a,b,c], Serje Robidoux [b,d], Lyndsey Nickels [b,d], Matthew Roberts [e], Vince Polito [b], Judy D. Zhu [b] and Eva Marinus [f]

[a]International Doctorate for Experimental Approaches to Language and Brain (IDEALAB), Universities of Groningen (NL), Newcastle (UK), Potsdam (GE) and Macquarie University, Sydney, (AU), Groningen, Netherlands; [b]School of Psychological Sciences, Macquarie University, Sydney, Australia; [c]Centre for Language and Cognition Groningen (CLCG), University of Groningen, NL, Groningen, Netherlands; [d]Macquarie University Centre for Reading, Macquarie University, Sydney, Australia; [e]Department of Computing, Macquarie University, Sydney, Australia; [f]Schwyz University of Teacher Education, Goldau, Switzerland

ABSTRACT

In the current study we aimed to determine which cognitive skills play a role when learning to program. We examined five cognitive skills (pattern recognition, algebra, logical reasoning, grammar learning and vocabulary learning) as predictors of course-related programming performance and their generalised programming performance in 282 students in an undergraduate introductory programming course. Initial skills in algebra, logical reasoning, and vocabulary learning predicted performance for generalised programming skill, while only logical reasoning skills predicted course-related programming performance. Structural equation modelling showed support for a model where the cognitive skills were grouped into a language factor and an algorithmic/mathematics factor. Of these two factors, only the algorithmic/mathematics factor was found to predict generalised and course-related programming skills. Our results suggested that algorithmic/mathematical skills are most relevant when predicting generalised programming success, but also showed a role for memory-related language skills.

ARTICLE HISTORY
Received 22 April 2022
Accepted 4 January 2023

KEYWORDS
Programming; coding; cognitive skills; learning; programming success

1. Introduction

Programming education has gained major importance and popularity worldwide. All over the world initiatives have been taken to teach people how to programme, focusing on both children and adults (European Schoolnet, 2015). In response to the growing interest in learning to programme, research in this field has also increased. Over the last 30 years, most efforts have come from the computer science education community, which has largely focused on different ways of teaching programming and the goals and motivations of learners (Guzdial, 2016). However, a smaller group within this community has also examined programming from a cognitive perspective. Guzdial and du Boulay (2019) identified two main streams of research with different objectives. One stream focuses on whether, and if so which, cognitive benefits accrue from learning to programme (e.g. Pea & Kurland, 1984). The other stream researches which cognitive skills are important when learning to computer program. The current study aimed to contribute to this second stream.

The cognitive skills underpinning learning to programme have been examined by earlier studies, mostly conducted before the mid-1990s (e.g. Pena & Tirre, 1992; Shute, 1991; Webb, 1985). After this, interest in programming education temporarily decreased, arguably because the transition to

CONTACT Irene L. Graafsma i.l.graafsma@rug.nl irenegraafsma@gmail.com International Doctorate for Experimental Approaches to Language and Brain (IDEALAB), Universities of Groningen (NL), Newcastle (UK), Potsdam (GE) and Macquarie University, Sydney, (AU), Groningen 9700 AB, Netherlands

interface-based software and computers removed the need for programming for most users (HackerRank, 2018). Over the last decade, the number of people who need to learn to programme has been increasing again. This is partly because the increase in the use of technology demands more software engineers (Seegerer et al., 2019). In addition, in many jobs, there has been a shift from usage to creation, with a range of professions expected to programme their own content, including designing their own websites or developing data analysis programmes (Rushkoff, 2012). Overall, currently more jobs rely on understanding the code that drives technology than was the case in previous decades.

As a consequence of the increased importance of programming in professional life, demand for programming education has increased. This has changed the current learning context in three ways. First, the number of students studying introductory programming courses has increased, resulting in more large-scale lectures and a need for more independent learning (Marasco et al., 2017). Second, students with a wider variety of backgrounds and with different strengths and weaknesses are now learning to programme. Finally, because of this diversification, instruction methods have been simplified (Guzdial, 2003; Kelleher & Pausch, 2003) and programming languages have been developed to more closely resemble natural languages to better facilitate learning for beginners (Fedorenko et al., 2019; O'Regan, 2012; Paulson, 2007). These changes mean that the skills involved in learning to programme today may be different from the skills found in the early studies of over 20 years ago. The current study therefore examined which cognitive skills are important in the modern-day context, focusing on answering the questions: "Which cognitive skills predict programming success in current undergraduates?" And, "To what extent do language and algorithmic /mathematical skills predict programming success?" Before describing the current study, we first give a short review of the results of the earlier studies that focused on cognitive skills in learning to programme.

1.1. Background

Overall, the results of early studies suggest that problem solving, logical reasoning, algebra and verbal skills are most strongly related to learning computer programming regardless of the age of the learners and specific programming course. Specifically, Pena and Tirre (1992) found that new army recruits with better general verbal knowledge (particularly in the area of general science), reasoning skills (such as recognising patterns and conditions in picture series), algebra word problem solving (particularly problem translation and problem decomposition), and working memory capacity showed better programming skill acquisition at the end of a 1.5-hour Pascal tutorial. In the context of a half-day BASIC programming course for 11–14-year-olds, Webb (1985) found that mathematical skill and non-verbal reasoning predicted knowledge of programming syntax. Finally, Shute (1991) found that, after a longer (7-day) programming course (for Pascal), working memory and word problem solving showed the highest correlations with learning progress during the course for high school students. More recent studies have also shown that general mathematical skills can predict programming performance (Bennedsen & Caspersen, 2005; Quille & Bergin, 2018, 2019). However, these studies did not look at the specific cognitive skills that may underlie mathematical ability.

These findings give an idea of the cognitive skills that may be involved when first learning to programme. However, the studies also have three important limitations. First, Pena and Tirre (1992) and Shute (1991) administered the cognitive tests either during or after the programming course. This means that it is possible that the cognitive skills had been affected by the programming tutorials. In other words, because of this testing structure, we cannot disentangle the effects that training may have had on programming ability from the effects that training may have had on the cognitive skills themselves. Second, because the tutorials in all three studies were relatively short, it remains unclear whether these cognitive skills are also important for learning over a longer time span. Finally, all three studies used programming scores within the tutorials as their measure of programming skill. Although this provides a measure of direct learning in the course, it does not address whether participants acquired programming skills that generalise to different programming settings and languages. This design also means that the programming measures from different studies are difficult to compare, because the tests are different in each study and are not standardised or validated.

The current study aimed to address these limitations in three ways. First, by administering cognitive

skill tests at the very start of the course, before any programming had been learned, we were able to determine whether initial cognitive skills at the start of the course predicted programming performance at the end. This methodology is in line with other more recent studies such as Prat et al. (2020) who also administered cognitive tests before studying programming in an online learning environment. The current study applied this in the context of classroom learning. Second, by testing students over a 12-week course, we were able to test how cognitive skills related to long-term learning in a typical university course. Finally, by using both the scores from the course assessments, as well as scores on an independent programming test conducted at the end of the course, we were able to assess both course-related and generalised programming skills. Moreover, the use of an independent programming test also allowed for easier comparison with other studies that use that same test.

The selection of cognitive skills was based on three sources. First, we looked at the findings of the previously discussed studies from the 1980s and 1990s, which suggested that problem solving, logical reasoning, algebra and verbal skills correlated with programming ability. Second, we examined the programming aptitude tests used by IBM (IBM, 1968) to see which skills experts thought may be important when learning to programme. These aptitude tests include elements assessing pattern recognition, algebra and logical reasoning, with problem solving as a component of the different algebra tests. Finally, our theoretical framework was the PGK-hierarchy of the programming process, which was named after its creators Perrenet, Groote and Kaasenbrood (Perrenet et al., 2005) and was further described by Armoni (2013). This theoretical framework further guided our research.

The PGK-hierarchy postulates that writing a computer programme to solve a certain problem involves steps at four different levels. We will explain the four levels using an example where a programmer is asked to programme an animation. At the highest level, the problem level, the programmer considers the solution the problem demands, and considers aspects of the problem such as solvability and complexity. In our example, at this level, the programmer determines what the animation should look like (e.g. shapes, colours, movements) and how difficult it will be to design these components. At this level, the programmer does not yet make specific plans around how to solve the problem. The second level is the object level. At this level, an algorithm (a plan detailing the specific steps that must be executed by the programme) is developed to solve the problem. Here the programmer specifies which functions or programming objects should be created, and in which order, for the animation to be presented as planned. Here the programmer may decide on whether to use techniques like loops or recursion to implement particular sections of the animation. However, the algorithm is not yet associated with a specific programming language, which happens in the third level, the programme level. In our example, the programmer will now select the language that best suits the algorithms and look up the specific functions in that programming language, and will write out the code with the correct terms and syntax. The lowest level of the PGK-hierarchy is the execution level, which is the interpretation and execution of the algorithm by the computer. Because it does not relate to human cognitive skills, this level is not relevant to the aims of this study.

From the PGK-hierarchy's description (Armoni, 2013; Perrenet et al., 2005), the first two levels seem to rely on algorithmic thinking (deriving a solution by defining the specific steps necessary to solve a problem) and mathematical skills. We therefore hypothesised that the cognitive skills shown to be important by the previous literature are related to these two levels. In particular problem solving, algebra, logical reasoning, and pattern recognition are relevant here. The third level of the PGK-hierarchy, the programme level, addresses the requirement to use specific programming languages. The relationship between specific language skills and programming performance has not yet been empirically tested, but several researchers have argued for a connection (Fedorenko et al., 2019; Floyd et al., 2017; Hermans & Aldewereld, 2017; Jacob & Warschauer, 2018; Siegmund et al., 2014; Vee, 2013; Vogel et al., 2019). In addition, indirect evidence from the second language learning literature can direct us to relevant language skills that may predict programming success.

Fedorenko et al. (2019) argued that throughout its existence, programming has been regarded as related to natural languages, with some educational systems including it as part of the foreign language curriculum. However, over the last decade the focus has shifted. Programming has been increasingly described as a Science, Technology, Engineering, and Mathematics (STEM)

subject, neglecting the language skills that may play a role in learning this skill. Hermans and Aldewereld (2017) argued against this shift away from natural languages and emphasised the similarities between programming and natural language writing. They maintained that both skills rely on high-level planning and problem solving at the start, and on specific structure and style rules at the later implementation stages. Similarly, a review by Pandža (2016) argued for more research on programming from a second language learning perspective. This is particularly pertinent given that modern programming languages were initially designed to resemble natural languages with the idea that users could rely on their natural language skills when using a programming language (Fedorenko et al., 2019; Paulson, 2007). As a result, it makes sense to expect that natural language skills are predictors of the ease of programming language acquisition.

The second language acquisition literature indicates that successful natural second language learning relies on phonemic coding ability (to discriminate and encode foreign sounds), grammatical sensitivity (to recognise functions of words in sentences), inductive language learning ability (to infer or induce rules from samples), and memory and learning (to make and recall associations between words and phrases in a first and second language) (Skehan, 1991). Learning a programming language involves learning the syntax rules and the specific functions and commands for that language. This could be considered comparable to learning grammar and vocabulary in a natural language. Therefore, we hypothesised that programming may rely on grammatical sensitivity and inductive language learning ability for syntax acquisition, and memory and vocabulary learning for memorisation of language-specific terms and functions.

1.2. Research questions and hypotheses

The current study examined which cognitive skills predict programming success in undergraduates. To achieve this aim, we tested five different cognitive skills: logical reasoning, pattern recognition, algebra, which we hypothesised relate to the first and second levels in the PGK-hierarchy; and vocabulary learning and grammar learning, which we hypothesised relate to the third level of the PGK-hierarchy. If this theory is correct, we would also expect the first three skills to be related to each other, while language skills may be relatively independent. Consequently, we tested whether the cognitive skills selected could be grouped into an algorithmic/mathematics factor (pattern recognition, algebra and logical reasoning) and a language factor(vocabulary learning and grammar learning). We predicted that both factors would predict final programming performance and examined the extent to which each factor contributed to the prediction of programming success.

2. Methods

2.1. Participants

All 838 students in an "Introductory Programming" course COMP115 at Macquarie University in the first semester of 2019 were required to complete the experimental tasks as part of the first and last tutorials of the semester. During the semester, participants learned to programme in "Processing", a simple programming language based on Java that is specifically intended for beginners to learn to code in a visual context. Although participation in the testing sessions was mandatory, only the students who gave written consent for their data to be used for research and who completed both testing sessions were included ($n = 344$). Participants were excluded from analysis if they rated their own level of English below "Good", which was 3 on a 5-point rating scale from 1 (minimal) to 5 (native). Participants were also excluded if, when probed at the end of the study, they indicated that they did not seriously attempt the tests, took notes when this was not allowed, or worked together. After removing these participants ($n = 62$), the sample consisted of 282 participants (49 female, 204 male, 2 other, 27 no gender given, mean age 19.32 years, $SD = 3.09$). For 129 participants this was their first programming experience. The majority of participants were enrolled in Engineering and Information Technology degrees (67%), but we also included students from science; business; education; environmental studies; health and medical sciences; security and intelligence; media; creative arts and communication; and society, history and languages. For most students, the current course was part of their mandatory study programme. The study received ethical approval from the Macquarie University Human Research Ethics Committee (Reference number: 5201800224).

2.2. Materials

The results reported here are part of a larger study. In context of that study we used two different versions for some of the tests listed below. When two different versions of a test were used, students were assigned a version based on the final digits of their assigned university student numbers which are assumed to be random, and scores on the tests were standardised to eliminate any differences in difficulty across versions. All tests were presented in a Qualtrics survey (Qualtrics, Provo, UT), see Section 2.3 "Procedure" for details.

2.2.1. Primary outcome measures

Our primary outcome measure was programming skill. We assessed this with two different measures: the Second Computer Science 1 Short (SCS1-Short) test, to measure generalised programming performance, and the students' course grades to measure course-related programming performance.

Second computer science 1 short (SCS1-short). This test is based on the Second Computer Science 1 (SCS1; Parker et al., 2016). The reliability of the original test was reported by Parker et al. as a Cronbach's alpha of .59 (Parker et al., 2016). We used a computer-based version of this test with test items split into two parallel versions, as described in Graafsma et al. (2020). Each version consisted of 13 questions, and covered the following topic areas: basics (i.e. applying simple mathematical formulae), conditionals, for loops, indefinite/while loops, logical operators, arrays, recursion, function parameters, and function return values. The division of item topics and types across versions can be found in the Appendix. In our validation study (Graafsma et al., 2020), based on the testing session at the end of the programming course we found a Cronbach's alpha of .29 for SCS1-Short version 1 and a Cronbach's alpha of .55 for SCS1-Short version 2. Reliability indices for these short versions were low, perhaps indicating that items on these scales tapped multiple domains of programming skill. Because SCS1-S version 1 had a lower reliability than version 2 we re-ran the analyses in this paper while including only participants who completed version 2 of the SCS1-S. With this subset of participants the findings remained the same. Therefore, the low reliability of version 1 does not seem to have affected the results. In this paper, we therefore included all participants regardless of SCS1-S version. At the end of the programming course the average accuracy was 29% correct for Version 1 (3.81 (SD = 1.88) out of 13), and 32% correct (4.18 (SD = 2.30) out of 13) for Version 2. The scores on the two versions did not differ significantly ($t(324.63) = -1.61$, $p = .11$). The SCS1 uses a pseudocode programming language specifically developed for this test. Participants were given a pseudocode guide which they could consult for information about the syntax and features of the programming language whilst completing the test. This guide could be accessed in a separate browser window by clicking a button in the Qualtrics survey. Participants were given 30 min to complete as many questions as possible. They could answer the questions in any order by scrolling back and forth between the questions in the online test environment (Figure 1).

Course grades. This outcome measure comprised the students' grades on the main course assessments of their course in the programming language "Processing". The main assessment was split over five module tests, each consisting of open questions where students solved small programming problems or answered conceptual questions. The five topics of the modules were: variables & conditionals, loops, functions, arrays & strings, and programme design & problem solving. Students were given three attempts for each module assessment. They were free to complete these during various exam sessions throughout the course, or during the exam session held two weeks after our testing session with the SCS1-Short. For each module, the student's highest score counted towards their final

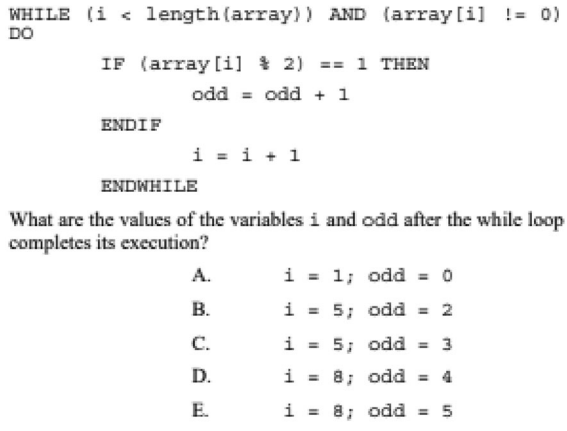

Figure 1. Example of SCS1-Short question. Note. This figure presents question 5 of Version 1 of the SCS1-Short.

grades. For more information see the Unit Guide for this course which is available in the Cognition of Coding project on the Open Science Framework (https://osf.io/8nax4/?view_only=eb0341df544b435792e436451929e2cd). In the current study we used each student's best raw module scores averaged over the subtopics, and disregarded penalties for lack of attendance, incomplete work or late submissions. We also excluded grades from a sixth additional module related to the history of computing, as it did not measure programming skills. We chose to use the students' best scores rather than first attempt scores, because motivation for first attempts varied too widely, with some students using the first attempt just to read through the questions and evaluate their scope and difficulty (Figure 2).

2.2.2. Predictor measures

Here we describe the five cognitive skill tests: logical reasoning, pattern recognition and algebra were categorised as mathematical/algorithmic skills and vocabulary learning and grammar learning as language skills.

Logical reasoning. Logical reasoning was tested with syllogisms (Handley et al., 2002). Each item consisted of statements such as "If it is a rectangle then it is purple. It is a rectangle. It is not purple." Participants had to evaluate whether the final statement ("It is not purple.") followed logically and with certainty from the previous two statements ("If it is a rectangle then it is purple. It is a rectangle."). The test consisted of 16 items and participants were given five minutes to complete as many as they could. We used two parallel versions. Version 1 used the exact items from Handley et al. (2002) and Version 2 used the same questions but with different shapes and colours (e.g. "If it is an oval then it is not black.").

Pattern recognition. We assessed pattern recognition with "Part 1, Number Series" from the Programming Aptitude Test (IBM, 1968) (see footnote 1), which we split into two parallel versions with alternating even and uneven question numbers in each version (i.e. Version 1 included items 1,4,5,8 … etc, Version 2 items 2,3,6,7,10, etc.) to ensure equal difficulty. Each item presented the participant with a series of six numbers, from which the participant had to deduce the pattern, and then, following the pattern, select the number that would come next in the sequence from amongst five alternatives (e.g. question: 3 6 9 12 15 18, answer options: 19 20 21 22 23). The test consisted of 13 items and participants had five minutes to complete as many items as possible.

Algebra. We measured algebra skill with an adapted version of the "Part 3, Arithmetic Reasoning" subtest of the Programming Aptitude Test (IBM, 1968).[1] In the original test the items comprised mathematical word problems for which participants had to pick the correct answer from five answer options consisting of numbers. For the current study, the test was given a more abstract format that did not rely as heavily on arithmetic. To do so, we replaced precise numbers (such as 22 degrees) with abstract variables (such as T1), and we formulated four multiple choice options as alternative formulae to solve the problem (one target and three distractors). Participants were required to choose the formula that would result in the correct answer. As an example, we present item 3 from Version 1:

"The temperature at 1:00 pm was T1 and at 6:30 pm it was T2. Assuming a constant rate of change, what was the temperature at 4pm?".

With answer options:

(a) T2−((6.5−1)(T1−T2)/(4−1))
(b) (4−1)(T1−T2)/(6.5−1)
(c) T1−((6.5−1)(T1−T2)/(4−1))
(d) T1−((4−1)(T1−T2)/(6.5−1))

As in the pattern recognition task, we split the 20-item version of the test into two parallel versions of 10 questions each, by alternating even and uneven question numbers in each version.

Vocabulary learning. We developed this test based on the vocabulary learning subtest of the LLAMA language aptitude test (Rogers et al., 2017). Participants were instructed to memorise the written names of 20 creatures displayed in pictures and were told that they would be tested on them later. The 20 pictures of novel creatures were selected from Romanova (2015), and were paired with 20 novel words (e.g. CEKEL, as shown in Figure 3) specifically developed for the current

[1]PAT Use Courtesy of International Business Machines Corporation, © International Business Machines Corporation.

Suppose that you have two integer variables called a and b and that a has a value between -10 and 10 (inclusive).

Consider the code below.
List which initial values of a will cause b to be assigned the value 1.
If no values of a satisfy this condition, explain why.

```
if ((a <= 7) || (a < 0)) {
  if (a <= 3) {
    b = 1;
  } else {
    b = 2;
  }
}
```

Figure 2. Example of course exam question. Note: This figure presents a question on the topic of "variables and conditions" from the course exam.

study. All creatures with their names were simultaneously presented on the screen, and participants were given 2.5 min to memorise the pairs without being allowed to take any notes. Participants were tested immediately after the learning phase and again, in a delayed recall test, 30 min later. During the test phases, all of the creatures' pictures and names were displayed on the screen and participants were given 3.5 min to drag and drop the names under the corresponding pictures. Two parallel versions of this test were used, which consisted of different pictures and names of novel creatures, with the length of the names matched across the two versions.

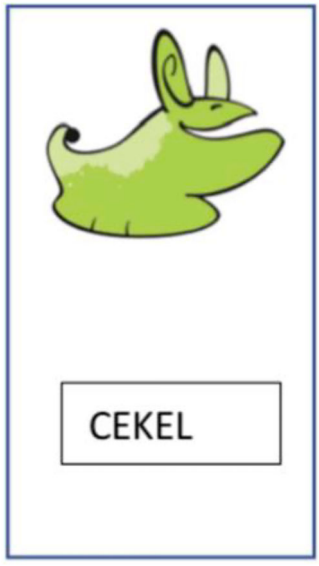

Figure 3. Example of a learning item on the vocabulary learning test. [To view this figure in color, please see the online version of this journal.] Note: This figure shows one of 20 learning items on the vocabulary learning test.

Grammar learning. We based this test on the grammar learning subtest of the LLAMA (Rogers et al., 2017) and on Part 4 of the Pimsleur Language Aptitude Battery (PLAB) (Pimsleur et al., 2004). The test consisted of nine sentence-image pairs from which participants had to deduce the rules of the grammar of an artificial language. For example, the sentence "unak-ek ipot-arap" described two red-circle creatures walking underneath a rectangle (see Figure 4). Across the training items, participants could deduce that "unak-ek" meant two creatures walking under, while "ipot-arap" meant red circles. Participants were presented with 20 questions where they were asked to select the grammatically correct descriptions of new pictures from four answer options. The test was structured in such a way that students could scroll back and forth through all the examples and questions, thus not having to memorise the vocabulary or grammar. Students were given eight minutes to complete the test.

Demographics. We collected information about the participants' age, gender, degree major, knowledge of programming languages, previous programming experience and level of English. We also asked students whether they had completed the tests for the current study according to the instructions (e.g. no calculators permitted). Demographics were used to describe and select the participant sample.

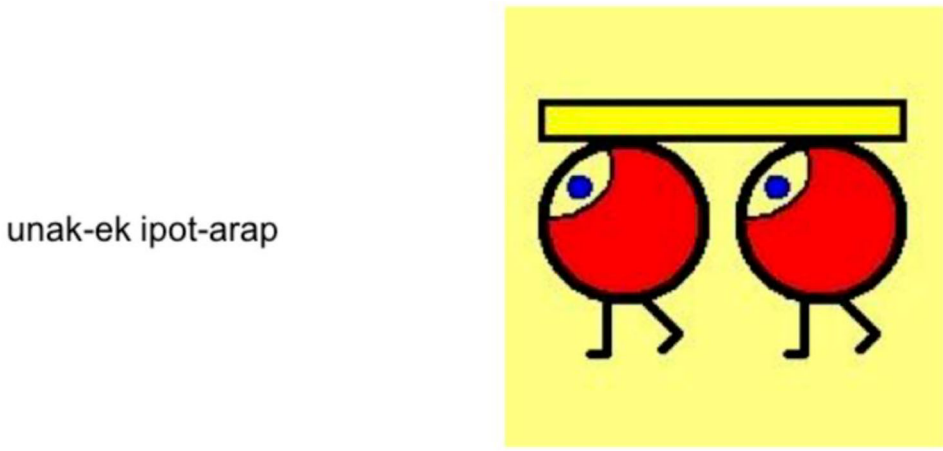

Figure 4. Example of a learning item on the grammar learning test. [To view this figure in color, please see the online version of this journal.] Note: This example from the grammar learning test shows the training sentence on the left that described the image on the right.

2.2.3. Tests not used for the current study

As part of a larger research project two more measures were administered that are not included in the current study: a sense of agency scale (Polito et al., 2013), measuring the participant's feelings of control while programming, and a behaviour and personality questionnaire that examines autistic traits in the general population (the Autism Spectrum Quotient; Baron-Cohen et al., 2001).

2.3. Procedure

The testing sessions took place during the first and last tutorials of the programming course and were led by the regular course tutors. Three per cent of the participants included in the study completed the tests at home because they could not attend the tutorials. Participants were informed that the aim of study was to find out which skills are important in learning to computer program. They were not given any specific information about the tests or expectations of the study.

Students were given a link to the Qualtrics surveys in which all tests were presented. The Qualtrics survey first displayed an information sheet about the study and gave them the choice to consent for their data to be used for research. During each time-limited test participants saw a countdown of the remaining time in the corner of the screen. Students were told that they were allowed to use pen and paper for all tests except for the vocabulary learning test.

Students were instructed to complete the tests in the online Qualtrics system. They were asked to do so individually at their own pace. All tests had a time limit that would automatically move the survey on to the next test once the time limit for a particular test was reached. To encourage the students to seriously attempt the tests and not to just skip through them, the button to move on to the next test only became available after one minute for the cognitive tests and after five minutes for the programming test. For each test, instructions were provided on a separate page of the survey before the student could start each test. All instruction pages were displayed for at least 20 s before the student could move on to the next page. In the testing session at the start of the semester, the order of tests was: (1) vocabulary learning and direct recall, (2) pattern recognition, (3) algebra, (4) logical reasoning, (5) vocabulary delayed recall, (6) grammar learning and (7) demographic questionnaire. During the testing session at the end of the semester, the students first completed the SCS1-Short and then the demographics questionnaire. The testing session at the start of the semester took approximately one hour, and the session at the end of the semester took approximately 30 min to complete.

Testing sessions took place in the first and the last week of the semester course, which were approximately 12 weeks apart. During the 12 weeks of semester students undertook the required coursework for the programming course with no additional tasks related to this study.

2.4. Planned analyses

To answer our first research question of which cognitive skills predicted programming success, we used regression analyses. To answer the second question of whether language skills and mathematical/algorithmic skills each predicted programming success we used Structural Equation Modelling. We used both classic Null Hypothesis Significance Testing (NHST) and Bayesian statistical approaches for the pre-processing *t*-tests and for the regression models. For NHST we adopted $\alpha = .05$ and considered results significant at *p*-values below 0.05. For the Bayesian analyses, we report the Bayes factors in favour of the alternative hypothesis (BF_{10}). Bayes factors between 0 and 0.333 show support for the null hypothesis, with lower values showing stronger support. Bayes values between 0.333 and 3 are considered inconclusive. And Bayes values above 3 show support for the alternative hypothesis, with higher values showing stronger support (Rouder et al., 2009). Analyses were primarily conducted in R version 4.0.2 (R Core Team, 2019). Bayesian statistics were computed in JASP version 0.10.2 (JASP Team, 2019).

3. Results

3.1. Pre-processing

For the tests that had different versions (pattern recognition, vocabulary learning, algebra, logical reasoning and SCS1 programming), *t*-tests were used to see whether there were version differences. For each *t*-test we only included those participants who attempted all cognitive tests and would therefore be included in the analysis of this paper ($n = 245$). We found no significant differences between the different versions of any cognitive tests (all $t < 2$, $p > .10$, $BF_{10} < 0.333$). Nevertheless, to eliminate any small differences that may not have reached statistical significance, we computed z-scores for all cognitive tests to standardise each version before further analysis.

For the programming test (SCS1-Short), the difference in performance between the two versions was not significant ($t(233.16) = -1.798$, $p = .073$). However, a Bayesian *t*-test indicated that results were inconclusive ($BF_{10} = .652$). Ideally, we would have computed z-scores for this test as well. However, in a separate validation study (Graafsma et al., 2020) we tested the quality of each SCS1-Short version and whether the versions were parallel. Version 1 was found to be of poorer quality (possibly more difficult, lower external validity and lower internal-consistency reliability) than Version 2. To avoid confounding the standardised z-scores with course grades while still eliminating any potential influence of SCS1-Short version difference or sample difference, we kept the raw test scores but added the "version" to the statistical models as a control variable. This allowed us to control for differences between versions without assuming that the samples performed equally well. We also used raw scores for the course grades.

Exploratory analyses with the demographic data showed that whether or not the participants had previous programming experience correlated positively with programming performance on both outcome measures (correlation with SCS1-Short: $r = .274$, $t(241) = 4.412$, $p = < .001$; correlation with course grades: $r = .324$, $t(239) = 5.296$, $p < .001$). We therefore included this measure (programming experience or not) as a control variable in the regression models.

3.2. Regression models

We used multiple regression to examine which cognitive skills at the start of the semester predicted final, end of semester, performance on the SCS1 programming test. We entered the scores on the cognitive tests for pattern recognition, algebra, logical reasoning, grammar learning and delayed vocabulary recall as predictors. Since immediate recall and delayed recall of the vocabulary learning test were highly correlated ($r = .880$, $p < .001$), we used only delayed recall because this seemed to be the measure that would better mirror the situation in the programming course: testing did not take place immediately after learning. We also added two control variables: whether or not this was the participants' first programming experience, and version of the SCS1-Short. The results are shown in Table 1 and Figure 5. Algebra, logical reasoning, and delayed vocabulary recall were significant predictors of performance on the SCS1. The Bayes factors indicated evidence for logical reasoning and vocabulary learning as predictors, and against pattern recognition. Bayesian results for algebra favoured an effect, but not quite strongly enough to meet the criteria of $BF_{10} > 3$. The BF_{10} for grammar learning skills favoured the null, but again, not quite strongly enough to meet the

criteria of $BF_{10} < .333$. Because the SCS1-Short Version 1 was of lower quality and slightly harder than Version 2 (see Table 1, results for SCS1-Short), we also ran parallel analyses with only the students that completed the SCS1-Short Version 2. The pattern of results was the same as for the full sample so they are not reported separately.

We ran the same multiple regression model with course grades as the dependent variable. The results are shown in Table 1 and Figure 5. We found that only logical reasoning was a significant predictor of course grade. The pattern of the results of the Bayesian analysis was similar: there was evidence for logical reasoning as a predictor of course grade and the results were inconclusive for the predictive value of the other cognitive skills.

3.3. Structural equation modelling

We examined the correlations between the cognitive tests, which are presented in the correlation matrix shown in Table 2. The language tests (vocabulary learning and grammar learning) correlated moderately. The mathematical/algorithmic tests (pattern recognition, algebra, logical reasoning) showed small, but significant, correlations with each other, and also showed some small, significant, correlations with the language tests. In particular, the correlations between logical reasoning and grammar learning, logical reasoning and vocabulary learning, and algebra and grammar learning were comparable in strength to the correlations between the mathematical/algorithmic tests themselves.

We used structural equation modelling (SEM) with chi-squared tests to determine whether a model with both mathematical (containing logical reasoning, algebra and pattern recognition) and language (containing vocabulary and grammar learning) latent variables better explained the correlation structure between the cognitive skills tests than a model with a single general latent variable. Comparison of the two models indicated that the model with separate latent variables for mathematical and language skills explained the data better ($X^2(1) = 6.424$, $p = .011$). Taking this structure, we then used separate latent variables for the mathematical and language skills to predict programming ability in two different models: one model with SCS1-Short as the outcome variable, and one with course grade as the outcome variable. We used five common goodness of fit measures to assess how well the models fit the data (Kline, 2005). The chi-squared statistic compares the correlation matrix generated by the model to the actual correlation matrix. Smaller values indicate less deviation, therefore a good match is indicated by a non-significant *p*-value (suggesting that the model's correlation matrix is "not different" from the observed matrix). The Root Mean Square Error of Approximation (RMSEA) and Standardised Root Mean Squared Residual (SRMR) are similar to the chi-squared statistic, they also measure how well a model's correlation matrix matches the actual correlation matrix. Again, smaller values indicate a better fit, typically using SRMR < .08 as a cut-off. The Comparative Fit Index (CFI) and the Adjusted Goodness-of-Fit Index (AGFI) both quantify the proportion of variance explained by the model, with higher values indicating a better fit. CFI and AGFI values greater than 0.9 are typically considered a good

Table 1. Predictors of score on the SCS1-short programming test and course grade at the end of the semester.

		SCS1-Short					Course grade				
	Type of variable	β Estimate	Std Error	*t*-value	*p*-value	BF_{10}	β Estimate	Std Error	*t*-value	*p*-value	BF_{10}
First experience	Control	1.072	.246	4.351	<.001***	1249.965^{+}	10.997	2.138	5.144	<.001***	30335.880^{+}
Version SCS1-Short	Control	0.598	.247	2.422	.016*	3.802^{+}	-	-	-	-	-
Pattern recognition	Predictor	0.042	.134	.327	.744	0.262^{-}	1.532	1.128	1.358	.176	0.564
Algebra	Predictor	0.300	.135	2.228	.027*	2.513	1.762	1.175	1.500	.135	0.682
Logical reasoning	Predictor	0.408	.134	3.040	.003**	17.628^{+}	3.106	1.181	2.630	.009**	5.937^{+}
Grammar learning	Predictor	0.122	.149	0.816	.415	0.341	1.750	1.289	1.358	.176	0.564
Vocabulary delayed recall	Predictor	0.370	.135	2.737	.007**	7.981^{+}	2.236	1.173	1.907	.058	1.304

Note: Results for the multiple regression models with standardised scores on the cognitive tests at the start of the semester as predictors, and raw scores on the SCS1-Short and the course grades at the end of the semester as outcome measures. Whether the course was their first programming experience was added as a control variable for both regression models. For the regression model with SCS1-Short as the dependent variable the version of the SCS1-Short was added at a control variable as well. *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001. $^{+}$indicates $BF_{10} > 3$; – indicates $BF_{10} < .333$.

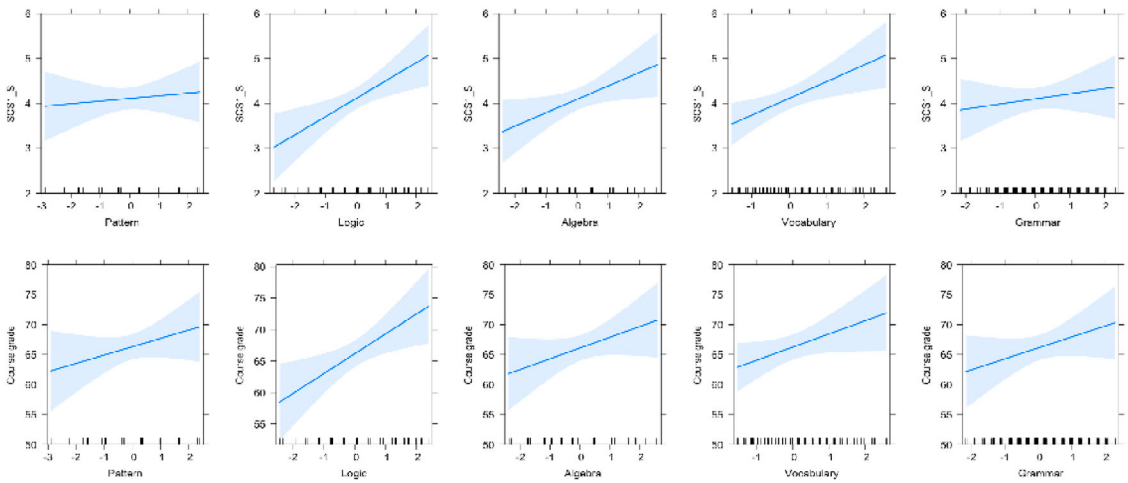

Figure 5. Partial slopes for each standardised cognitive skill predicting scores on the SCS1-short and course grades. [To view this figure in color, please see the online version of this journal.] Note: Top row: Partial slopes for each standardised cognitive skill predicting raw scores on the SCS1-Short. Bottom row: Partial slopes for each standardised cognitive skill predicting raw course grades. For the models relating to each plot the other predictors and control variables are held constant. The shaded area represents a pointwise confidence band based on standard errors. The lines on the horizontal axes show the exact scores on the cognitive tests for each individual participant.

fit. For the current models, we found good fits for both the model with SCS1-Short ($X^2(7, n = 245) = 6.598$, $p = .472$, RMSEA = 0.000, SRMR = .028, CFI = 1.000 and AGFI = .972) and the model with course grades ($X^2(7, n = 243) = 4.095$, $p = .769$; SRMR = .023; RMSEA = 0.000; CFI = 1.000 and AGFI = .983). Figures 3 and 4 depict the full models with fitted parameters.

The latent variable for algorithmic/mathematical thinking significantly predicted scores on the SCS1-Short programming test ($\beta = .493$, $SE = .241$, $z = 2.044$, $p = .041$), and course grades (® = .542, SE = .269, $z = 2.018$, $p = .044$). However, the latent variable for language did not predict scores on either the SCS1-Short programming test ($\beta = .057$, $SE = .223$, $z = .257$, $p = .797$) nor the course grades ($\beta = .000$, $SE = .245$, $z = .001$, $p = .999$). There was also a strong correlation between the latent variables for language and algorithmic/mathematical thinking, both for the model with the SCS1-Short ($\beta = .736$, $SE = .108$, $z = 6.850$, $p < .001$) and for the model with the course grades ($\beta = .738$, $SE = .113$, $z = 6.507$, $p < .001$). Full tested models with estimates, including covariance can be seen in Figures 6 and 7.

4. Discussion

The aim of this study was to examine which cognitive skills predict programming performance following a modern-day programming course. Specifically, we tested whether five cognitive skills (logical reasoning, algebra, pattern recognition, grammar learning and vocabulary learning) predicted course-related programming performance, and generalised programming performance at the end of a 12-week first year university course. We also examined whether these cognitive skills could be grouped into an algorithmic/mathematical factor (comprising pattern recognition, algebra and logical reasoning) and a language factor (vocabulary learning and grammar learning), and whether these factors predicted programming success.

Table 2. Correlations between cognitive tests.

	Pattern recognition	Algebra	Logical reasoning	Grammar learning	Vocabulary learning
Pattern recognition	1	0.270*	0.185*	0.197*	0.082
Algebra	0.270*	1	0.254*	0.287*	0.173*
Logical reasoning	0.185*	0.254*	1	0.267*	0.219*
Grammar learning	0.197*	0.287*	0.267*	1	0.369**
Vocabulary learning	0.082	0.173*	0.219*	0.369**	1

Note: Correlations between cognitive tests only including complete observations. *indicates a small correlation, and ** indicates a moderate correlation.

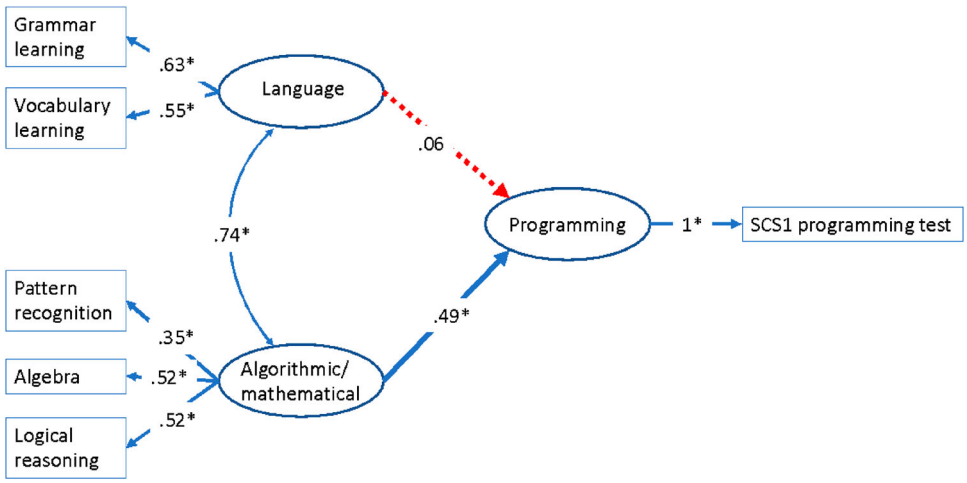

Figure 6. Fitted structural model representing the relationship between language and algorithmic/mathematical thinking and generalised programming skill as measured by the SCS1-short. [To view this figure in color, please see the online version of this journal.] Note: Blue arrows and * indicate significant estimates at $p < .05$. Red arrows indicate $p > .05$. Estimates are written along their corresponding model lines.

There are three findings that need to be discussed further. First, we found a discrepancy between predictors of course-related and generalised programming performance. Only logical reasoning predicted course-related programming performance, whereas logical reasoning, vocabulary learning and, according to frequentist statistics, algebra, each predicted generalised programming performance on a test using a pseudocode programming language (SCS1-Short). In both cases, the skills were predictive even after controlling for programming experience prior to the start of the course. Second, based on the results of earlier studies (Pena & Tirre, 1992; Shute, 1991; Webb, 1985), we expected that all tested cognitive skills would be predictive of programming performance. However, we found no predictive value for pattern recognition and grammar learning. Finally, we found that the cognitive skills could be grouped into an algorithmic/mathematical factor consisting of logical reasoning, algebra and pattern recognition, and a language factor consisting of

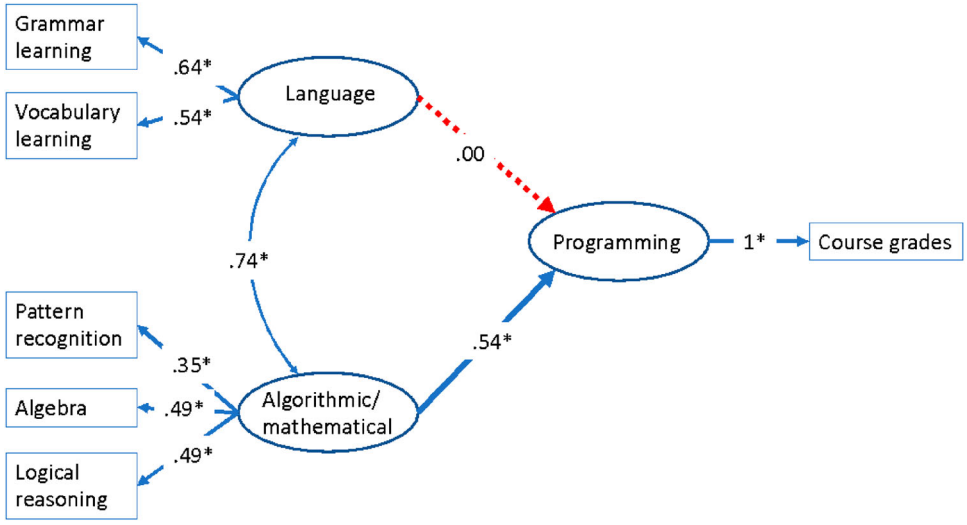

Figure 7. Fitted structural model representing the relationship between language and algorithmic/mathematical thinking and course-related programming skill as measured by the course grades. [To view this figure in color, please see the online version of this journal.]

Note: Blue arrows and * indicate significant estimates at $p < .05$. Red arrows indicate $p > .05$. Estimates are written along their corresponding model lines.

grammar learning and vocabulary learning. Given the language-like nature of the programming languages used in the course and in the SCS1-Short, we expected that both factors (algorithmic/mathematical and language) would be predictive of programming skill. The algorithmic/mathematical factor was indeed predictive of both generalised programming performance and course-related programming performance, but the language factor was not predictive of either measure of programming performance.

4.1. Generalised versus course-related programming performance

What explanation could there be for the finding that generalised programming performance was predicted by algebra and vocabulary learning while course-related programming performance was not? One possibility may lie in the difference in format between the two types of assessments: the independent programming test had a strict time limit, required participants to learn to apply a pseudocode programming language on the spot, and did not allow participants to use a calculator or the Internet, while for assessments that underpinned the course grade, the students were allowed more time, several attempts, and the use of external resources. It is thus possible that the stricter format of the independent programming test meant that participants needed to rely more on their memory and learning skills (related to the vocabulary learning test) and their mathematical skills (related to the algebra test) when completing this test compared to the course assessments.

It is also possible that the difference in results relates to the fact that the course assessments allowed for multiple attempts. It may be the case, therefore, that the scores were more dependent on student persistence and less on pure programming skill. This would result in cognitive skills having less predictive value for this kind of assessment, which matches our findings. It also explains the contrast with previous studies that found several cognitive skills to be predictive of course exams (Pena & Tirre, 1992; Shute, 1991; Webb, 1985). The programming assessments used by these older studies were more similar to our independent programming test, than to our course assessment. They had time limits, allowed only one attempt, and sometimes restricted use of outside sources. Therefore, it is possible that the cognitive skills we examined have more predictive value for more structured and restricted test formats. Future research can further clarify this by administering a wider variety of tests with various structures and levels of restrictions.

4.2. Lack of predictive ability for pattern recognition

The results of the role of pattern recognition in programming are unclear. In the current study, we did not find pattern recognition to be predictive of programming skill. Pena and Tirre (1992), however, whilst using a similar test of programming, did find a predictive role for pattern recognition. Further muddying the waters, Webb (1985) found predictive ability, but only for the syntax component of their programming test—a dimension that is not included in either our or Pena and Tirre's (1992) tests. Thus, it remains unclear which dimensions of programming skill, if any, rely on pattern recognition.

We speculate that pattern recognition may be specifically relevant for programming when the assessment demands a high level of programming efficiency. For example, when a piece of code has to be written in a limited number of lines the programmer needs to make efficient use of control structures, such as loops or functions. To use loops or functions effectively, the programmer has to recognise patterns in the required steps that can be captured in these structures, which might be what can be measured with pattern recognition tasks. Our independent programming test did not require participants to write pieces of code, so it did not demand efficient code design, and in the course assessments students were evaluated on the output/results of the code rather than on its efficiency. For example, they were not penalised for using repetitive code instead of a loop or a clever function. We suggest that the current pattern recognition test might be predictive of programming performance on assessments where programming efficiency would be evaluated more strictly. Future studies are required to test this hypothesis.

4.3. Lack of predictive ability for language skills

Despite the modern trend for programming languages to more closely resemble natural languages, we did not find that language skills as

a cluster, or grammar learning skills specifically, predicted programming performance. Contradicting our findings, Prat et al. (2020) found that language aptitude predicted 17% of variance in programming skill. However, Prat et al. (2020) measured language aptitude with the Modern Language Aptitude Test, which includes tasks that rely heavily on working memory, for example, memorising lists of auditory numbers or sound-symbol relationships. Therefore, it is possible that it is the memory component of language skills that plays a role when learning to programme, and hence language skills are only found to be predictive if there is a strong memory component to the language test. Looking at the older studies, only Shute (1991) tested a language-based memory component with a word span working memory test and found that this task was predictive of programming skill. This is also in line with our findings, where vocabulary learning, which relies primarily on memory, was found to be a predictor of programming ability, while grammar learning, where the examples stayed available throughout the test, was not. It is possible that the memorisation aspect of language is most important when learning to programme, and that this aspect was underrepresented in the language factor of the current study.

We offer two further possible explanations for the fact that grammar learning was not predictive of programming outcomes. Firstly, it is possible that the way in which programming courses are currently taught and assessed does not allow for grammar skills to play a role. Because of the written nature of programming languages, the format of assignments and tests usually allows programmers to take time to look up rules or copy structures from examples. This diminishes the importance of memorising the syntax of a specific programming language. We see this reflected in the way programming is currently taught. In programming courses, the focus is usually on problem solving and algorithmic thinking. Specific programming languages are only used as a tool to express these other skills and are rarely explicitly focussed on (Hermans & Aldewereld, 2017). This means that in terms of the PGK-hierarchy, education and assessment focus primarily on the first two levels: the problem level and the object level. This is also the case in our measures of programming, where students were allowed to use outside sources or a pseudocode guide to copy syntax. Neither measure required students to explicitly evaluate the syntax rules of the programming language. Researchers such as Hermans and Aldewereld (2017) have argued that explicit teaching of programming languages should be more central to programming education. If this were to be implemented, it is possible that grammar learning specifically, and language skills as a whole would become stronger predictors of programming skill.

Finally, it is also possible that grammar learning in a natural language is too distant from syntax learning in a programming language. Further research into the linguistic properties of programming languages is necessary to determine whether there are fundamental differences between programming syntax and natural language grammar. This could be investigated in future studies by comparing both behavioural and neurological responses to both programming languages and natural languages.

4.4. Limitations

In the current study, we used Structural Equation Modelling to determine to what extent language and algorithmic/mathematical skills predict programming success. One limitation of this method is that the subdivision of predictor measures was selected by the researchers. Therefore, we cannot exclude the possibility that the tests could be grouped in different ways that would fit the data more accurately. For example, we saw that logical reasoning correlated with grammar learning and vocabulary learning similarly to the extent to which it correlated with algebra and pattern recognition. It is possible that the tests could have been grouped in a way that better explained the data if they were split along declarative/procedural or crystalised/fluid axes.[2] These would be interesting avenues for future studies to explore.

Similarly, the SEM analysis cannot show to what extent language skills might predict programming skills indirectly. The SEM models show a possible indirect path to programming skill for language skills via algorithmic/mathematical skills. It is possible that language skills facilitate performance on mathematical and algorithmic tasks and thereby influence programming performance. To determine whether this is the case, future studies could be designed to specifically study these interactions. These further investigations may then shed more light on the

[2]We would like to thank our reviewer Chantel Prat for this insight.

connection between language skills and the general programming process. In the current study, we speculated that language skills may be most related to the third level of the PGK hierarchy, as this is the level where a programming language is implemented. However, if language skills facilitate algorithmic skills, they may well be involved in earlier stages of the programming process.

When interpreting the results of the current study caution is also advised with regard to the generalisability. The current study was conducted within a specific population of university students who had voluntarily chosen to take this programming course and had explicitly consented to their data being used in the current study. The sample of the current study was predominantly male, and relatively young. This sample is likely to be representative of most university courses, however, further research is needed to determine whether the results found with this population can be generalised to the broader population of individuals who may learn programming. Similarly, while the programming measures and teaching formats in the current study may be representative of many university face-to-face programming courses, they may differ from other contexts and formats, such as the online learning environment as used by Prat et al. (2020), or programming in a professional context. Results may also differ in contexts where different programming languages are used. Additionally, the course assessment in the current study was not a standardised and validated measure, and should therefore be interpreted with caution when comparing results to those of other studies.

5. Conclusions

Overall, the current study demonstrates that logical reasoning is a reliable predictor of generalised and course-related programming performance, and that algebra and vocabulary learning skills are successful predictors of generalised programming performance at the end of a semester-long undergraduate programming course. Our results suggest that algorithmic/mathematical skills are most relevant when predicting generalised programming success, but also show a role for memory-related language skills. Although we cannot directly compare the results to previous studies which did not test generalised programming performance, we do see converging evidence for the skills that were found to be predictive in the studies of the 1980s and 1990s (Pena & Tirre, 1992; Shute, 1991; Webb, 1985). This suggests that these skills are still relevant in the current context.

Though modern programming languages resemble natural languages, this study did not find strong evidence for language skills as predictors of programming success. We suggest that these results differ from those of Prat et al. (2020) because the course and assessments in our study mostly focused on the problem and object levels of the PGK-hierarchy and less on the third level where specific programming languages are used. Future research will be necessary to understand the role of language skills in programming and to further investigate to what extent cognitive skills that predict programming performance depend on the format and content of the programming assessment (e.g. time pressure, use of outside sources, number of attempts and focus on efficiency).

Acknowledgements

We would like to thank the Department of Computing at Macquarie University for allowing us to conduct this research in their programming courses and all the students who completed the tests. In particular, we would like to thank the course coordinators, tutors and computing students for their efforts and support and the computing interns who participated in pilots, helped with pilot data analysis and with digitisation of Qualtrics tests. We would also like to thank IBM for allowing us to use their programming aptitude tests. Finally, we would like to thank Miranda Parker and Mark Guzdial for allowing us to use the SCS1 for this research. Irene Graafsma conducted this as part of a Joint PhD—the International Doctorate in Experimental Approaches to Language and Brain (IDEALAB) awarded by Macquarie University (Australia), Groningen University (The Netherlands), Potsdam University (Germany) and Newcastle University (United Kingdom). Funded by a Sandwich project "Natural versus programming language: Similarities and differences in underlying cognitive processes", awarded to Tops, Bastiaanse, Nickels, and Marinus, and an International Macquarie University Research Excellence Scholarship (iMQRES).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Data availability statement

The data that support the findings of this study are openly available in the Cognition of Coding project in OSF storage at https://osf.io/8nax4/?view_only=eb0341df544b435792e436451929e2cd.

ORCID

Irene L. Graafsma http://orcid.org/0000-0001-7071-1498
Serje Robidoux http://orcid.org/0000-0002-4581-3297
Lyndsey Nickels http://orcid.org/0000-0002-0311-3524
Matthew Roberts http://orcid.org/0000-0002-2553-6157
Vince Polito http://orcid.org/0000-0003-3242-9074
Judy D. Zhu http://orcid.org/0000-0003-0958-6047
Eva Marinus http://orcid.org/0000-0002-3628-0695

References

Armoni, M. (2013). On teaching abstraction in CS to novices. *Journal of Computers in Mathematics and Science Teaching*, *32*(3), 265–284. https://www.learntechlib.org/primary/p/41271/.

Baron-Cohen, S., Wheelwright, S., Skinner, R., Martin, J., & Clubley, E. (2001). The autism-spectrum quotient (AQ): Evidence from Asperger syndrome/high-functioning autism, males and females, scientists and mathematicians. *Journal of Autism and Developmental Disorders*, *31*(1), 5–17. https://doi.org/10.1023/A:1005653411471

Bennedsen, J., & Caspersen, M. E. (2005, October). An investigation of potential success factors for an introductory model-driven programming course. *Proceedings of the First International Workshop on Computing Education Research* (pp. 155–163).

European Schoolnet. (2015, November 30). Computing our future. Computer programming and coding. Priorities, school curricula and initiatives across Europe. http://www.eun.org/c/document_library/get_file?uuid=3596b121-941c-4296-a760-0f4e4795d6fa&groupId=43887.

Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The language of programming: A cognitive perspective. *Trends in Cognitive Sciences*, *23*(7), 525–528. https://doi.org/10.1016/j.tics.2019.04.010

Floyd, B., Santander, T., & Weimer, W. (2017). Decoding the representation of code in the brain: An fMRI study of code review and expertise. *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (pp. 175–186). https://doi.org/10.1109/ICSE.2017.24.

Graafsma, I. L., Robidoux, S., Nickels, L., Roberts, M., & Marinus, E. (2020). *Validating two short versions of the second computer science 1 programming ability test*. Open Science Framework. https://osf.io/exkbm/?view_only=719e21fee326407cad1b39fbfb866d80.

Guzdial, M. (2003). A media computation course for non-majors. *Proceedings of the 8th Annual Conference on Innovation and Technology in Computer Science Education* (pp. 104–108). https://doi.org/10.1145/961511.961542.

Guzdial, M. (2016). Synthesis lectures on human-centered informatics. *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1–165. https://doi.org/10.1007/978-3-031-02216-6

Guzdial, M., & du Boulay, B. (2019). The history of computing education research. In S.A. Fischer & A.V. Robins (Eds.), *The Cambridge handbook of computing education research* (pp. 11–39). Cambridge University Press.

Hackerrank. (2018, January 23). *Developer skills report*. http://research.hackerrank.com/developer-skills/2018/.

Handley, S. J., Capon, A., Copp, C., & Harper, C. (2002). Conditional reasoning and the tower of Hanoi: The role of spatial and verbal working memory. *British Journal of Psychology*, *93*(4), 501–518. https://doi.org/10.1348/000712602761381376

Hermans, F., & Aldewereld, M. (2017). Programming is writing is programming. *Science and Engineering of Programming*, *1*, 1–8. https://doi.org/10.1145/3079368.3079413.

IBM. (1968). *Aptitude test for programmer personnel*.

Jacob, S. R., & Warschauer, M. (2018). Computational thinking and literacy. *Journal of Computer Science Integration*, *1*(1), https://doi.org/10.26716/jcsi.2018.01.1.1

JASP Team. (2019). JASP (Version 0.10.2) [Computer software].

Kelleher, C., & Pausch, R. (2003). *Lowering the barriers to programming*. *ACM Computing*.

Kline, R. (2005). *Principles and practices of structural equation modeling* (2nd ed.). Guilford Press.

Marasco, E. A., Moshirpour, M., & Moussavi, M. (2017). Flipping the foundation: A multi-year flipped classroom study for a large-scale introductory programming course [Paper presentation]. *ASEE Annual Conference & Exposition, Columbus, Ohio, USA*. https://peer.asee.org/28372.

O'Regan, G. (2012). History of programming languages. In G. O'Regan (Ed.), *A brief history of computing* (pp. 121–144). Springer. https://doi.org/10.1007/978-1-4471-2359-0_9.

Pandža, N. B. (2016). Computer programming as a second language. In N.B. Pandža (Ed.), *Advances in human factors in cybersecurity* (pp. 439–445). Springer. https://doi.org/10.1007/978-3-319-41932-9_36.

Parker, M. C., Guzdial, M., & Engleman, S. (2016). Replication, validation, and use of a language independent CS1 knowledge assessment. *Proceedings of the 2016 ACM Conference on International Computing Education Research* (pp. 93–101). https://doi.org/10.1145/2960310.2960316.

Paulson, L. D. (2007). Developers shift to dynamic programming languages. *Computer*, *40*(2), 12–15. https://doi.org/10.1109/MC.2007.53

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, *2*(2), 137–168. https://doi.org/10.1016/0732-118X(84)90018-7

Pena, C. M., & Tirre, W. C. (1992). Cognitive factors involved in the first stage of programming skill acquisition. *Learning and Individual Differences*, *4*(4), 311–334. https://doi.org/10.1016/1041-6080(92)90017-9

Perrenet, J., Groote, J. F., & Kaasenbrood, E. (2005). Exploring students' understanding of the concept of algorithm: levels of abstraction. *ACM SIGCSE Bulletin*, *37*(3), 64–68.

Pimsleur, P., Reed, D. J., & Stansfield, C. W. (2004). *Pimsleur language aptitude battery: Manual 2004 edition*. Second Language Testing.

Polito, V., Barnier, A. J., & Woody, E. Z. (2013). Developing the sense of agency rating scale (SOARS): An empirical measure of agency disruption in hypnosis.

Consciousness and Cognition, *22*(3), 684–696. https://doi.org/10.1016/j.concog.2013.04.003

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C.-H. (2020). Re-epithelialization and immune cell behaviour in an ex vivo human skin model. *Scientific Reports*, *10*(1), 1–10. https://doi.org/10.1038/s41598-019-56847-4

Quille, K., & Bergin, S. (2018). Programming: Predicting student success early in CS1. A re-validation and replication study. In *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education* (pp. 15–20). https://doi.org/10.1145/3197091.3197101.

Quille, K., & Bergin, S. (2019). CS1: how will they do? How can we help? A decade of research and practice. *Computer Science Education*, *29*(2-3), 254–282. https://doi.org/10.1080/08993408.2019.1612679

R Core Team. (2019). *R: A language and environment for statistical computing*. *R Foundation for Statistical Computing*. URL https://www.R-project.org/.

Rogers, V., Meara, P., Barnett-Legh, T., Curry, C., & Davie, E. (2017). Examining the LLAMA aptitude tests. *Journal of the European Second Language Association*, *1*(1), 49–60. https://doi.org/10.22599/jesla.24.

Romanova, A. (2015). *Word class effects on representation and processing in non-brain damaged speakers and people with aphasia*. [Doctoral dissertation, Macquarie University]. Macquarie University Library. https://www.researchonline.mq.edu.au/vital/access/services/Download/mq:44500/SOURCE1?view=true.

Rouder, J. N., Speckman, P. L., Sun, D., Morey, R. D., & Iverson, G. (2009). Bayesian t tests for accepting and rejecting the null hypothesis. *Psychonomic Bulletin & Review*, *16*(2), 225–237. https://doi.org/10.3758/PBR.16.2.225

Rushkoff. (2012). Code literacy: A 21st-century requirement. *Edutpia*. https://www.edutopia.org/blog/code-literacy-21st-century-requirement-douglas-rushkoff.

Seegerer, S., Michaeli, T., & Romeike, R. (2019). Informatik für alle-Eine Analyse von Argumenten und Argumentationsschemata für das Schulfach Informatik [Computer science for everyone – an analysis of arguments and argumentation schemes for the school subject computer science.]. *INFORMATIK 2019: 50 Jahre Gesellschaft für Informatik–Informatik für Gesellschaft*. https://doi.org/10.18420/inf2019_77.

Shute, V. J. (1991). Who is likely to acquire programming skills? *Journal of Educational Computing Research*, *7*(1), 1–24. https://doi.org/10.2190/VQJD-T1YD-5WVB-RYPJ

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., & Brechmann, N. (2014). Understanding understanding source code with functional magnetic resonance imaging. *In Proceedings of the 36th International Conference on Software Engineering* (pp. 378–389). https://doi.org/10.1145/2568225.2568252.

Skehan, P. (1991). Individual differences in second language learning. *Studies in Second Language Acquisition*, *13*(2), 275–298. https://doi.org/10.1017/S0272263100009979

Vee, A. (2013). Understanding computer programming as a literacy. *Literacy in Composition Studies*, *1*(2), 42–64. https://doi.org/10.21623/1.1.2.4

Vogel, S., Hoadley, C., Ascenzi-Moreno, L., & Menken, K. (2019). The role of translanguaging in computational literacies. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 1164–1170). https://doi.org/10.1145/3287324.3287368.

Webb, N. M. (1985). Cognitive requirements of learning computer programming in group and individual settings. *AEDS Journal*, *18*(3), 183–194. https://doi.org/10.1080/00011037.1985.11008398

Appendix

Items of SCS1-S versions 1 and 2.

SCS1-S version 1				SCS1-S version 2			
Question number SCS1-S1	Original question from SCS1	Topic	Type of question	Question number SCS1-S2	Original question from SCS1	Topic	Type of question
1	1	for	definitional	1	3	while	definitional
2	2	logical operator	tracing	2	4	arrays	definitional
3	5	function return values	tracing	3	7	while	code completion
4	6	if	definitional	4	8	for	Tracing
5	9	while	tracing	5	10	logical operator	definitional
6	12	basics	definitional	6	11	function return values	definitional
7	13	for	code completion	7	16	function parameters	code completion
8	14	recursion	definitional	8	19	if	tracing
9	17	arrays	code completion	9	20	function parameters	Definitional
10	18	recursion	code completion	10	21	if	code completion
11	22	arrays	tracing	11	23	basics	tracing
12	25	basics	code completion	12	24	recursion	Tracing
13	27	function parameters	tracing	13	26	logical operator	code completion