Computer programming skills: A cognitive perspective

Graafsma, Irene

DOI:
10.33612/diss.168003240

Chapter 3

Using cognitive skills to predict programming performance following an introductory computing course[1]

[1] This chapter has been submitted as an article for publication.

3.1 INTRODUCTION

Programming education has gained major importance and popularity worldwide. All over the world initiatives have been taken to teach people how to program, focusing on both children and adults (European Schoolnet, 2015). In response to the growing interest in learning to program, 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, 2015). 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 program (e.g., Pea & Kurland, 1984). The other stream researches which cognitive skills are important when learning to computer program. The current study aims to contribute to this second stream.

The cognitive skills underpinning learning programming were 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 interface-based software and computers removed the need for programming for most users (HackerRank, 2018). Over the last decade, the number of people required to learn to program has been increasing again. This is partly because the increase in use of technology demands more software engineers (Seegerer et al., 2019). In addition, in many jobs there has been a shift from using to creating, with a range of professions expected to program their own content, such as designing their own websites or developing data analysis programs (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, meaning 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 program. Last, 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 program today may be different from the skills found in the early studies of over twenty years ago. The current study will therefore examine which cognitive skills are important in the modern-day context. Before describing the current situation, we first give a short review of the results of the earlier studies that focused on cognitive skills in learning to program.

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 recognizing 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 to 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.

These findings give an idea of the cognitive skills that may be involved when first learning to program. 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. Last, 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 is learnt. This enabled us to determine whether cognitive skills at the start of the course predicted programming performance at the end. Second, by testing students over a 12-week semester, we could test how cognitive skills related to long term learning in a typical university course. Finally, by assessing programming skill with two measures to provide both a measure of course-related programming performance (from scores on the course assessments) and generalised programming performance (from scores on an independent programming test at the end of the course). The use of the independent programming test also allows for easier comparison with studies that use that same test.

The cognitive skills that we tested were selected 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 think may be important when learning to program. These tests include elements assessing for pattern recognition, algebra and logical reasoning, with problem solving as a component of the different algebra tests. Last, we looked at a recent theoretical framework of the programming process: the PGK-hierarchy, named after Perrenet, Groote and Kaasenbrood, who first defined it (Perrenet et al., 2005) and further described by Armoni (2013).

The PGK-hierarchy postulates that writing a computer program 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 program an animation. At the highest level, the problem level, the programmer considers the solution the problem demands and considers the 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. At this level, s/he does not yet look at the solution of the problem or make specific plans. The second level is the object level. At this level, an algorithm (a plan detailing specific steps that have to be executed by the program) is developed to solve the problem. Here the programmer specifies which functions or programming objects should be created for the animation and in which order. However, the plan is not yet associated with a specific programming language. The third level is the program level. This is where an algorithm is written in a specific programming language. In our example, the programmer will now look up the specific functions in the programming language that s/he wants to use and will write out the code with the correct terms and syntax. The lowest level is the execution level, which is the interpretation and execution of the algorithm by the computer; thus, it does not relate to human cognitive skills and we will not consider it further.

The first two levels of the PGK-hierarchy seem to rely on algorithmic thinking (deriving a solution by defining the specific steps necessary to solve a problem) and mathematical skills. We therefore hypothesise 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 program 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; Hermans & Aldewereld, 2017). 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) argue that throughout its existence, programming has been regarded as related to natural languages, with some educational systems allowing students to take 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. A recent paper by Hermans and Aldewereld (2017) argues against this shift away from natural languages and emphasises again the similarities between

programming and natural language writing. They maintain that both processes rely on high level planning and problem solving at the start, and on specific structure and style rules at the later stages of implementation. Another review by Pandža (2016) argues 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 (Fedorenko et al., 2019; Paulson, 2007). As this may lead to transfer between natural language and programming skill, this leads to the possibility that natural languages could be strong predictors of the ease of acquisition of programming languages.

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), memory and learning (to make and recall associations between words and phrases in L1 and L2) (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 hypothesise that programming may rely on grammatical sensitivity and inductive language learning ability for syntax acquisition, and memory and vocabulary learning for memorization of language-specific terms and functions.

In sum, the current study examined which cognitive skills predict programming success in current 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 model is correct, we would also expect the first three skills to be related to each other, while the language skills may be relatively independent. Therefore, we tested whether our selected cognitive skills would cluster into an algorithmic/mathematics component (pattern recognition, algebra and logical reasoning) and a language component (vocabulary learning and grammar learning). We predicted that both components would predict final programming performance and examined the extent to which each component contributes to the prediction of programming success.

3.2 METHODS

3.2.1 Participants

All 838 students in an "Introductory Programming" course (COMP115) at an Australian 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. 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)", or, when probed, if they indicated that they did not seriously attempt the tests, took notes when not allowed, or worked together. After removing these participants, 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 participants were also 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 program. The study received ethical approval from the Macquarie University Human Research Ethics Committee (Reference number: 5201800224).

3.2.2 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 their randomly assigned student number, 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 Procedure for details.

Primary outcome measures

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

Second Computer Science 1 Short (SCS1-S). This test is based on the Second Computer Science 1 (SCS1; Parker et al., 2016). We used a computer-based version with test items split into two parallel versions, as described in Chapter 2. The SCS1 uses an artificial programming language specifically developed for this test by its developers. 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 minutes to complete as many questions as possible. They could answer the questions in any order by scrolling back and forth through the questions on screen.

Course grades. These were the student's grades on the main course assessment of their programming course. 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 program design & problem solving. Students were given three attempts for each module assessment. They were free to complete these attempts during various provided exam sessions, which could be either throughout the course, or in the exam session two weeks after our testing session with the SCS1-S. For each module the student's highest score counted towards their final grade. 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 =eb0341df544b435792e 436451929e2cd). 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 skill.

Predictor measures

We included measures of cognitive skills[1] that we hypothesised to predict programming skill at the end of the course. All of these tests were presented digitally using the online survey tool Qualtrics. For each test, participants could scroll back and forth through the questions to answer them in any order. Once the allotted time for a test was reached, or when a test was submitted, the system moved on to the next test.

Logical reasoning. This was a test of logical reasoning with syllogisms (Handley et al., 2002). Each item consisted of a syllogism 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, where Version 1 used the exact items from Handley et al. (2002) while 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)[2], 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 ,

[1] Full tests, descriptive results for the tests, and correlations between tests can be found on the OSF: https://osf.io/8nax4/

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

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)[2]. The original test used items that described a mathematical word problem, with the final answer amongst five answer options. For the current study, the test was adapted to have a more abstract format that did not rely on arithmetic by instead providing different formulas as the answer options. To do this we adapted the questions so that some numbers were changed into variables, and we specifically developed four formulae as answer options for each question. We split this test into two parallel versions of 10 questions each, by alternating even and uneven question numbers in each version in the same way as we did for the pattern recognition test. 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)

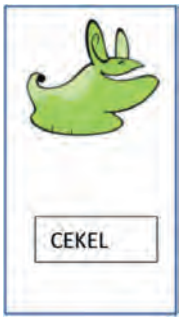
c) T1-((6.5-1)(T1-T2)/(4-1))

d) T1-((4-1)(T1-T2)/(6.5-1))

Vocabulary learning. We developed this test based on the vocabulary learning subtest of the Language Learning Aptitude for MA students (LLAMA; 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 3.1) specifically developed for the current study. All creatures with

their names were simultaneously presented on the screen, and participants were given 2.5 minutes to memorise the pairs without being allowed to take any notes. They then underwent a first testing session immediately after the learning phase, and a delayed recall test 30 minutes later. In the testing phases, all of the creatures' pictures and names were displayed on the screen and participants were given 3.5 minutes 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 both versions.

Figure 3.1. Example of a learning item on the vocabulary learning test.

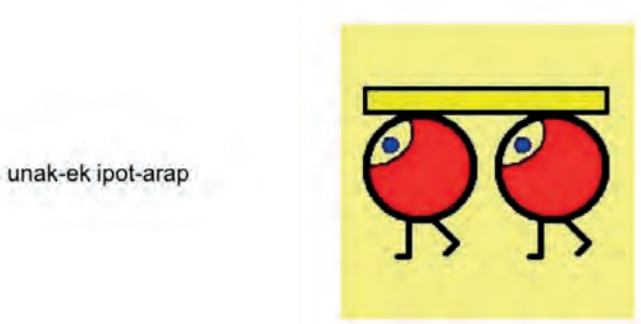


Note: This figure shows one of 20 learning items on the vocabulary learning test. Participants were asked to memorise the names of the creatures. In the immediate and delayed recall stages they were asked to drag and drop the correct names under the corresponding pictures.

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). We took the material for the items from the LLAMA test, which provided participants with a series of example pictures with a matching description for each picture written in the artificial language with simple grammatical rules. One such example sentence is "unak-ek ipot-arap", which describes two red circle creatures walking underneath a rectangle as shown in Figure 3.2. The examples were then followed by questions where participants were asked to select the grammatically correct descriptions of new pictures. We adapted the structure of the test in order to make it less reliant on

memorization and more reliant on grammatical deduction. We did this by following a similar structure as the PLAB, where examples remain accessible whilst answering questions. We structured the test in such a way that participants could scroll back and forth through the 20 test items, with nine examples spread across the test in three blocks of three. To make the questions a bit more difficult now that they relied less on memory, we let participants choose from amongst four answer options instead of just two answer options as used in the original LLAMA (one correct, three foils). Students were given eight minutes to complete the test.

Figure 3.2. Example of a learning item on the grammar learning test.



Note: This figure shows a learning example from the grammar learning test. The sentence on the left described the image on the right. Participants were asked to use examples like this one to answer questions where they had to select the correct sentence describing a new picture from four answer alternatives.

Demographics. Two questionnaires were administered. In the testing session at the beginning of the course we collected information about the participants' age, gender, major, knowledge of programming languages, previous programming experience and level of English. In the testing session at the end of the semester we asked students whether they had completed the tests according to the rules (e.g., no calculators, no help from others and no note taking when not allowed).

Tests not used for the current study

As part of a larger study two more measures were administered that were not included in the current results: 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).

3.2.3 Procedure

The testing sessions took place during the first and last tutorials of the programming course and were led by the regular course tutors. Participants were informed that the aim of the study was to see which skills are important in learning computer programming. They were not given any specific information about the tests or expectations of the study.

At the start of the tutorial 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. In order 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 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 seconds before the student could move on to the next page. In the first testing session 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. The order of the second testing session was: SCS1-S and demographic

questionnaire. The first testing session took approximately one hour, and the second session took approximately 30 minutes 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 usual coursework for COMP115 with no additional tasks related to this study. Three percent of participants included in the study completed the tests at home because they could not attend the tutorials.

3.3 RESULTS

3.3.1 Analysis

We considered results significant at *p*-values below .05. We used both frequentist and Bayesian statistics for the pre-processing *t*-tests and for the regression models. 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).

3.3.2 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 no 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 wanted to compute *z*-scores for this test as well. However, in a separate validation study (Chapter 2) we tested the quality of each SCS1 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. In order to avoid confounding the standardised *z*-scores with course grades while still eliminating any potential influence of SCS1 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 version difference without assuming that the samples were equal.

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

3.3.3 Regression models

We used a multiple regression model to examine which cognitive skills at the start of the semester predicted final 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 seems the most relevant form of learning for learning programming over several months. We also added two control variables: whether or not this was participants' first programming experience, and version of the SCS1 as control variables. The results are shown in Table 3.1 and Figure 3.3. 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 and grammar learning skills as predictors were inconclusive.

We ran the same multiple regression model with course grades as the dependent variable. The results are shown in Table 3.1 and Figure 3.3. We found that only logical reasoning was a significant predictor of performance on the course grade. The Bayes factors indicated evidence for logical reasoning as a predictor and were inconclusive for the other cognitive skills as predictors.

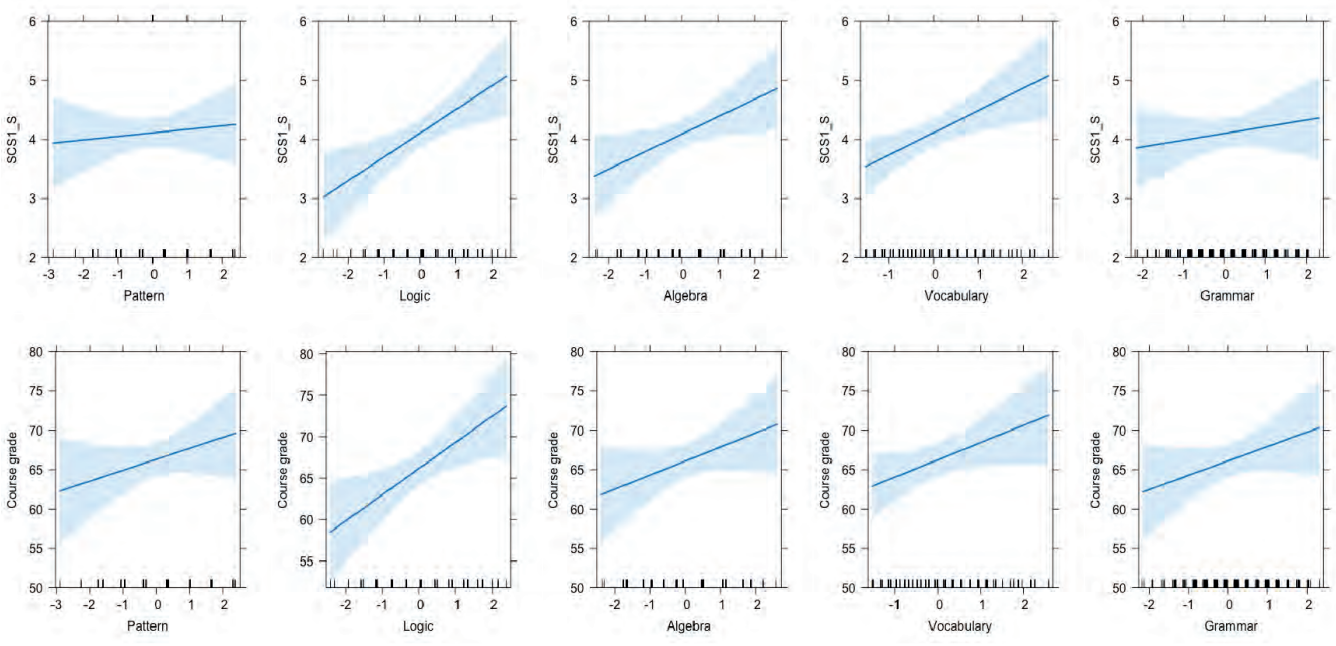
Table 3.1. Predictors of score on the SCS1-S programming test and course grade at the end of the semester.

		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	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-S and the course grades at the end of the semester as outcome measures [3]. Whether the course was their first programming experience was added as a control variable for both regression models. For the regression model with SCS1-S as the dependent variable the version of the SCS1-S 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 $BF_{10} < .333$.

[3] The same predictors (logical reasoning, algebra and vocabulary learning) were found when only including participants who completed Version 2 of the SCS1-S, which was found to have higher internal-consistency and reliability.

Figure 3.3. Partial slopes for each cognitive skill predicting scores on the SCS1 and course grades.



Note: Partial slopes for each standardised cognitive skill predicting raw scores on the SCS1-S in the top row of graphs, and on the course grades in the bottom row of the graphs. For each graph the other predictors and control variable are held constant. The shaded area represents a pointwise confidence band based on standard errors. The lines on the horizontal axes show the locations of the scores on the cognitive tests.

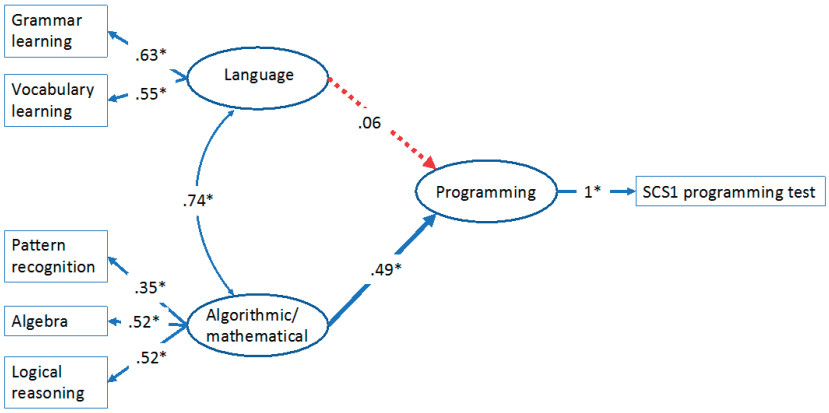
3.3.4 Structural equation modelling

We used structural equation modelling (SEM) with chi-squared tests to test 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 one latent variable. Comparison of the two models indicated that the model with separate latent variables for mathematical and language skills explains 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 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, so a good match is indicated by a non-significant *p*-value (so 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 fit. For the current models, we found good fits for both the model with SCS1 (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 3.4 and 3.5 depict the full models with fitted parameters.

The latent variable for maths significantly predicted scores on the SCS1 programming test ($\beta = .493$, $SE = .241$, $z = 2.044$, $p = .041$), and course grades ($\beta = .542$, $SE = .269$, z = 2.018, $p = .044$). However, the latent variable for language did not predict scores on either the SCS1 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

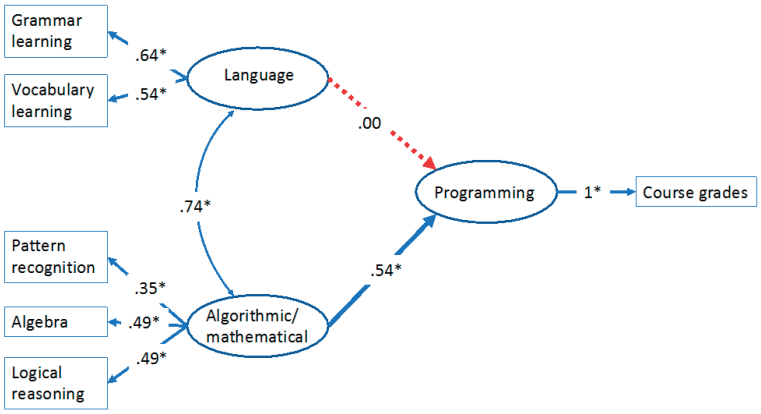
the latent variables for language and algorithmic/mathematical skill, both for the model with the SCS1 (β = .736, *SE* = .108, *z* = 6.850, *p* < .001) and for the model with the course grades (β = .738, *SE* = .113, *z* = 6.507, *p* < .001). Full tested models with estimates, including covariance can be seen in Figures 3.4 and 3.5.

Figure 3.4. Structural model with generalised programming skill.



Note: Structural model representing the relationship between language and algorithmic/mathematical thinking and generalised programming skill as measured by the SCS1-S. Estimates are written along their corresponding model lines. Arrows for which the estimate was significant at *p* < .05 are coloured in blue. Arrows for which the estimate was not significant are coloured in red. *indicates *p*-values below .05.

Figure 3.5. Structural model with course-related programming skill.



Note: Structural model representing the relationship between language and algorithmic/mathematical thinking and course-related programming skill as measured by the course grades. Estimates are written along their corresponding model lines. Arrows for which the estimate was significant at $p < .05$ are coloured in blue. Arrows for which the estimate was not significant are coloured in red.*indicates *p*-values below .05.

3.4 DISCUSSION

The aim of the current study was to examine which cognitive skills predict programming performance in 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 clustered into an algorithmic/mathematical component (comprising pattern recognition, algebra and logical reasoning) and a language component (vocabulary learning and grammar learning), and whether these components predicted programming success.

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 'pseudo' programming language (SCS1-Short). In both cases the skills were predictive after controlling for programming experience prior to the start of the course. Second, based on the results of older 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. Last, we found that the cognitive skills clustered into an algorithmic/mathematical component consisting of logical reasoning, algebra and pattern recognition, and a language component consisting of grammar learning and vocabulary learning. Given the language-like nature of the programming languages used in the course and in the SCS1-S, we expected that both components (algorithmic/mathematical and language) would be predictive of programming skill. The algorithmic/mathematical component was indeed predictive of both generalised programming performance and course-related programming performance, but the language component was not predictive of either type of programming performance.

3.4.1 Generalised versus course-related programming performance

What explanations 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 explanation 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 pseudo programming language on the spot, and did not allow participants to use a calculator or the internet, while the course tests 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 be expected to result in cognitive skills having less predictive value for this kind of assessment, as we found here. 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 assessments, with time limits, only one attempt and sometimes restricted use of outside sources. Therefore, it is possible that the cognitive skills we examined have more predictive value in 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.

3.4.2 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. Penna 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 Penna and Tirre's (1992) tests. Penna and Tirre (1992) and Webb (1985) used similar visual tests of pattern recognition producing differing results (ours differed, relying on numbers). Thus, it remains unclear which dimensions of programming skill, if any, rely on which dimensions of 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. This efficiency requires the programmer to have good knowledge of syntax in order to make optimal use of control structures such as loops or functions, which could be argued require the recognition of code patterns that can be captured by one such control structure. 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.

3.4.3 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 and colleagues (2020) found that language aptitude predicted 17 percent 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, to memorise 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 program, 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 memorization aspect of language is most important when learning to program, and that this aspect was underrepresented in the language component 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 assessments focus primarily on the first two levels: the problem level and the object level. This is also the case in our current programming assessments, where students were allowed to use outside sources or a pseudocode guide to copy syntax. Neither assessment 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.

3.4.4 Conclusion

Overall, the current study confirms 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. Despite the fact that modern programming languages resemble natural languages, this study found little evidence for language skills as predictors of programming success. These results differ from those found by Prat et al. (2020). We suggest that this is 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 reconcile our results with those of Prat et al. (2020) 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).