Computer programming skills: A cognitive perspective

Graafsma, Irene

DOI:
10.33612/diss.168003240

Computer programming skills: A cognitive perspective

Irene L. Graafsma

The research reported in this thesis has been carried out under the auspices of the Center for Language and Cognition Groningen (CLCG), the research school of Behavioral and Cognitive Neuroscience (BCN) of the University of Groningen, and the International Doctorate for Experimental Approaches to Language And Brain' (IDEALAB) of the Universities of Groningen (NL), Newcastle (UK), Potsdam (DE) and Macquarie University, Sydney (AU).

Publication of this thesis was financially supported by the Graduate School of Humanities (GSH) of the University of Groningen and by the research school of Behavioral and Cognitive Neuroscience (BCN) of the University of Groningen.

Groningen Dissertations in Linguistics 201

Cover: Ella Hope
Printed by Ipskamp Printing (NL) - www.ipskampprinting.nl

Computer programming skills: a cognitive perspective

PhD thesis

to obtain the joint degree of PhD at the
University of Groningen, University of Potsdam, Macquarie University and Newcastle University

on the authority of the
Rector Magnificus of the University of Groningen Prof. C. Wijmenga,
President of the University of Potsdam, Prof. O. Günther,
Deputy Vice Chancellor of Macquarie University, Prof. S. Bruce Downton, and
Vice Chancellor of Newcastle University, Prof. Ch. Day

and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on

Thursday 29 April 2021 at 14:30 hours

by

Irene Lotte Graafsma

born on 19 April 1993
in Saint Martin D'Hères, France

Supervisors

Prof. Y.R.M. Bastiaanse
Prof. L. Nickels

Co-supervisors

Dr. E. Marinus
Dr. S. Robidoux
Dr. S. Popov

Assessment Committee

Prof. M.J. Guzdial
Prof. M. Nissim
Prof. G.J.M. van Noord
Prof. P.C.J. Segers

Acknowledgements

This PhD has been a great academic and personal adventure. Being able to do research in a fascinating, relatively new field whilst living in two amazing countries has been a fantastic opportunity that would not have been possible without the extensive support of a large group of wonderful people.

First of all, I would like to thank my supervisors. I want to thank my promotor Roelien Bastiaanse for her help and support, especially with Chapter 5, and for her general help with the thesis writing and submission process. Her extensive experience has been very valuable in shaping this thesis. Special thanks go out to Eva Marinus, without whom the project would have never existed. She supervised me throughout the entire PhD, and has read my drafts many, many times. Although we were only in the same physical place briefly at the start of the PhD, Eva has stayed very involved and has inspired me with her passion for the subject. I also want to thank both Lyndsey Nickels and Serje Robidoux for their help and supervision during my time at Macquarie, and for their help with the analysis and writing once I was in Groningen. I also want to thank them for providing such a supportive and friendly atmosphere at Macquarie, where I very quickly felt at home. I also want to thank Serje for helping with the testing during the pilot study. I want to thank Nathan Caruana for supervising the writing of Chapter 4. Thanks to his thorough and always very timely feedback the writing process for this chapter was very smooth. I also learnt a lot, both about autism research, and about the writing and publishing process. Finally, I want to thank Srdjan Popov for his help and supervision of Chapter 5. I would never have been able to set up, run, and write up this experiment without his help.

I would also like to thank my assessment committee, Mark Guzdial, Malvina Nissim, Gertjan van Noord and Eliane Segers. Thank you for reading my thesis in such detail, and for your helpful comments.

I also want to thank Lesley McKnight and Katie Webb at Macquarie University, and Alice Pomstra, Marijke Wubbolts and Christina Englert at the University of Groningen for all their help with organisational matters.

I would also like to thank the IDEALAB program in general, for making this beautiful international PhD program possible. Specifically, I would like to thank the directors, David Howard, Barbara Hole, Lyndsey Nickels, Roelien Bastiaanse and Gabriele Miceli. Thank you

for the amazing program, and in particular for the summer and winter schools, where I learnt a lot from your questions and expertise. I would also like to thank the other IDEALAB PhD students for their excellent questions and feedback during the winter and summer schools and for making the moves across countries more fun.

Throughout the PhD I have had help from several other kind individuals. Special thanks go out to Matthew Roberts. Thanks to Matt we were able to test all students from the programming units in the Department of Computing at Macquarie. Matt even arranged for testing to be integrated in the course tutorials. The first three chapters of the thesis would not have been possible without his help and collaboration. Matt also contributed to the design of the studies and was our programming and computing expert in the Cognition of Coding research group.

I would also like to thank the other two members of the Cognition of Coding group, Vince Polito and Judy Zhu. They contributed many good ideas, as well as important help and feedback with test selection and design. They also contributed to the writing and publication process of Chapter 3.

I specifically want to thank those who helped with recruitment of participants for the EEG study: Jan van Beek, Rik Teerling and Bart Barnard. More generally, I also want to thank the department of Computational Linguistics at the University of Groningen, as well as the student interns in the Department of Computing at Macquarie University who helped design the EEG stimuli and helped with the pilots and test digitisation for Chapters 2, 3 and 4.

I want to thank Ella Hope for designing the beautiful cover of this book. She immediately understood what I was looking for, made an amazing design and helped me to get it printed correctly.

Also, I am very grateful to all the (former and present) members of the Neurolinguistics research group (not an exhaustive list): Adrià, Aida, Annie, Atilla, Ben, Dörte, Frank, Jakolien, Jidde, Juliana, Kaimook, Liset, Nathaniel, Nermina, Pauline, Roel, Roelant, Roelien, Sara, Seçkin, Serine, Srdan, Suzan, Svetlana, Teja, Vânia, and Yulia. Thank you for the interesting meetings, and for your help and feedback on my presentations. In particular I want to thank Seçkin for teaching me to do ERP analysis for Chapter 5 in Matlab.

This allowed me to analyse my date during lockdown and has greatly contributed to finishing my thesis in time. I also want to thank Frank for answering all my questions about the submission and printing process.

Special mention should also be made of those in Sydney who helped me when I mistakenly decided that we could use printed tests for the pilot. Vince, Judy, Andrea and Leonie kindly helped me sort, assemble and staple over 200 paper tests. During these hours we learnt important lessons, such as how to best set up an assembly line, how after a while, your fingers seem to automatically be able to find the correct page, and most importantly, to never use paper tests again if it can be avoided.

I want to thank those fellow PhD students who have become my good friends over the last three years, as well as my non-PhD friends who supported me throughout this project. You are what makes this work fun. Thanks for all the support, the talks, the laughs, the rants and the general good times: Andrea, Winnie, Pauline, Aida, Hannah, Leonie, Diane, Suzan, Britta, Ana, Frank, Simon, Michiel and others. Special thanks also go to Jidde and Myra for letting me and Izaak live in their apartment while they are in Sydney. It has been the perfect place to start our stay in the Netherlands.

Of course, my academic career would never have existed if it wasn't for the support and encouragement of my parents, Heinz and Angelien, and the general support of my close family: Denise, Chantal, Bart, Babette, Rob, Robert, Karin, Marga and others.

Finally, special thanks go out to my partner, fiancé and now husband Izaak Lea, who moved with me twice for this PhD, once interstate and once intercontinentally, and didn't complain about it once. Thank you for your never-ending positive attitude and support. Due to the pandemic, the final year of work for my PhD took place almost exclusively at home, and your company is what kept me sane.

CONTENTS

List of figures

List of tables

Chapter 1

General Introduction

1.1 HISTORY OF PROGRAMMING AND DEVELOPMENT OF PROGRAMMING LANGUAGES

In the early 19th century, Joseph Marie Jacquard created a device to read punched cards that carried information on weaving patterns for the manufacturing of fabrics. His machine, combined with a traditional loom, simplified the process of manufacturing textiles by automating the creation of complex patterns (Essinger, 2004). This work formed the basis for the analytical engine created by Charles Babbage later that century, which consisted of columns that could be assigned different values for the mechanical machine to execute various mathematical computations (Dasgupta, 2014). The punched cards of Jacquard and the column structures of Babbage's machines were the first patterns designed for people to give instructions to a machine and could therefore be considered early forms of programming languages. Between 1842 and 1849, Ada Lovelace wrote the world's first published computer program, which consisted of a detailed method for calculating Bernoulli numbers with Charles Babbage's Analytical Engine (Fuegi & Francis, 2003).

Although these early machines and programs had a structured system designed for humans to communicate instructions to machines, these instructions were still limited to a single purpose, and were very abstract and technical. They did not resemble modern programming languages. In addition, despite these initial developments, it took almost a century for these types of instructions to be used more widely. Moreover, it was not until the 1940s that the first modern programming languages were created, that is, a full language-like system designed to communicate a wide range of instructions to a computer. However, these programming languages were still highly inefficient. Their use took a considerable amount of mental effort, because instructions had to be written in machine code directly (Knuth & Pardo, 1980). As machine code is the code that directly influences the computer hardware, a programmer was required to understand and instruct the exact steps that the computer hardware had to perform to execute a certain task. This required extensive hardware knowledge and very detailed and extensive code for operations that would today be considered very simple (Knuth & Pardo, 1980).

In the 1950s, there was rapid development of several programming languages, sparked by the development of the first compilers. A compiler is a program that translates instructions from a high-level language into the machine code that operates the computer

hardware directly, making it no longer necessary for programmers to write programs in complicated and inefficient machine code (Aho et al., 2007). In the 1960s and 70s, several higher-level programming languages were developed (e.g., Speakeasy, Smalltalk, C) that still form the foundation of most programming languages used today (Bergin & Gibson, 1996). In the 1980s, the focus was on developing programming languages to be more efficient and to allow for higher-level structures (e.g., classes for variables). In the 1990s, web programming gained importance and suitable "Rapid Application Development" languages, such as Java were developed (Bergin & Gibson, 1996). These later languages from the 90s and early 2000s were developed to increasingly resemble natural human languages (Fedorenko et al., 2019; Paulson, 2007).

With the creation of the first high-level programming languages also came the first programming education. The earliest programming languages were primarily developed to operate specific programs. Of course, they still had to be learned by the early programmers, but they were not suitable for general programming education. This made it difficult to learn to program, and only a select group of science and computing students engaged in such education programs. It was not until the 1980s, when the first computers became widely available, that there was a big spike in programming education, in primary and secondary school and for undergraduate students. This led to the first efforts to make programming languages more suitable for learners. To achieve this, three approaches were used: 1) mini-languages, these are languages specifically developed for learning, such as LOGO (Brusilovsky et al., 1997); 2) sub-languages, languages that select a subset of commands from a bigger language, such as Professor for Java (Gray & Flatt, 2003); and 3) an incremental approach, where a complex programming language is learnt by starting with just a few commands and then adding more and more complex commands (Hermans, 2020).

When click interfaces became more common in the 1990s and early 2000s, interest in widespread programming education seemed to diminish (HackerRank, 2018). However, over the last decade interest in programming education has started to rise again, with more employers demanding relatively advanced digital literacy from their employees in order to be able to customise analyses, applications or web pages (Rushkoff, 2012).

This recent increase in programming education has been accompanied by a renewed interest in programming as a skill. Since the 1990s, the field of computer science education has largely focused on different ways of teaching programming and the goals and motivations of learners (Guzdial, 2015). However, to teach programming optimally, there needs to be an understanding of which skills and traits underpin it. With a skill that is only 50 years old, which has evolved along with rapid development of different programming languages, requiring different skills, there is still much to discover in this area. This includes which skills or personality traits are important when learning to program in a modern-day context and how programming is processed in the brain. In order to start answering these fundamental questions about programming as a skill we need to study it from a cognitive perspective.

1.2 THE COGNITIVE PERSPECTIVE ON PROGRAMMING

The term “cognition” refers to the way the mind internally represents the external world and performs the mental computations required for all aspects of thinking. Cognitive science is a broad field of study focussing on the vast set of mental operations associated with such things as perception, attention, memory, language, and problem solving (Gazzaniga et al., 2009). The field uses a range of methods to study these processes, including studying the relationships between a target skill (like reading or programming) and specific underlying cognitive processes, the workings of the brain related to certain skills, and individual differences such as personality traits that may influence the acquisition of skills.

The cognitive perspective is particularly important with regards to programming education because it can tell us more about the nature of programming as a skill. This gives us the fundamental information necessary to teach programming, but also to predict who will be good at programming, and ultimately even to adjust programming languages themselves to be more suitable for the way people understand and learn.

Although the cognition of programming has not been the main focus of programming education research, there have been some important contributions in this area (Robins et al., 2019). Guzdial and du Boulay (2019) identified two main streams of cognitive research with different objectives. One stream focuses on whether cognitive

benefits accrue from learning to program (e.g., Pea & Kurland, 1984). The other stream researches which cognitive skills are important when learning to program. However, most studies in these streams were conducted in the 1980s and 1990s. Critically, since then the nature of both programming languages and of programming education has changed (Guzdial & du Boulay, 2019). At present, most computer science education research is performed by computer scientists and computer science education researchers, and less is conducted by psychologists and cognitive scientists. However, recently, the cognitive perspective has started to (re)gain more interest. Labs such as that of Professor Amy Ko at the University of Washington, have started to use an interdisciplinary approach to study topics such as programming language learning, and programming and problem solving. In particular they have contributed to the topic of programming and self-regulation (Xie et al, 2020) and to theories of programming instruction (Xie, Loksa et al., 2019). Other recent research (e.g., Prat et al., 2020) looks at cognitive skills and brain activity in relation to learning in a programming course. These researchers are starting to lay the foundations of modern-day cognitive research of programming. However, it is also clear that there are still many gaps regarding our understanding of the skills involved in learning to program (Prat et al., 2020; Xie, Loksa et al., 2019). In addition, not much is known about the neural processes involved in programming skills (Floyd et al., 2017; Prat et al., 2020; Siegmund et al., 2014). Even fewer studies have been performed focussing on autistic traits playing a role when learning to program (Wray, 2007). Therefore, it is interesting to study programming from a broad cognitive perspective in the modern-day context.

The studies in the current thesis fit into the second stream described by Guzdial and du Boulay (2019): they focus on which cognitive skills are important when learning to program, not on the benefits of learning it. They use two cognitive research approaches. The first is conducted in a programming education setting, aiming to learn more about the nature and subskills of programming by studying the skills and personality traits that predict learning success.

Within the first approach, the first step is to develop necessary methodology by adapting and validating an instrument to measure programming skill in university students (Chapter 2), specifically answering the research question:

(1) Can we create two reliable parallel short versions of the Second Computer Science 1 (SCS1) programming test?

This instrument is then used to test which cognitive skills predict programming success (Chapter 3), aiming to answer the research question:

(2) Which cognitive skills predict success at the end of a programming course?

The short versions of the SCS1 are also used to investigate which individual differences in personality, in the form of autistic traits, predict programming success (Chapter 4), focussing on the research question:

(3) Do autistic traits predict success in a programming course?

The second cognitive approach of this thesis is the use of neuroimaging to examine the brain activity related to programming and compare this to other skills for which the brain activity patterns are known. Specifically, in Chapter 5, Event Related Potentials (ERPs) are used to compare processing of a programming language and natural human languages in the brain, answering the research question:

(4) Is a programming language processed similarly to a natural language?

1.3 THEORETICAL FRAMEWORKS AND COGNITIVE MODELS OF PROGRAMMING

In more established cognitive research fields, such as memory and language, there are many theoretical frameworks that describe the processing that underpins these skills. Even for more specifically education-related skills such as reading and mathematics, several theoretical models are available on which to base research, allowing researchers to formulate explicit hypotheses and make specific predictions about potential outcomes. Examples of such models are the E-Z Reader and SWIFT models for reading (Rayner, 2009), and the problem-based learning model for mathematics (Mulyanto et al., 2018). In programming research, however, such models are still lacking.

In the 1980s and 90s, some models and frameworks were developed for problem solving in programming. These were mainly based on the results of studies that used think-aloud protocols where a participant is asked to verbalise their thought processes whilst solving a problem, such as Brooks' theory of the comprehension of computer programs (Brooks, 1977; 1978; 1983). A second range of models simulated the debugging process, such as the PUDSY system by Lukey (1980). More recently, Bednarik and Tukjainen (2006)

developed an eye-tracking model to characterise program comprehension processes. These models form an interesting start, however, none focus on the full programming process, but rather on a programming-related subskill, such as problem solving and debugging. Additionally, think-aloud studies typically use few participants and a limited number of tasks, because the reasoning process for each question has to be observed and analysed individually. Therefore, models based on these methods may be specific to that participant and task, and thus difficult to generalise. In cognitive science, researchers typically formulate both detailed models of subskills, and higher-level models that aim to provide broad generalisable understanding of a skill as a whole. These previous models of programming are lacking this broad overview and generalisability. Compared to reading and mathematics, programming is more complex, involving a wider range of subskills and, therefore, harder to model. However, in order to study programming skill in its entirety, a theoretical model of the full programming process is needed.

Recently, Armoni (2013) described a theoretical model of the programming process, called the PGK-hierarchy (named after Perrenet, Groote and Kaasenbrood who first defined it; Perrenet et al., 2005), which does aim to describe the programming process in general, regardless of the specific task or programming language (see Figure 1.1). According to the PGK-hierarchy, the programming process consists of four levels, each with a different purpose and different skills involved: The problem level, the object level, the program level and the execution level. At the problem level the programmer assesses the problem and the solution that is being asked for. Then, at the object level, an algorithm is formulated to solve the problem. This algorithm is not yet written out in a specific programming language, this happens at the next level, namely the program level. Finally, at the execution level, the computer interprets and executes the algorithm.

In the current thesis, the PGK-hierarchy is used to guide the predictions for the experiment described in Chapter 3, where we select cognitive skills that are possibly related to the first three levels of the model. I argue that the first two levels (the problem level and the object level) are related to algorithmic thinking or mathematical skills, while the third level, the program level, is related to natural language skills.

The relationship between language skills and programming performance has not been extensively tested, but several researchers have argued for a connection (Fedorenko et al., 2019; Hermans & Aldewereld, 2017). Throughout the current thesis, I explore this connection by looking at language skills as predictors of performance in a programming course; the relationship between learning success, language skills and autistic traits in programming students; and by comparing neural processing of syntax violations in a programming language, to grammar violations in programmers' first and second natural languages.

Figure 1.1. PGK-hierarchy model and related skills.

Level	Problem level	Object level	Program level	Execution level
Aim	Clarify problem and find solution	Translate high-level solution into an algorithm	Implement the algorithm in a specific programming language	The computer executes the code
Type of cognitive skills	Algorithmic thinking and mathematics		Language	No related cognitive skills

Note: This figure shows the four levels of the PGK-hierarchy model, first defined by Perrenet et al. (2005) and further described by Armoni (2013). The four levels are described in the top row, the aims of the levels as described by Armoni are listed in the second row. The third row indicates which type of cognitive skill I expect to play a role during each level of the hierarchy.

1.4 METHODS IN PROGRAMMING EDUCATION RESEARCH

Early programming education research in the 1980s and 90s mostly focussed on comparing novice to expert performance, specifically focussing on the number and different types of programming errors made by these groups. The focus was on overall programming performance, without relating this to underlying cognitive skills. The goal of these efforts was to optimise both teaching and the programming languages themselves (Guzdial & du Boulay, 2019). When the psychology of programming movement gained importance in the 1970s and 80s, the research focus shifted away from the programming languages and interface and put increasing focus on modelling the user. Over time, studies changed from think-aloud experiments with just a few participants in a laboratory to studies with larger

groups of learners and over longer periods of time in a more natural classroom setting (Guzdial & du Boulay, 2019). The teachers of these classes typically used a constructivist approach, where students were encouraged to learn programming languages by using them, without much direct instruction (Hermans, 2020; Portnoff, 2018). Recently, some researchers have argued that direct instruction, similar to the methods used in second language education and learning to read, are more beneficial (e.g., Hermans, 2020; Hermans & Aldewereld, 2017; Portnoff, 2018). However, this is still not the norm in today's programming education (Hermans, 2020). The students studied in the current thesis were still taught in a largely constructivist way. Another characteristic of the early classroom studies was that they used sample sizes that would today be considered small, usually consisting of a single classroom with 20 to 30 students per experiment. Today much larger samples are used in educational studies. Accordingly, the studies described in Chapters 2, 3 and 4 of this thesis are conducted with all the students from an undergraduate programming course, providing from 282 to 354 participants across the various chapters.

Another important development for the generalisability of the results of empirical research in modern computer science education was put forward by McCracken et al. (2001) who performed the first Multi-Institutional Multi-National (MIMN) computer programming research study. They emphasised the importance of testing programming skill and processes across different countries and institutions. This thesis acknowledges the importance of generalizability and implements the MIMN principle by further validating and using an independent programming test in Chapter 2, that has been used across countries and institutions, as well as by using this test in Chapters 3 and 4, thereby making the results easier to compare to existing and future studies that use the same instrument.

1.5 NEUROIMAGING METHODS IN PROGRAMMING LANGUAGE RESEARCH

In addition to examining which skills predict programming performance in education, the current thesis examines brain responses related to this skill. For natural language skills, such as reading and listening, it has been established how the brain responds to violations that are syntactic or semantic in nature.

Programming languages, however, have not yet been studied as extensively through neuroimaging. At the start of the current PhD project, a few studies examining

programming skills using functional Magnetic Resonance Imaging (fMRI) had been published (e.g., Floyd et al., 2017; Siegmund et al., 2014). The results of these studies suggest that areas that have been identified as being involved in natural language processing are activated during programming as well (Siegmund et al., 2014) and that brain activation patterns when reading a programming language are almost indistinguishable from the activation pattern during natural language reading once a programmer is sufficiently proficient in the programming language (Floyd et al., 2017). However, towards the end of the current PhD project, another fMRI study was published by Ivanova et al. (2020). They argue that reading and understanding programming languages is more reliant on the multiple demand system, which is the brain network most associated with mathematics, than on the language system. They also suggest that programming cannot be fully equated to either maths or language, but rather seems to rely on a novel cognitive network. However, their results also suggest that although they found that the language system was not activated in the same way for code as for prose, parts of the language system were still involved when reading code, particularly in the more language-based programming language Python. In addition, Prat et al. (2020) recently studied whether differences in resting brain activity were associated with success in learning programming. This study aimed to predict programming learning success in a university course from participants' performance on a number of cognitive tasks (including language tasks) and the students' resting state brain activity (measured with electroencephalography, EEG). They found that learning success in the programming course was mostly predicted by language skills, problem solving skills and working memory, while resting state EEG had little additional predictive value.

Although all four of these previous studies suggest that language skills play a role when programming, it remains unclear whether this means that the brain processes a programming language similarly to a natural language, and which elements of a programming language are similar or different to elements of natural languages. In order to directly study these differences we need to use a neuroimaging technique that allows us to measure responses to specific linguistic elements and compare those across languages.

One method to study brain responses to language is to use Event Related Potentials (ERP) in EEG signals. EEG, when measured in response to a specific stimulus, has a high

temporal resolution. While fMRI can tell us which brain areas are involved in a specific process, its temporal resolution is poor, making it difficult to link a response to, for example, a specific word in a sentence. EEG however, is very accurate in its timing, and can therefore be used to measure responses to specific words or symbols in spoken or written language. Typically, ERPs are used when studying natural languages and their structures (Carreiras & Clifton, 2004; Friederici et al., 2002). Thanks to research in many different natural languages we can make predictions about ERP-effects when a reader or listener is presented with a specific grammatical or semantic violation.

However, to date, we are not aware of any ERP studies examining violations in programming languages. Therefore, in Chapter 5, ERP responses to violations in a programming language (Java) are compared to those elicited by violations in the participants first and second natural languages. The results of this study directly contribute to our knowledge of programming languages by showing whether a violation in such a language elicits a similar response to a violation in a natural language. It also allows us to determine whether processing a programming language is more similar to processing of a second language than a first language.

1.6 CONTRIBUTION OF THE THESIS

With the four experimental studies described in this thesis, I aim to contribute to a better understanding of programming as a skill in modern society and education. The studies contribute to four different areas that are important in studying programming from a cognitive perspective: standardised measurements of programming skill (Chapter 2), cognitive skills that are predictive of programming performance in beginning programmers at the university level (Chapter 3), the relationship between programming and autistic traits (Chapter 4), and the use of brain responses to study similarities and differences between processing of programming language and natural human languages (Chapter 5). With the results of the studies in these four areas I aim to contribute to the existing knowledge on the cognition of programming and to lay foundations for future research. In the General Discussion (Chapter 6), I summarise and discuss the main methodological contributions and theoretical findings of the thesis, and present future directions for research in this field.

Chapter 2

Validating two short versions of the Second Computer Science 1 programming ability test[1]

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

2.1 INTRODUCTION

The use of digital technology in daily life has seen a large increase over the last two decades. As a result, both current and future generations need to be able to use digital technology (Rushkoff, 2012). In addition, a substantial proportion of the future workforce will need training to work in fields that develop these technologies (Bailey & Mitchell, 2006; Seegerer et al., 2019). Because of these changes, programming has gained major interest and importance in educational systems worldwide (Balanskat & Engelhardt, 2015). As we see more careers requiring programming skills at the tertiary level, more students choose to enrol in Computer Science degrees, and more students from other fields choose to undertake additional computer programming courses (European Commission, 2018; Marasco et al., 2017).

Although there has been an increase in demand for programming education, little is understood about how programming skills are best acquired and what characterises optimal training. To answer such questions, we need to combine knowledge and expertise from computer science education with other disciplines such as cognitive science and psychology (Guzdial, 2015). A prerequisite to researching the acquisition of programming skills is a validated instrument to measure programming skill that allows for comparisons between different research studies and that can measure change within a study (Parker et al., 2016). The present research aims to contribute to the development of such an instrument.

Aiming to measure programming skill in undergraduate novice programmers, Parker et al. (2016) developed the Second Computer Science 1 (SCS1) assessment. To date, two studies (Parker et al., 2016; Xie, Davidson et al., 2019) have investigated the difficulty and the reliability of the SCS1 test items. However, there are still limitations to the current SCS1 test and its validation. 1) The test takes an hour to administer, limiting its use in many research studies with time constraints. 2) It does not offer multiple versions to allow for repeated testing without introducing possible test-retest effects. This restricts the test's usefulness for research studies with an interest in measuring improvement over time. 3) No study to date has examined the SCS1's external validity. This means that we don't know how performance on the SCS1 relates to performance on other tests of programming ability, such as course exams. 4) The difficulty and reliability of the SCS1 were tested only

using American undergraduate students, and at only two institutions (Georgia Institute of Technology university and the University of Washington). As a result, it is still unknown whether the SCS1 performs similarly for other student populations, limiting its use for researchers at other institutes.

The aim of the current study is to address these four limitations by creating and validating two parallel short versions of the SCS1, the SCS1-Short Version 1 (SCS1-S1) and Version 2 (SCS1-S2). By doing so, we aim to address the first and second limitation by reducing testing time and allowing for repeated testing. We address the third limitation by testing the SCS1's external validity for the first time, by correlating scores on the short versions, and for individual items, with performance on programming assessments in a programming course. Lastly, we will assess the difficulty and internal-consistency of the short versions, thereby addressing the fourth limitation by generalizing knowledge on test and item quality to a new student population. In the sections of the Introduction that follow, we further elaborate on the characteristics of the SCS1 and on each of these four limitations.

2.1.1 Second Computer Science 1 (SCS1) as an assessment tool

The SCS1 (Parker et al., 2016) was developed as a parallel version of the Foundational Computer Science 1 (FCS1) Assessment (Parker et al., 2016; Tew, 2010; Tew & Guzdial, 2011). The FCS1 is no longer publicly available. However, Parker et al. (2016) developed a very similar test in the SCS1, consisting of the same topics and question types, but with different content.

The SCS1 consists of 27 multiple choice questions requiring students to read, evaluate and complete pieces of code in a pseudo-code programming language. Although it is impossible to develop a test that is fully independent from specific programming languages, by using a pseudo-language (Guzdial, 2019), Parker et al. (2016) aimed to make it as independent as possible from the main languages in undergraduate computer science. The SCS1 questions were designed to measure 9 different programming content areas (basics - i.e., applying simple mathematical formulae -, for-loops, while-loops, if-statements, logical operators, arrays, function parameters, function return values,

recursion) with three different question types (definitional, code tracing, code completion) each with five answer choices. In addition, students receive a two-page pseudo-code guide, which they were instructed to use as a reference when answering the questions.

2.1.2 Need for short versions

Behavioural research studies often require a number of different experimental measures. However, total testing time is usually constrained because of limits in both participant attention span and research funding. This means that using an assessment that is as long as the SCS1 is often not feasible. One way to deal with test length is to use a subset of items. Although Parker et al. (2018) did this in a later study, the subset used in that study was not validated separately. This means that the difficulty, validity and reliability of that subset are unknown. Consequently, there remains a need for a validated, short assessment of programming skill.

2.1.3 Need for parallel versions

Additionally, an important goal in any field of education is to be able to measure progress on a certain skill (Marston et al., 1986), for example, in the context of comparing different courses or interventions. One recent study by Nelson et al. (2017) attempted to measure change in programming skill by administering the SCS1 before and after a one-day course where participants were randomly assigned to two conditions with different course structures. This design aimed to compare improvement across conditions. However, because they used the same items twice, their conclusions are confounded with test-retest (practice) effects. This means that the true improvement as a result of change in underlying skill is likely lower than reported, as the participants could have benefitted from attempting the same questions for a second time. One way to minimise test-retest effects is through the use of two parallel versions of a (programming) test, allowing researchers to test students twice without using the same questions. Critically, parallel versions of a test should be equal in difficulty and measure the same knowledge.

2.1.4 Lack of evidence for external validity

It is essential that any test assesses what it is supposed to be measuring: the SCS1 should

reflect the programming skill of novice computing students. Moreover, the skills measured should reflect a broad scope of programming knowledge, covering all central topics. When Parker et al. (2016) validated the SCS1 with students from three different introductory programming courses, they did not report the correlations between the SCS1 and course grades, but did report these correlations for the FCS1, on which they based their SCS1 questions. Scores on the FCS1 showed a significant correlation of $r = .483$ ($p < .001$) with grades on a Python course for students with Computer Science majors ($n = 140$), but no significant correlations with grades on a MATLAB course ($r = .509$, $p = .076$) for a small sample of students with Engineering majors ($n = 13$), or grades of students undertaking a media computation course ($n = 30$) ($r = .298$, $p = .110$). It is possible that these correlations did not reach significance because of the small sample sizes. Parker et al. (2016) did find that the total scores on the SCS1 correlated with the total scores on the earlier FCS1 Assessment. However, this correlation was only moderate ($r = .57$), which was surprising, as the questions of the SCS1 were modelled on the questions of the FCS1.

Xie, Davidson et al. (2019) tested the relationship between the different questions of the SCS1 using confirmatory factor analysis. They showed that 24 of the 27 questions loaded on one latent variable, presumably programming skill. However, questions 20, 24 and 27 did not load on this variable and therefore seem to measure something else. Hence, while the majority of SCS1 questions seem to measure related concepts, it remains unclear to what extent those concepts are programming knowledge specifically. That is, the extent to which the test has external validity.

2.1.5 Reliability and difficulty

Both Parker et al. (2016) and Xie, Davidson et al. (2019) tested the internal-consistency and reliability of the SCS1 by calculating Cronbach's alpha. Parker et al. (2016) found moderate internal-consistency and reliability (Cronbach's alpha = .59), but this was higher in Xie, Davidson et al. (2019) (Cronbach's alpha = .70). This suggests that the internal-consistency and reliability of the test might vary across different student populations. To further determine this, the internal-consistency and reliability needs to be tested in more different courses.

In addition to being reliable, an ideal test of programming skill would also be of appropriate difficulty. This means that it should not be too easy or too difficult overall for the target group, and that it should have items of varying levels of difficulty to discriminate between learners with varying knowledge (Xie, Davidson et al., 2019). In their initial validation study, Parker et al. (2016) found that most questions were difficult for beginning programmers. For 22 of the 27 questions, less than half of the students answered correctly, and on the remaining five questions, 51-85% of students answered correctly. Although this showed that many questions were difficult, the chance level would lie at 20 percent correct for questions with five multiple choice answer options. It, therefore, seems that, even on these harder questions, students still performed above chance and that although the SCS1 seems to be rather difficult, it can be effectively used to distinguish between students of different programming ability.

Xie, Davidson et al. (2019) used item response theory for a more fine-grained analysis of the difference between item difficulty and learner knowledge. They tested students with a wider range of programming ability than Parker et al. (2016), by adding a sample of more proficient programmers, therefore generalizing to a population beyond beginner programmers. They found that only four items (Qs 5, 13, 15 and 18) were particularly difficult for their beginner programmers, with only 12-21% of students answering those questions correctly, thus performing at chance level. For the students with more programming experience two of these questions remained difficult (Q13 and 15), while performance on the other two questions (Q5 and 18) increased to 30 and 31% respectively. These results seem to replicate Parker et al.'s (2016) finding that the SCS1 is difficult for beginning programmers, but that most questions are still answered correctly at rates greater than chance. However, the results from Xie, Davidson et al. (2019) cannot be directly compared to Parker et al. (2016). Firstly, Xie, Davidson et al. (2019) excluded participants who attempted fewer than 10 of the 27 questions. It is therefore likely that the weaker programmers were not included in the analysis, thereby making the questions seem easier than if all students were included in the analysis. This might also have led to the higher internal-consistency and reliability. Secondly, Xie, Davidson et al. (2019) used the SCS1 as a pre-test at the start of a programming course, rather than as a post-test at the end. This means that performance was less dependent on the course content, and more

dependent on previous programming experience. The fact that they found differences in performance between beginners and more advanced programmers suggests that the test successfully discriminates between different levels of programming skill. However, it is possible that in the assessment of basic programming skill in true novices the SCS1 is not very useful because of the overall difficulty of the test: It might be better employed after some initial programming education. To further determine the difficulty of the tests and individual items they need to be administered to a wider variety of students with different backgrounds.

In sum, the current study aims to address the four limitations of the SCS1 that we have described. By developing and validating two short versions of the SCS1 in a new student population and by correlating performance with course grades we aim to achieve 1) a shorter testing time, 2) the possibility for repeated testing, 3) knowledge on the external validity of the tests and items, and 4) generalization of knowledge on test and item quality to a student population outside Georgia Institute of Technology. We will compare our findings to those of Parker et al. (2016). However, we will refrain from making a direct comparison to the Xie, Davidson et al.'s (2019) data, as they excluded participants who answered fewer than 10 questions and administered the SCS1 at the start rather than at the end of the course.

2.2 METHODS

2.2.1 Participants

As part of a mandatory tutorial, 668 students undertaking an undergraduate introduction to programming course at Macquarie University (link: https://osf.io/y9bkw/), completed the testing session at the end of their course. Of these participants, 415 (60%) consented for their data to be used for the current study. The students were enrolled in a wide variety of undergraduate majors, such as science, mathematics, business, and computing. As all our test instructions and many of the test items were in English, participants were excluded if they indicated that their level of English was lower than "Good", level 3, on a 5-point rating scale from 1 (minimal) to 5 (native) (16 participants excluded), or indicated that they

cheated or did not seriously attempt to answer the questions in a post-test probe question (48 participants). Thus, the results reported here were from the remaining 354 participants (68 female, 255 male, 31 other or unspecified; mean age 19.87 years, *SD* = 3.52). The study received ethical approval from the Macquarie University Human Research Ethics Committee (Reference number: 5201800224).

2.2.2 Materials

This study reports results that form part of data collected for a larger study, in the context of which we also collected data on other skills (e.g., tests of pattern recognition, logical reasoning and grammar learning skills). Participants were also administered two questionnaires: a sense of agency scale, measuring their feelings of control while programming, and behaviours and personality questionnaire that looks at autistic traits in the general population. Below we describe the two tests that were used for the study reported here: the two adapted SCS1 tests of programming skill and the tests completed by the students for their course grades.

Second Computer Science 1 (SCS1) test

The SCS1 (Parker et al., 2016) formed the basis of our assessment of programming skill. The SCS1 consists of the following content areas: basics (i.e., applying simple mathematical formulae), conditionals, for loops, indefinite/while loops, logical operators, arrays, recursion, function parameters, and function return values. We developed two short versions of the SCS1 by dividing the questions in such a way that both versions covered all content areas and all three question styles (definitional, code tracing, code completion). As the SCS1 has an uneven number of items, we removed one item to make two versions of equal length. Based on the first validation study by Parker et al. (2016) and our own pilot work, we dropped item 15, in the study by Parker et al. (2016) it was among the hardest items with poor discrimination, as less than 50 percent of students answered it correctly, and it showed a point-biserial correlation below 0.1. In our own pilot of 9 participants none answered this question correctly (Xie, Davidson et al., 2019 which was published after the start of our study found similar results for Q15). The division of items across versions resulted in combining the original items 1, 2, 5, 6, 9, 12, 13, 14, 17, 18, 22, 25, 27 into

Version 1, and items 3, 4, 7, 8, 10, 11, 16, 19, 20, 21, 23, 24, 26 into Version 2. Appendices 1 and 2 show the items assigned to each version with the level of difficulty as estimated according to the first validation of the SCS1 by Parker et al. (2016) and according to our pilot work. We kept the original pseudocode guide for participants to use as a guide while answering the questions. Each version was converted into an online format using Qualtrics (Qualtrics, Provo, UT), this was similar to the online format used by Xie, Davidson et al. (2019).

Student grades

These were the student's final course grades on the main course assessments of their university undergraduate programming course. These assessments consisted of six tests each comprising open questions where students were asked to answer conceptual questions or to solve small programming problems. Students could attempt each test three times throughout the course, but were assigned equivalent but different questions on each attempt. Their highest scores counted as their final grades and were used to compute the total grade for the current study. The tests could be completed throughout the semester as well as during the final exam testing session, which took place two weeks after the testing session with the SCS1-S. The six different tests addressed the topics of: fundamentals of computing, variables and conditionals, loops, functions, arrays and strings, program design and problem solving. For more information see the Unit Guide in the Cognition of Coding project on the Open Science Framework (https://osf.io/y9bkw/). We used the raw module scores from five of the six modules, averaged over the subtopics for our analysis and disregarded penalties for incomplete work or lack of attendance. We excluded grades from the module related to the fundamentals of computing, which focused mainly on the history of the field, and did not reflect programming skill as measured by the SCS1.

2.2.3 Procedure

During the last tutorial of the introductory programming course, students were given a link to the Qualtrics survey and were asked to complete the test individually. The Qualtrics survey started with information about the study and participants were asked for consent for their data to be used, and for their anonymised data to be made publicly available for

future research. Participants were then assigned SCS1-S1 or SCS1-S2 based on whether their student number was even or odd. They were informed that they had 30 minutes to complete the test as well as they could. During the test they could scroll back and forth through the questions and saw a clock with the remaining time in the corner of the screen. Skipping questions was allowed in this format. There was a button on the page that opened the pseudocode guide in a separate window, which they could use as a reference to help answer the questions. Students were informed that they were free to also make use of paper and pen to help them solve the problems. Students who missed their tutorials had the opportunity to complete the tests at home for partial attendance credit. 343 of the 354 participants used in analysis completed the experiment during the tutorial, with the remaining 11 completing it at home. Students completed their module tests that together made up their final course grades throughout the semester, with the final test opportunity during the scheduled exam time approximately two weeks after our experimental testing session. For all consenting students we obtained their grades on their module tests once these were finalised and had been released to the students.

2.2.4 Analysis

We ran four separate analyses to address the four aims of the current study. For all analyses we only included participants if they had answered at least one question on their SCS1-S version (167 participants for Version 1 and 170 participants for Version 2). To determine whether it was possible to develop reliable parallel short versions of the SCS1, the first analysis compared the difficulty of both versions by comparing accuracy on each version using a student's *t*-test and a Bayesian *t*-test. The Bayesian *t*-test allows examination of the strength of the evidence for the null hypothesis (there is no difference in the difficulty of the two versions) versus the alternative hypothesis (the two versions differ in difficulty). Bayes factors can be interpreted as a direct ratio and can therefore be interpreted without a cut-off criterion. However, typically the Bayes factors are interpreted so that Bayes factors between 0 and 0.33 show support for the null hypothesis, with lower values showing stronger support. Bayes factors between 0.33 and 3 are considered inconclusive. Bayes factors above 3 show support for the alternative hypothesis, with higher values showing stronger support (Rouder et al., 2009).

Secondly, to examine and compare the external content validity of each version we correlated accuracy on both versions with student's course grades on the programming course. We then compared the correlations across the two versions using Fisher's *z*-transformation.

To determine the reliability and difficulty of the versions, we analysed the properties of the individual items on both SCS1-S versions. We first used classical test theory to examine each item's difficulty by computing the percentage of students who answered each question correctly. Since most students were not able to complete all questions within the allotted 30 minutes, we used two different measures for item difficulty. Firstly, we considered accuracy per item when looking at all participants, for this we counted unanswered items as incorrect. Secondly, we looked at the accuracy per item whilst taking only participants who attempted that item into account, for this we counted unanswered items as missing values. We also considered discriminability of each item by computing point-biserial correlations between the item accuracy and the total score on their SCS1-S version, and with the course grade.

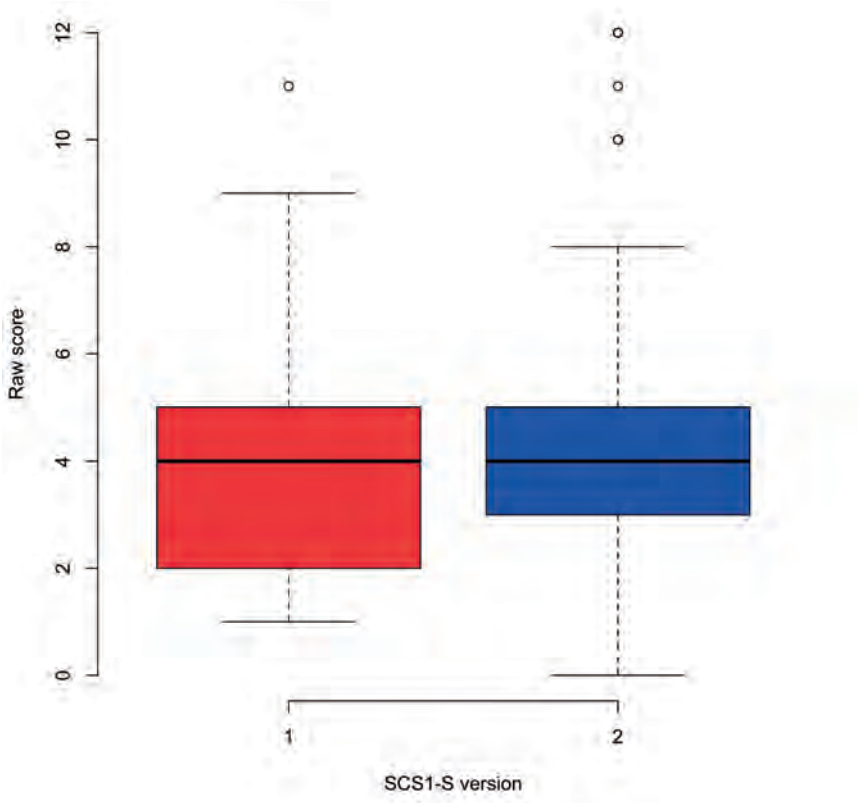
Finally, we used Item Response Theory (IRT) analysis to further examine the quality of the individual items. Mirroring the Xie, Davidson et al. (2019) analysis procedure, we started with a confirmatory factor analysis (CFA) to determine whether all questions loaded on the same factor, which would suggest that the test measured a single concept. Next, we computed Cronbach's alpha for each version separately to examine the internal-consistency and reliability. Finally, we conducted IRT analyses of the test items.

2.3 RESULTS

2.3.1 Test difficulty

On the two short versions of the SCS1-S, 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 ($t(324.63) = -1.61$, $p = .11$; see Figure 2.1 for boxplots). However, a Bayesian t-test showed that the data favoured the alternative hypothesis that the scores on the two versions differ, but only to a modest (inconclusive) degree ($BF_{10} = 2.4$). In other words, although there was no significant difference between the scores on both versions, we cannot conclude that the versions were equal in difficulty for our cohort of participants.

Figure 2.1. Median raw scores and distributions per version of the SCS1-S.

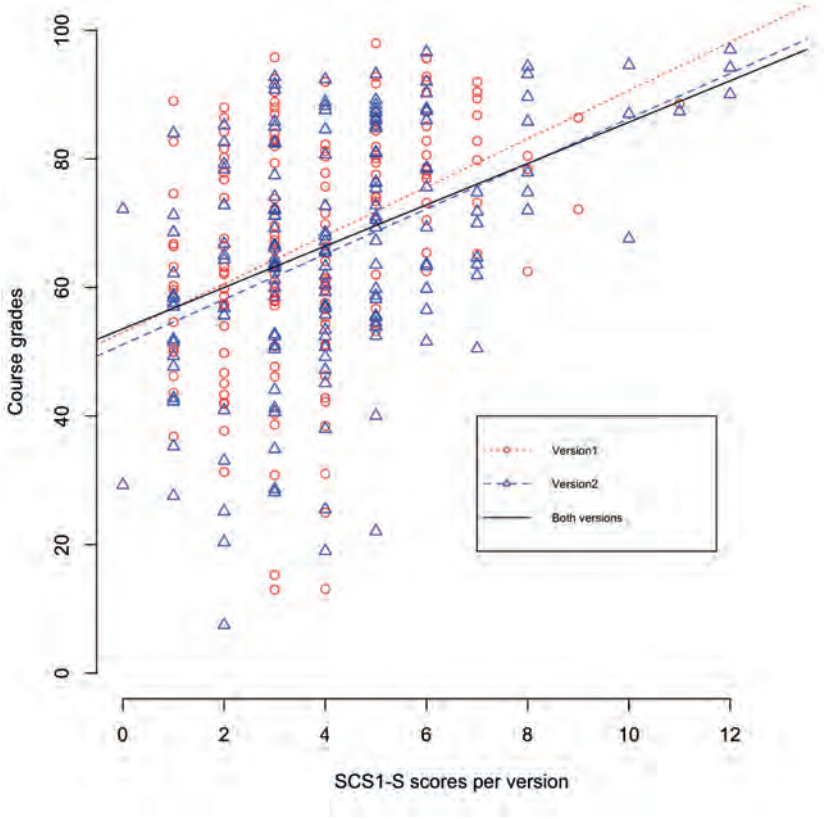


Note: Thirteen is the maximum score on both versions. The horizontal black lines indicate the median score.

2.3.2 External validity

To evaluate the external validity of each version of the SCS1-S, we computed individual correlations between the SCS1-S test scores and the exam grades. When correlating the scores on the SCS1-S with course grades across versions, results showed a moderate and significant correlation between SCS1-S scores and student exam marks ($r(329) = .41$, $p < .001$). For both versions separately we found modest and significant correlations between total scores on the version and exam grades (Version 1: $r(164) = .40$, $p < .001$; Version 2: $r(163) = .44$, $p < .001$). These correlations did not differ significantly between the versions ($z = -.47$, $p = .64$). Figure 2.2 depicts the scatter plots and lines of best fit for these data.

Figure 2.2. Scatter plot between raw SCS1-S scores and student grades on the course exam.



Note: Raw SCS1-S scores are plotted per version with lines of best fit per version and for both versions combined.

2.3.3 Internal validity: Quality of individual items

Below we report the analyses of the quality of the individual items using respectively CTT and IRT.

Classical test theory

Every question was completed by at least 83% of participants, with attempt rates being slightly lower for questions at the end of the versions because not all participants managed to complete the full test (see Table 2.1). We used a binomial model to calculate the critical accuracy rates that had to be reached for each item to conclude that participants performed significantly above chance at an alpha-level of .05 (one-tailed). The probability for the null binomial model was assumed to be .2, since each item had 5 response alternatives. The exact critical value differed for each item depending on the number of participants who answered the question, with the critical values ranging from 25.29% answered correctly for the items with a 100% response rate, to 25.71% answered correctly for Version 1 item 13 which had the lowest number of responses. For items 3, 7, 10 and 13 of Version 1, and items 2, 3 ,6 ,7 and 12 of Version 2 performance was poor and we cannot reject the possibility that participants were responding at or below chance. Performance on all other items was significantly above chance.

Table 2.1. Accuracy and response rates for the two versions of the SCS1-S.

	Version 1				Version 2			
Question number SCS1-S	Original question number on full SCS1	% of participants attempted	Accuracy (all)	Accuracy (attempted only)	Original question number on full SCS1	% of participants attempted	Accuracy (all)	Accuracy (attempted only)
1	1	100	27.54	27.54	3	100	51.76	51.76
2	2	95.21	35.33	37.11	4	100	20.00	20.00*
3	5	99.40	16.77	16.87*	7	95.29	21.76	22.84*
4	6	98.80	40.72	41.21	8	100	35.29	35.29
5	9	97.60	29.34	30.06	10	98.24	49.41	50.30
6	12	94.01	46.11	49.04	11	100	21.18	21.18*
7	13	94.01	20.36	21.66*	16	97.06	22.94	23.64
8	14	93.41	38.92	41.67	19	98.24	44.12	44.91
9	17	91.02	28.74	31.58	20	97.06	19.41	20.00*
10	18	86.23	13.77	15.97*	21	94.12	29.41	31.25
11	22	86.83	31.74	36.55	23	94.12	60.59	64.38
12	25	86.23	31.74	36.80	24	90.59	16.47	18.18*
13	27	83.83	19.76	23.57*	26	89.41	25.29	28.29

Note: For each version the first column shows the percentage of participants who attempted each question. The second column shows the percentages of all participants who answered a question correctly, with unanswered questions counted as incorrect. The third column shows the percentage of correct attempts per question, with unanswered questions treated as missing values. * indicates items where performance is not significantly above chance (Binomial test, $p > .05$).

We also examined the extent to which a participant's accuracy on each item was related to their accuracy on the entire test (see Table 2.2). Higher correlations indicate a stronger relationship between the item and the test total and indicate good internal validity for that item. In these analyses we counted unanswered items as incorrect. That is, all participants are included, regardless of whether or not they attempted the item. On Version 1 items 10 and 13 appeared problematic as they did not show significant correlations with the total score on the version. On Version 2 this was only the case for item 3.

For both versions we also tested the correlation of each individual item with the course grades (Table 2.2). On Version 1 seven questions correlated significantly with the scores on the unit exam (1,2,6,7,8,9 and 13). This was the case for eight questions on Version 2 (1,5,6,7,8,10,11 and 12).

Regarding the difficulty and discriminability of the individual items, Parker et al. (2016) did not report the exact values but rather categorised the items into three categories as seen in Table 2.3. In Table 2.3 we categorised the items using the same scheme as Parker et al. (2016) to allow for direct comparison. We used the original item numbers from the full SCS1 version to allow for easier comparison. Overall, the students in the current study seemed to perform more poorly than participants in the study by Parker et al. (2016). However, in the current study the items were generally more predictive of the final score on the SCS1-S version than in the study by Parker et al. (2016).

Item response theory

Below we first report the results of the verification of IRT assumptions, followed by the results of the IRT model. We used the same steps and models as Xie, Davidson et al. (2019) on our SCS1-S versions.

Table 2.2. Point biserial correlations of individual items with SCS1-S total score and course grade.

	Version 1					Version 2				
Question number SCS1-S	Original question number on full SCS1	Correlation r with SCS1-S total score	p-value	Correlation r with course grade	p-value	Original question number on full SCS1	Correlation r with SCS1-S total score	p-value	Correlation r with course grade	p-value
1	1	.40	<.001***	.28	<.001**	3	.44	<.001***	.37	<.001***
2	2	.35	<.001***	.18	.023*	4	.44	<.001***	.12	.116
3	5	.26	<.001***	.06	.471	7	.10	.211	-.00	.957
4	6	.27	<.001***	.02	.772	8	.40	<.001***	.12	.115
5	9	.33	<.001***	.15	.059	10	.42	<.001***	.19	.013*
6	12	.39	<.001***	.21	.007*	11	.48	<.001***	.25	.001**
7	13	.38	<.001***	.19	.016*	16	.47	<.001***	.20	.010*
8	14	.48	<.001***	.41	<.001**	19	.56	<.001***	.46	<.001***
9	17	.45	<.001***	.21	.006*	20	.27	<.001***	-.10	.199
10	18	.15	.050*	-.14	.065	21	.41	<.001***	.17	.029*
11	22	.32	<.001***	.13	.095	23	.39	<.001***	.23	.003**
12	25	.30	<.001***	.08	.286	24	.35	<.001***	.16	.037*
13	27	.05	.515	-.22	.004*	26	.36	<.001***	.03	.713

Note: *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table 2.3. Crosstabulation of item difficulty and discriminability using the scheme from Parker et al. (2016).

		Difficulty		
		Parker et al.		
		< 50%	50 - 75%	> 75%
Present Study	< 50%	4,5,6,7,8,9,11,12,13,14,16, 17,18,20,21,22,24,25,26,27	1,2,19	-
	50 - 75%	10	3,23	-
	> 75%	-	-	-
		Discriminability		
		Parker et al.		
		Poor (< 0.1)	Fair (0.1 - 0.3)	Good (> 0.3)
Present study	Poor (< 0.1)	27	-	-
	Fair (0.1 - 0.3)	5,18,20	6,7,25	-
	Good (> 0.3)	8,24	4,9,10,11,12,13, 16,17,21,22,26	1,2,3,14,19

Note: Difficulty is defined as the percentage of correct attempts for each question. Discriminability is defined by the point biserial correlation between the score on each item (counting unanswered questions as incorrect) and total SCS1-S score. For easier comparison we used the full SCS1 item numbers. The corresponding SCS1-S item numbers can be found in Table 2.2 for conversion to item numbers per version.

Verifying IRT assumptions

To check whether each of the SCS1-S versions loaded onto one factor we performed a confirmatory factor analysis (CFA) with diagonally weighted least squares (WLSMV) as the estimation method, and the variance of the latent factor constrained to 1. Chi-square tests indicated good model fits for both versions (Version 1: $X2(65) = 66.75$, $p = .42$; Version 2: $X2(65) = 64.35$, $p = .50$). Other goodness of fit statistics also indicate that the two versions of the SCS1-S seem to be measuring one latent variable: Comparative fit index, Version 1 0.96, Version 2 1.0 (above 0.90 is acceptable); root mean square error of approximation, Version 1 0.013, Version 2 < .001 (below .1 is acceptable); standardised root mean square residual, Version 1 .067, Version 2 .066 (acceptable below .8).

To test for the internal-consistency and reliability of the versions we also computed Cronbach's α for each version. For Version 1 we found a low Cronbach's α of .29, and for

Version 2 a higher Cronbach's α of .55. Both of these values are below the recommended level of reliability of .70 that is recommended for early-stage research (Lance et al., 2006). Table 2.4 summarises the sensitivity of α to each item. For each item, we report the change in Cronbach's α if that item was to be removed from the analysis. For Version 1 removal of items 4, 10 and 13 led to improved (higher) Cronbach's α, and on Version 2 this was true only for the removal of item 3. Items for which deletion would lead to a positive change are items on which scores show low consistency with other items on the version.

Fitting the IRT model

Xie, Davidson at al. (2019) tested a Rasch/1Parameter Logistic (PL) model, a 2PL model and a 3PL model. They found that the 2PL model showed the best fit, the Rasch model had the second-best fit. Consequently, in the current study we compared Xie, Davidson et al.'s two best models, a Rasch model (1PL) and a 2PL model. As for Xie, Davidson et al. (2019) the 2PL model provided the best fit for both versions of the SCS1-S, the ANOVA showing a greater Bayesian information criterion (BIC) for the 2PL model than for the Rasch model (Version 1: 2PL *BIC* = 2629, 1 PL *BIC* = 2595; Version 2: 2PL *BIC* = 2629, IPL *BIC* = 2590). We therefore performed the IRT analysis with the 2PL model. In contrast to Xie, Davidson and colleagues (2019), who excluded items 20, 24 and 27 of the full SCS1 version after finding that Cronbach's α would improve by deleting the items, we decided to leave all items in for the IRT analysis. We did this because we found different items that would have to be removed from those excluded by Xie, Davidson and colleagues (2019), therefore removing those items would make comparison with other studies more difficult. Table 2.4 shows the results of this analysis with the difficulty scores and discrimination scores for each item on each of the versions. Generally, an item with good properties would have a difficulty rating below three, and a discriminability rating above zero (Xie, Davidson et al., 2019). This suggests that for Version 1 item 3 may be a bit too difficult, and items 10, 11 and 13 are problematic discriminators, as for those items participants who got the item correct tend to do badly overall and vice versa. For Version 2, item 3 is a problematic discriminator and item 9 (and probably 12) too difficult.

Table 2.4. Sensitivity of Cronbach's alpha, difficulty and discriminability for each item.

	Version 1					Version 2				
Question number SCS1-S	Original question number full SCS1	Cronbach's α after deleting item	Change in Cronbach's α	Difficulty	Discriminability	Original question number full SCS1	Cronbach's α after deleting item	Change in Cronbach's α	Difficulty	Discriminability
1	1	.24	-.05	2.42	0.42	3	.52	-.03	-.13	0.63
2	2	.27	-.02	1.17	0.55	4	.52	-.03	1.65	0.99
3	5	.28	-.01	5.49	0.30	7	.59	+.04	-4.03	-0.32
4	6	.31	+.02	1.28	0.30	8	.53	-.02	0.82	0.83
5	9	.27	-.02	1.84	0.51	10	.53	-.02	0.04	0.59
6	12	.25	-.04	0.21	0.83	11	.51	-.04	1.73	0.87
7	13	.24	-.05	1.58	1.03	16	.51	-.04	1.67	0.82
8	14	.20	-.09	0.49	1.15	19	.49	-.06	0.14	2.89
9	17	.21	-.08	1.33	0.77	20*	.55	0	6.48	0.22
10	18	.31	+.02	-5.92	-0.32	21	.53	-.02	1.16	0.87
11	22	.28	-.01	-65.85	-0.01	23	.54	-.01	-0.61	0.82
12	25	.29	0	2.71	0.29	24*	.53	-.02	3.18	0.54
13	27*	.36	+.07	-2.18	-0.71	26	.54	-.01	2.41	0.47

Note: Sensitivity of Cronbach's alpha to the presence of each item and difficulty and discriminability of items. Values were determined with the 2PL IRT model. *Items that in Xie, Davidson et al. (2019) removal resulted in higher Cronbach's alpha.

2.4 DISCUSSION

The current study aimed to split the SCS1 programming test (Parker et al., 2016) into two parallel short versions to allow for repeated testing and shorter testing times. By doing so, we aimed to improve on the original SCS1 in four ways, by enabling: 1) shorter testing time, 2) the possibility for repeated testing, 3) knowledge of the external validity of the tests and items, and 4) generalization of knowledge on test and item quality to a student population outside Georgia Institute of Technology, where the full SCS1 was primarily developed. Below, we discuss how well we succeeded in reaching each of our aims.

We created two short versions of the SCS1 - the SCS1-Short, Version 1 (SCS1-S1) and Version 2 (SCS1-S2) - balanced for content and difficulty based on the content of the test items, on the Parker et al. (2016) and our own pilot studies. We tested the two SCS1-S versions on a relatively large cohort (N = 354) of university undergraduate programming students. Although the two SCS1-S versions did not show a significant difference in difficulty, Bayesian analyses demonstrated that we also could not convincingly conclude that they were of equal difficulty. Although not significant, scores on Version 2 were numerically higher than on Version 1, and it correlated slightly more strongly with course content (but again not significantly different from Version 1). Moreover, Version 2 showed a higher internal-consistency and reliability than Version 1. In sum, we are unable to conclude that the versions are fully parallel, and Version 2 seems to be of better quality than Version 1.

On average, the SCS1-S scores in our study were lower than the SCS1 scores in the study by Parker et al. (2016) who found an average 35% correct (9.68/27) on the full version of the SCS1. One possibility is that this is due to a difference in course content that the students were exposed to. Both the original FCS1 and the full SCS1 (which was inspired by the FCS1) were developed by researchers from Georgia Institute of Technology, USA and tested primarily on their students. It could therefore be that the questions were more closely related to the contents of their curriculum than to those of other universities, in particular in other countries. That course content may affect performance on SCS1 items is also suggested by the different correlations found for the FCS1 with course scores on the three different programming courses from which Parker et al. (2016) recruited their participants: performance of students who had undertaken MATLAB and Python courses

correlated more strongly with the FCS1, than that of students in a media computation course. Given that the correlations vary considerably by course, it may be that the test is sensitive to the specific material taught. In order to confirm this hypothesis, the SCS1-S (or the whole SCS1) would need to be systematically tested across different programming courses, preferably in a wide variety of contexts.

With regard to external validity, performance on both SCS1-S versions showed a significant correlation with course grade and these correlations did not differ significantly. This suggests that the questions in both versions reflect knowledge that is similar to the knowledge assessed in the unit exam. In addition, the magnitude of these correlations is similar to the correlations of the FCS1 (after which the SCS1 was modelled) with the students undertaking MATLAB and Python courses, but higher than the FCS1 correlation with the students in a media computation course (Parker et al., 2016) (see Introduction for the exact correlations). This suggests that the SCS1-S might reflect programming knowledge to a similar extent as the FCS1.

The SCS1-S2 showed similar internal-consistency (Cronbach's alpha= .55) to the full SCS1 as reported in the post-course test by Parker et al. (2016) (Cronbach's alpha = .59), but lower than for the full SCS1 as reported in the pre-course test by Xie, Davidson et al. (2019) (Cronbach's alpha = .70). To allow us to investigate whether these differences were due to pre-course versus post-course use of the SCS1, B. Xie (personal communication, March 27, 2020) calculated the internal consistency of their SCS1 post-course data samples, which included the Parker et al. (2016) post-test data. Importantly, in these calculations, as for their SCS1 pre-test data (Xie, Davidson et al., 2019), Xie, Davidson et al. (2019) used an exclusion criterion that we and Parker et al. (2016) did not use. Namely, they excluded all participants who answered fewer than 10 questions. When applying their exclusion criterion to the Parker et al. (2016) part of their post-test sample, the internal consistency increased to the same level as for the SCS1 pre-test values reported in Xie, Davidson et al. (2019) (Cronbach's alpha = .74). These results suggest that the differences in internal consistency between our and the other two studies, are not due to the timing of testing, pre- or post- course, but more likely an effect of the applied exclusion criterion. SCS1's internal consistency appears to be higher when only including the students who complete more questions. There are several potential explanations for this effect. It could be that this

excludes the less motivated or less able students, thereby improving the quality of the answers and reducing the guessing rate. It is also possible that fewer missing values per student improves the statistical quality of the sample.

At the individual item level, we did not compare findings with the Xie, Davidson et al. (2019) study because of the differences in their sample (assessed before the start of the course; exclusion of students who answered fewer than 10 questions). Compared to Parker et al. (2016), the items were generally more difficult for participants in the current study. However, for our sample, many items had better levels of discrimination than they did for Parker at al.'s (2016) sample. A factor contributing to these differences could be that we allowed students to skip questions while Parker et al. (2016) did not, which may have reduced the guessing rate in our data. Of the 27 original items in the full SCS1, if one were to keep only the items that, in our study: 1) were answered correctly at levels greater than chance, 2) showed sufficient discriminability, and 3) contributed to stronger internal-consistency and reliability, 12 items would have to be excluded, and 14 items retained – retaining those with original SCS1 numbers 1-3,8-10, 12, 14, 17, 19, 21, 23, 25 and 26. These good quality items cover all topics and question types, except for the topics "function parameters" and "function return values". It is of note that function-use was not covered very extensively in the course undertaken by our sample. Therefore, it might be that those items showed poorer quality in the current study, because those topics received relatively less coverage than in the Georgia Institute of Technology courses. This again suggests that the SCS1 questions are sensitive to course content, and that these items may show better quality for students following a different course. Therefore, until this hypothesis has been proven to be incorrect, we recommend keeping items covering all topics in the tests to ensure content validity, even if this means potentially sacrificing some test quality.

2.4.1 Future research

Given that Versions 1 and 2 of the SCS1-S were not found to be parallel, there remains need for two parallel short versions. Future studies could address this in three ways. Firstly, it is possible that SCS1-S1 might show better qualities in a course with different content. Future studies could test this by administering both short versions to different student populations. Secondly, future studies could use the current and past findings on item

quality to attempt to create different subsets of the items to try and achieve better quality parallel versions, while ensuring that question content remains balanced. Lastly, future studies could attempt to create new items. For example, by selecting the best performing current items and creating items that test the same skill with a different question. The new items would then need to be validated with several student populations.

Additionally, combined with the results from previous studies, our results suggest that item and test quality are dependent on course content. Future studies could further investigate these effects by using the SCS1 or SCS1-S2 after a variety of different programming courses. Meanwhile, researchers who are looking to use the SCS1 or the SCS1-S2 in their studies need to be aware of these dependencies and are advised to look closely at how their course content relates to the SCS1 and SCS1-S programming items.

2.4.2 Conclusion

This study evaluated the effectiveness of two novel half-versions of an existing hour-long test of programming skill (the SCS1). The SCS1-S2 performed similarly to the full SCS1 in Parker et al. (2016), suggesting that it can be considered a validated short version of the SCS1, allowing researchers the option of a test that can be completed in 30 rather than 60 minutes. This will make the test more practical for use in a wider variety of studies and contexts, expanding the opportunities for use of this tool.

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)

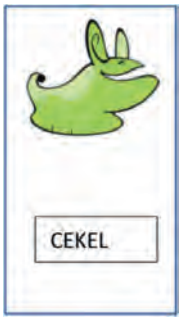
c) T1-((6.5-1)(T1-T2)/(4-1))

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.

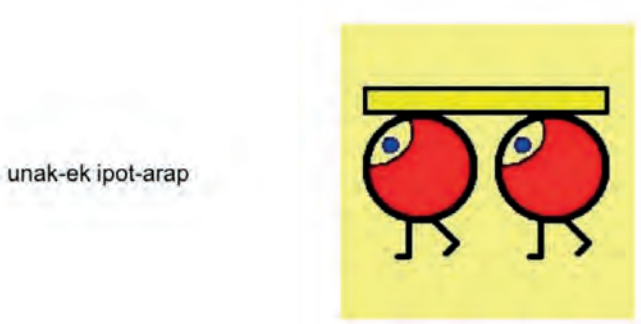


Note: This figure shows one of 20 learning items on the vocabulary learning test. Participants were asked to memorise the names of the creatures. In the immediate and delayed recall stages they were asked to drag and drop the correct names under the corresponding pictures.

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.

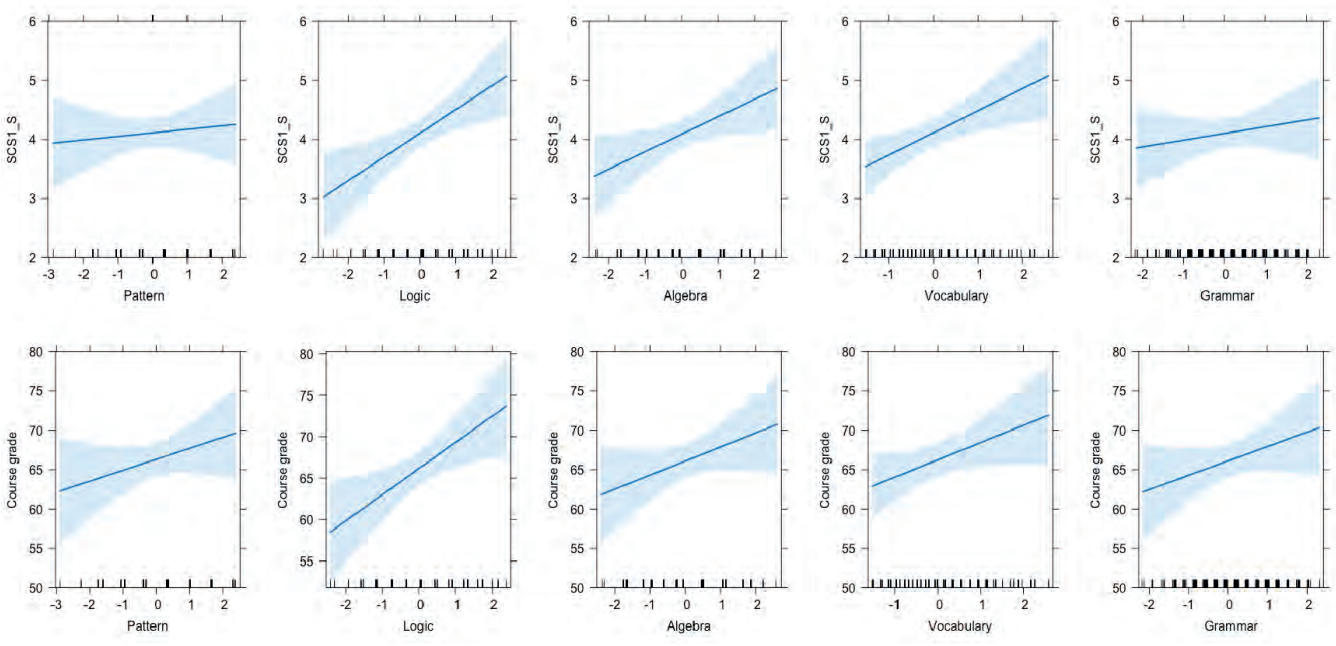
Table 3.1. Predictors of score on the SCS1-S programming test and course grade at the end of the semester.

		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.

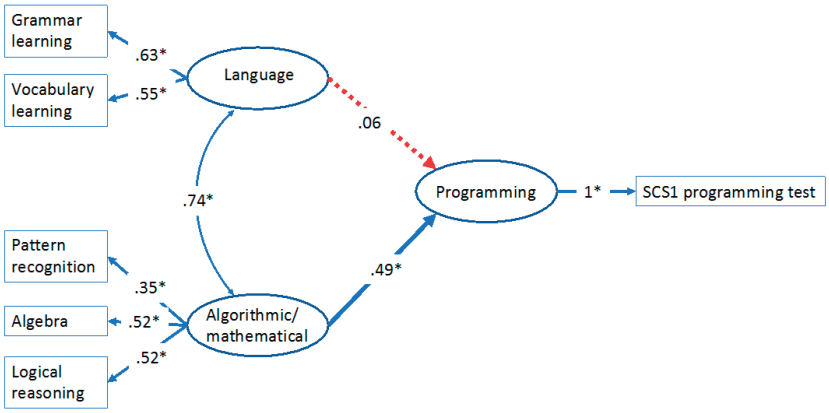
3.3.4 Structural equation modelling

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

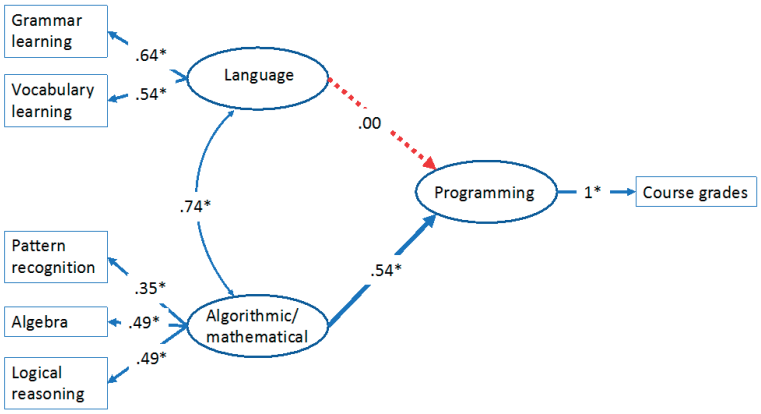
the latent variables for language and algorithmic/mathematical skill, both for the model with the SCS1 (β = .736, *SE* = .108, *z* = 6.850, *p* < .001) and for the model with the course grades (β = .738, *SE* = .113, *z* = 6.507, *p* < .001). Full tested models with estimates, including covariance can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

Chapter 4

Autistic traits and programming learning outcomes in an introductory computing course[1]

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

4.1 INTRODUCTION

Currently, we rely on individuals with programming skills to deliver and maintain a wide range of software-based tools and services, such as apps, websites, games, data analysis tools and online work and teaching environments. In tandem, the number of individuals pursuing training in programming and software design has been increasing as demand grows for these skills (European Commission, 2018). However, as in all areas of education and training, some individuals are more successful than others in the pursuit to become proficient programmers. Researchers have been searching for ways to predict which students are most likely to succeed, as well as those who may have difficulty in acquiring this skill (Wray, 2007). This knowledge could be used to inform career counselling by identifying students best suited to these courses and associated professions or to identify those that may need additional support. Some studies have shown that general intelligence and some specific cognitive skills (e.g., mathematical skills, working memory, logical reasoning etc.) are important when learning to program, and therefore predict programming skills (Guzdial & du Boulay, 2019; Pena & Tirre, 1992; Shute, 1991; Tirre & Pena, 1993; Webb, 1985). In previous work we found that logical reasoning, algebra and vocabulary learning skills were predictors of programming skill (Chapter 3), highlighting that individual differences in neurocognitive profiles can predict education and vocational outcomes. Since the 1980s, whilst recruiters and organisational psychologists have mostly focussed on the role of personality traits in workplace performance and learning (Barrick et al., 2001), there is now growing recognition that other individual differences – including unique neurocognitive profiles – may help better predict programming skill outcomes (Baren-Cohen, 2001; Catherine & Wheeler, 1994; Focquaert et al., 2007; Golding et al., 2006; Wray, 2007). Indeed, the recent neurodiversity movement has highlighted that all individuals differ in their cognitive profiles and neural make-up which gives rise to unique strengths, and this can be more pronounced in portions of the population with neurodevelopmental or psychiatric conditions (e.g., autism, schizophrenia; den Houting, 2019).

Autism is a neurodevelopmental condition characterised by differences and difficulties in social interactions and communication, specific sensory processing sensitivities and tendencies towards repetitive behaviours and restricted interests

(American Psychiatric Association, 2013; Lord et al., 2000). Autism is heterogenous, which means that the way it presents in individuals can vary widely (Lenroot & Yeung, 2013). The 'broader autism phenotype' account suggests that autism is the extreme end of a continuous spectrum, with autistic traits also being present in non-autistic relatives of autistic individuals, and in the general neurotypical population (Baron-Cohen et al., 2001; Landry & Chouinard, 2016). It has been argued that autistic traits may influence a person's interests and talents (Baron-Cohen et al., 2001). For example, Baren-Cohen et al. (2007) showed that there were up to seven times more autistic individuals undertaking mathematics degrees at Cambridge University compared to other degrees, suggesting that autistic individuals have increased interest and aptitude for mathematics. Therefore, it is of interest whether autistic traits also relate to programming aptitude.

Some support for the idea that people with autistic traits are more successful programmers comes from a study by Baron-Cohen et al. (2001), who evaluated autistic traits using a self-evaluation questionnaire - the Autism Spectrum Quotient (AQ; Baron-Cohen et al., 2001). They found that science students, including those in computer science, scored higher on the AQ scale than students in the social sciences and humanities. Furthermore, science students in fields that are more human or life-centred (e.g., biology and medicine), had a lower AQ score than students in more abstract fields of science (e.g., mathematics, computer science and physics). If autistic traits do indeed predict programming skill, it is of interest to explore why this is the case. Baron-Cohen (2012) proposed that genes underlying autism predispose unique neurodevelopmental pathways which lead individuals to process information differently. Specifically, Baron-Cohen suggests that people with autistic traits have a stronger tendency to 'systemise', that is, they have a tendency to process information and understand phenomena by identifying patterns and rules (Baron-Cohen, 2006). It is argued that 'systemisers' find it easier to study systems in nature, such as the laws of physics, or man-made systems (e.g., train schedules). This information processing style has also been proposed to shape the way autistic individuals understand human social behaviour by trying to fit stringent social rules, rather than intuitively and flexibly evaluating social information across contexts (Baron-Cohen, 2012). Alternatively, Baron-Cohen (2012) argues that people with a more empathizing-driven cognitive style intuitively relate to and understand other's emotions, and that this is

negatively associated with autistic traits. For example, if someone is crying – an empathiser may intuitively feel compelled to comfort them, while a systemiser may learn that tears are a sign of sadness, and that the standard social protocol is to offer a tissue. These constructs of empathizing versus systemizing have been argued to (1) characterise autistic individuals and those with autistic traits (high systemizing and low empathizing would indicate high autistic traits) and (2) be associated with other outcomes, including career choice. For example, Focquaert et al. (2007) found that individuals in the sciences possessed a cognitive style that was more systemizing-driven than empathizing-driven, whereas individuals in humanities possessed a cognitive style that was much more empathizing-driven than systemizing-driven. They argue that this relationship reflects a difference in brain structure between empathisers and systemisers that makes them prone to choose a degree that matches their thinking style. A remaining question is whether thinking style only affects degree choice, or whether it also predicts learning success within such degrees.

Wray (2007) tested whether measures of systematizing (SQ) and empathizing (EQ) predict programming skill. Higher scores on these measures indicate a greater natural tendency to systemise or empathise with others. However, Wray found that neither SQ nor EQ alone predicted programming performance, but that the difference between these measures (SQ minus EQ) did. That is, having relatively higher SQ than EQ was associated with greater programming abilities. Nevertheless, the generalisability of these findings is limited by the relatively modest sample size (N = 19) for a study examining individual differences and the absence of females in the sample. Indeed, a later study by Borzovs et al. (2017) failed to replicate Wray's (2007) findings in a larger and more diverse sample (n = 73, 39.7% female). They found no significant correlations between SQ, EQ, nor the difference between SQ and EQ, and programming skill. This could partly have been influenced by the high attrition rates in the study, when they attempted to examine long-term learning outcomes (Borzovs et al., 2017). Together this suggests that SQ and EQ are not reliable predictors of programming skill. However, it is important to recognise that SQ and EQ can only provide an indirect measure of autistic traits, and the sensitivity and validity of this indirect measure for investigation of individual differences within neurodiverse populations remains unclear. As such, SQ and EQ scores may not be sufficiently sensitive to detect a possible relationship between autistic traits and programming skill. This possibility

is supported by a study by Wheelwright et al. (2006), who found that there were only moderate correlations between SQ, EQ and the AQ. Indeed, the idea that autism – and by extension autistic traits - is characterised by reduced abilities or tendencies to empathise with others has been disputed by proponents of the 'double empathy problem' (e.g., Milton et al., 2012; Mitchell et al., 2019). They argue that the evidence for empathizing deficits in autistic individuals is inconsistent – with social challenges better characterised by an incompatibility in empathising between autistic and non-autistic individuals, rather than a reduced capacity to empathise in autism. For this reason, exploring the relationship between autistic traits and programming skill acquisition requires a more holistic measurement of autistic traits.

Critically, no previous study has explicitly examined the relationship between autistic traits and programming skill. Therefore, the aim of the current study was to test whether autistic traits, when measured using the AQ, predict programming skill at the end of a programming course. We define programming skill based on both test performance during the course and generalised programming skill at the end of the course assessed by an independent measure of programming skill (Parker et al., 2016). First, we examined how AQ scores in our student sample compared to the general population. Then we investigated the predictive value of the AQ for course-related and generalised programming skill. We hypothesised that higher autistic traits at the start of the semester would be associated with better programming skill on the course assessments as well as better generalised programming skill at the end of the semester.

We also addressed two additional exploratory questions. Firstly, based on previous research, it remains unclear which autistic traits predict programming skill. To explore whether there are specific domains of autistic traits that best characterise this relationship, we analysed whether any of the individual AQ subscales predicted programming skill (see Table 4.1, below). Significant subscale effect(s) may elucidate the specific cognitive features which drive any observed relationship between autistic traits and programming outcomes.

Secondly, we explored the relationship between autistic traits and cognitive skills related to programming skill. Even if autistic traits do not directly predict programming skill, it may be that the cognitive skills learners rely on as programmers vary depending on their

AQ score. Therefore, we tested whether autistic traits exhibited any relationships with the cognitive skills that play a role when learning to program (i.e., logical reasoning, pattern recognition, algebra, vocabulary learning, grammar learning). For this exploratory analysis, we had no a priori predictions.

4.2 METHODS

4.2.1 Ethics statement

The protocol for the current study received ethical approval from the Macquarie University Human Research Ethics Committee (Reference number: 5201800224). We followed the approved protocol where all students on a programming course received an information form at the start of the Qualtrics survey which gave them the choice to consent for their data to be used for research. Only students who consented for their data to be used were included in the current study.

4.2.2 Participants

Students enrolled in an undergraduate "Introduction to Programming" course at Macquarie University (COMP115) completed a testing session as part of a mandatory tutorial in the first and the final weeks of their 13-week course. Of the 838 students, 344 consented to their data being used in the current study. For the majority of students, this course was part of their mandatory study program. Most students were enrolled in Engineering or Information Technology degrees (67%), but there were also students from a wide variety of other Science and Arts majors, ranging from science to society, history and languages. Participants were excluded if they indicated, in a post-test probe questionnaire, that they had cheated or had not seriously attempted to answer the questions or if they self-reported a less than "Good" level of English on a 5-point rating scale from 1 (minimal) to 5 (native; 62 participants; see OSF link for details of the post-test probe questionnaire: https://osf.io/5d4s6/?view _only=0d88ef3d4da346779f82d20aa4f6df72). Thus, the results reported here are from the remaining 282 participants (49 female, 204 male, 2 other, 27 no gender given; mean age 19.32 years, *SD* = 3.09).

4.2.3 Materials

The results reported here are part of a larger study examining various aspects of cognition in relation to programming skill (see Chapter 3). For the current study, students completed a demographics questionnaire, the Autism Spectrum Quotient, five tests of cognitive skill (logical reasoning, pattern recognition, algebra, vocabulary learning, grammar learning), the Second Computer Science 1 Short (SCS1-S) programming test and we obtained their grades for the course. In addition, as part of the larger study they also completed a 'Sense of Agency' measure (Polito et al., 2013), however, these data are not reported here. All tests were presented in a Qualtrics survey, see Procedure for details.

Demographics

We included two questionnaires to obtain background information and demographics. At the start of the course we collected demographic information including age, gender, degree major, level of English, knowledge of programming languages and previous programming experience. At the end of the course we asked the students about their attendance and time spent on the course. We also asked whether they had cheated in any way whilst completing the tests.

Autism Spectrum Quotient

Students completed the full 50 item version of the Autism Spectrum Quotient (AQ; Baron-Cohen et al., 2001). Participants were asked to choose the answer option that most closely indicated how much they agreed with the statement. The AQ consists of five subscales, each consisting of 10 questions: Social skill, Attention switching; Attention to detail; Communication; and Imagination. Table 4.1 shows one example item per subscale. For each item students selected one response that best described how strongly each item applied to them. The response options were: Definitely agree - Slightly agree - Slightly disagree - Definitely disagree. Half of the items were reverse scored. For each item one point would be counted if participants gave one of the two answers in accordance with autistic traits (e.g., for the item "I prefer to do things the same way over and over again." one point would be scored if the participant answered Definitely agree or Slightly agree, and zero points

would be scored if the participant replied Slightly disagree or Definitely disagree. Therefore, the possible range of scores for this measure was 0-50, with higher scores indicative of more autistic traits. A profile of high autistic traits would be low social skill, low attention switching, high attention to detail, low communication, and low imagination.

Table 4.1. Example items for each subscale of the AQ.

Subscale	Example question
Social skill	*I find it hard to make new friends.*
Attention switching	*I prefer to do things the same way over and over again.*
Attention to detail	*I often notice small sounds when others do not.*
Communication	*Other people frequently tell me that what I've said is impolite, even though I think it is polite.*
Imagination	*I don't particularly enjoy reading fiction.*

Note: These examples are all worded so that they produce an "agree" response for high autistic traits. Approximately half of the items were worded in the opposite way, where a "disagree" response indicated high autistic traits. Those items were reversed scored (e.g., "I prefer to do things with others rather than on my own.").

Cognitive skills tests[1]

Logical reasoning. We used the syllogism test described by Handley et al. (2002). Each item was a syllogism of the form "If it is a triangle then it is yellow. It is yellow. It is a triangle." Participants were asked to evaluate whether the final statement followed logically and with certainty from the previous statements. The test had 16 items and two parallel versions. 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 a rectangle then it is not pink".). Participants were randomly assigned to complete Version 1 or Version 2

[1] Full tests, descriptive results for the tests, and correlations between tests can be found on the OSF: https://osf.io/5d4s6/

depending on their student identification number. They had five minutes to complete all items.

Pattern recognition. Pattern recognition skill was assessed with Part 1 Number Series from the Programming Aptitude Test from IBM (IBM, 1968) [2]. In each question the participant was presented with a series of six numbers and had to determine what the pattern of the sequence was, and then select the next correct number in the sequence from five alternatives (e.g., question: 3 6 9 12 15 18, answer options: 19 20 **21** 22 23). We split the test into two parallel versions, alternating even and uneven item numbers in each version (e.g., Version 1 included items 1,4, 5, 8… etc…) to ensure equal difficulty. Participants were randomly assigned to complete Version 1 or Version 2 depending on their student identification number. Each version consisted of 13 items and participants had five minutes to complete as many as possible.

Algebra. Algebra skill was measured with an adapted version of Part 3 from the Arithmetic Reasoning subtest of the Programming Aptitude Test from IBM (IBM, 1968)[2]. In the original version, the test presented numerical mathematics word-problems with five alternative answers. We changed the questions so that each question was followed by four formulas (equations) as answer options. Participants were asked to select the formula that would produce the correct answer. This enabled measurement of abstract and mathematical reasoning rather than arithmetic. The answer equation options were specifically created for this study. We split this test into two parallel versions, with alternating even and uneven question numbers in each version. Participants were randomly assigned to complete Version 1 or Version 2 first depending on their student identification number. They had 15 minutes to complete as many questions as possible.

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

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))

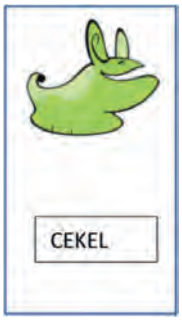
b) (4-1)(T1-T2)/(6.5-1)

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

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

Vocabulary learning. This test was based on the vocabulary learning subtest of the Language Learning Aptitude for MA students (LLAMA; Rogers et al., 2017). Participants learned the names of a series of creatures and were told that they would be tested on them later. They were instructed not to take any notes during the test. Participants were presented with 20 pictures of novel creatures (selected from Romanova, 2015) paired with 20 novel words (e.g., CEKEL, as shown in Figure 4.1) simultaneously on the screen and were given 2.5 minutes to memorise them. They then underwent two testing sessions – the first immediately after the learning phase. In the testing phase, all the creatures and names were displayed on the screen and participants were asked to drag and drop the names under the correct pictures within 3.5 minutes. Approximately 30 minutes later, at the end of the experimental session, they were tested again to assess delayed recall. We devised two parallel versions of the test using different creatures and novel words in each. Participants were randomly assigned to complete Version 1 or Version 2 depending on their student identification number.

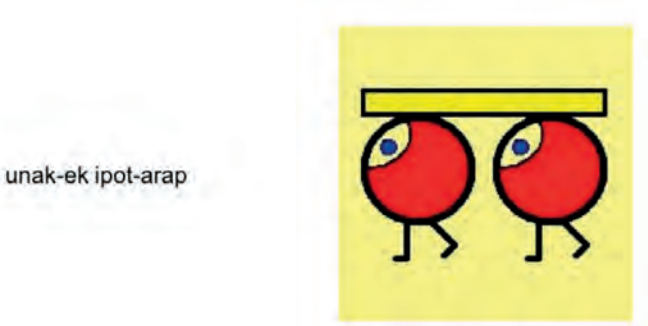
Figure 4.1. Example of an item in the vocabulary learning test.



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. This test was based on the grammar learning subtest of the LLAMA language aptitude test (Rogers et al., 2017) and Part 4 of the Pimsleur Language Aptitude Battery (Pimsleur et al., 2004). Participants were presented with three example blocks, each containing three example picture descriptions written in a novel, artificial language with simple grammatical rules (i.e., nine examples in total). One example sentence is “unak-ek ipot-arap”, which describes two red circle creatures walking underneath a rectangle as shown in Figure 4.2. The examples were then followed by 20 questions in which participants were asked to select the grammatically correct description of a picture from amongst 4 sentences in the novel language. Participants were free to scroll back and forth through all questions and examples. Students were given 8 minutes to complete the test.

Figure 4.2. Example of a learning item from the grammar learning test.



Note: This figure shows a learning example from the grammar learning test. The sentence on the left describes 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.

Outcome measures

Shortened version of the Second Computer Science 1 (SCS1-S). This test was based on the SCS1 (Parker et al, 2016). We used a fully computer-based version, that was split into two parallel versions based on the difficulty of the questions as reported by Parker et al. (2016) and our own pilot experiments (see Chapter 2 for details of the psychometric properties of each subtest). Participants were randomly assigned to complete Version 1 or Version 2, depending on their student identification number. The SCS1 uses an artificial programming language invented by the test developers. Participants were instructed to use a pseudocode guide which provided them with information about the syntax and features of the programming language. This guide could be accessed in a separate browser window by clicking a button in the Qualtrics survey. Participants had 30 minutes to complete as many questions as possible. As the two versions were not of equal difficulty (see Chapter 2), we standardised the scores for each version and used these in the analysis.

Student grades. We used the students' final grades on the main course assessments of their university undergraduate programming course. The main assessments consisted of five module tests each with open questions in which students were asked to answer

conceptual questions or to solve small programming problems. Each of the module tests could be attempted three times on three different occasions throughout the semester, as well as during the final exam testing session, which took place two weeks after the testing session with the SCS1-S. The student's highest score on each module was used to calculate the total grade. The five module test topics were: variables & conditionals, loops, functions, arrays & strings, and program design & problem solving. For more information see the Unit Guide in the Cognition of Coding project on the Open Science Framework (https://osf.io/5d4s6/?view_only=0d88ef3d4da346779f82d20aa4f6df72). For our analysis, we used the raw module test scores from each student's best attempt averaged over the subtopics.

4.2.4 Procedure

We presented all tests in Qualtrics. Students were given a link to the Qualtrics survey during their first and last tutorial of the course and completed the tests individually. During each test participants could scroll back and forth through the questions and, where appropriate, 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 vocabulary learning.

The testing sessions were led by the regular course tutors. Tutors briefly introduced the study, after which students followed the instructions given in the Qualtrics program. Participants were informed that the study was being conducted to see how students learn computer programming, and which skills and personality traits may be important in that process. The students were not given any specific information about what the tests were meant to be measuring nor what the expectations of the study were.

Students were instructed to complete the tests individually at their own pace in the online Qualtrics system. All tests had a time limit that resulted in the survey automatically moving on to the next test if the student had not completed within a set time. In order to prevent students from skipping through all tests without attempting them, the cognitive tests and the AQ were presented in such a way that participants could not move on to the next test for at least one minute. For the programming test students could only move on after 5 minutes. For each test, instructions were provided on a separate page of the survey

that was displayed for 20 seconds before the student could progress. The order of tests for all participants in the first session was: Vocabulary learning including immediate recall; pattern recognition; algebra; logical reasoning, vocabulary delayed recall; grammar learning; and demographic questionnaire. In the second session participants only completed the short form of the SCS1 and a demographic questionnaire. Session 1 took approximately one hour and Session 2 about 30 minutes. Testing sessions took place in the first and last weeks of the semester (i.e., 12 weeks apart). Students who could not attend these tutorials (3% of participants) were allowed to complete the tests at home.

4.2.5 Analysis

Analyses were performed according to a predetermined analysis plan (https://osf.io/n836g). Participants were only included if they attempted all cognitive tests, the SCS1-S and completed the AQ (N = 223). To compare our sample to the average AQ scores in the general population, we computed the average AQ score of our sample, as well as the average AQ score by gender. We compared this to that of the general population, as reported by Ruzich et al. (2015), using independent samples t-tests. To determine whether autistic traits (AQ total score) predicted generalised programming skill (scores on the SCS1-S) and programming skill in the course (final course grade), we used linear regression models. To investigate whether these effects were influenced by previous programming experience we also ran the regression models while controlling for whether or not this was the participant's first programming experience. To examine whether specific features of the AQ scale were related to programming skill we performed an exploratory analysis where we ran two linear regression models examining whether the AQ subscales predicted SCS1-S or final grade.

To examine whether AQ related to underlying cognitive skills, we performed an exploratory analysis for which we computed correlations between the AQ total score and the various cognitive skills (i.e., pattern recognition, logical reasoning, algebra, vocabulary learning and grammar learning). We only included delayed recall in the analyses because immediate and delayed recall of the vocabulary learning test were very highly correlated. For all analyses we used both Null Hypothesis Significance Testing and Bayesian statistics to test support for the null versus the alternative hypotheses. Bayes factors (BF_{10}) between 0

and 0.333 show support for the null hypothesis, with lower values showing stronger support. Bayes factors between 0.333 and 3 are considered inconclusive. Bayes factors above 3 show support for the alternative hypothesis, with higher values showing stronger support (Rouder et al., 2009).

4.3 RESULTS

The mean score on the AQ in the current sample was 19.35 (*SD* = 5.80). The means for male and female participants did not differ significantly ($t(67.61) = 0.62$, $p = .54$; Males: mean = 19.44 (*SD* = 5.85), Females: 18.84 (*SD* = 5.69)) and the Bayes factor ($BF_{10} = 0.214$) showed moderate support for the null hypothesis, together suggesting that AQ scores did not differ by gender. The mean AQ score in the current sample was significantly higher than the mean of 16.94 (*SD* = 5.59) in the general population from a systematic review of the literature by Ruzich et al. (2015) ($t(5152) = 6.26$, $p < 0.001$).

AQ scores at the start of the semester did not predict programming skill at the end of the semester on either the course grade (Linear Regression: $\beta = .070$, $p = .301$, $BF_{10} = 0.243$) or the SCS1 ($\beta = .050$, $p = .454$, $BF_{10} = 0.190$). This pattern did not change after controlling for whether this was the first programming experience (SCS1: $\beta = .050$, $p = .438$, $BF_{10} = 0.214$[3]; course grade: $\beta = .053$, $p = .418$, $BF_{10} = 0.256$). Similarly, none of the individual subscales were significant predictors of programming skill on either the SCS1 or course grades (see Table 4.2). This is also supported by the Bayesian analyses which showed weak support for the null hypothesis for all subscales in the model with the SCS1 scores (all $BF_{10} = 0.250$, suggesting 4 times as much evidence for the null hypothesis), and are inconclusive for all subscales in the model with course grades ($BF_{10} > 0.333$ and < 0.8, favouring the null by 1.25 to 3 times).

[3] In Chapter 2, Version 2 of the SCS1-S was found to have higher internal-consistency and reliability than Version 1. When preforming this regression analysis with only participants who were assigned Version 2, we found that autistic traits did predict performance ($\beta = .220$, $p = .015$). This suggests that the greater reliability of this version may have increased the ability to detect a relationship between autistic traits and programming skill.

Table 4.2. AQ subscales as the predictors of SCS1-Short scores and course grades.

	SCS1					Course grade				
	β	*SE*	*T*	*p*	BF_{10}	β	*SE*	*T*	*p*	BF_{10}
Social skill	.053	.093	0.569	.570	0.250⁻⁻	.131	.093	1.405	.161	0.800
Attention switching	.027	.076	0.351	.726	0.250⁻⁻	.069	.076	0.911	.363	0.471
Attention to detail	.040	.069	0.574	.567	0.250⁻⁻	-.005	.069	-0.078	.938	0.333
Communication	-.013	.092	-0.147	.883	0.250⁻⁻	-.043	.092	-0.465	.643	0.364
Imagination	-.006	.070	-0.090	.929	0.250⁻⁻	-.083	.070	-1.175	.241	0.615

Note: Estimates and *p*-values for the effects of the different AQ subscales in the regression models on SCS1-S scores and course grades. All scores were standardised. * indicates $p < .05$, ** indicates $p < .01$, *** indicates $p < .001$; ⁺indicates $BF_{10} > 3$; ⁻⁻indicates $BF_{10} < .333$.

There were no significant correlations between AQ and any of the measured cognitive skills (see Table 4.3). The Bayes factors show moderate evidence for the null hypothesis for pattern recognition and grammar learning, and weak evidence for the null hypothesis for algebra and logical reasoning. The Bayes factor for vocabulary learning is inconclusive.

Table 4.3. Correlations between AQ score and scores on the cognitive tests.

	Correlation (*r*)	*p*-value	BF_{10}
Pattern recognition	.010	.883	.085⁻⁻
Algebra	.100	.137	.225⁻⁻
Logical reasoning	.127	.058	.204⁻⁻
Vocabulary learning	-.131	.050	.561
Grammar learning	-.029	.670	.092⁻⁻

Note: * indicates $p < .05$, ** indicates $p < .01$, *** indicates $p < .001$; ⁺indicates $BF_{10} > 3$; ⁻⁻ indicates $BF_{10} < .333$.

4.4 DISCUSSION

This study explored the relationship between autistic traits and programming skill in beginner programmers. This knowledge could be useful in identifying students best suited to learning programming. To this end, we measured autistic traits at the start of a 12-week beginner programming course for undergraduate students and examined whether they predicted programming skill acquisition. We also examined whether autistic traits were related to specific cognitive skills at the start of the course (i.e., pattern recognition; algebra; logical reasoning; grammar learning; vocabulary learning).

We found that the students in the current study scored higher on autistic traits than the general population (Ruzich et al. 2015). However, overall autistic traits did not predict programming skill at the end of the course, even when we controlled for previous programming experience. Similarly, no individual subscale predicted programming skill, nor were there correlations between autistic traits and the cognitive skills that may underpin programming.

One possible explanation for these results is that one overall measure of autistic traits (i.e., total AQ scores) may not predict programming skill because only some autistic traits support the successful acquisition of programming. In the context of previous research looking at the Systemizing Quotient (SQ) and the Empathy Quotient (EQ; Wray, 2007; Borzovs et al., 2017), it may be the case that AQ items which load on systemizing behaviours (e.g., items relating to attention to detail) and empathizing behaviours (e.g., items relating to social skill) have the most predictive power for future programming skill compared to those items that capture other autism characteristics. Examining the predictive power of the individual subscales could be one way to explore this, however, this approach is limited given that the current subscales show low test-retest reliability (Austin, 2005; Hurst et al. 2007; Stevenson & Hart, 2017) and questionable construct validity (Austin, 2005; Hoekstra et al., 2008; Hurst et al. 2007; Stevenson & Hart, 2017). Due to these limitations, the subscales are unlikely to have the sensitivity to identify relationships between specific autistic traits and programming skill acquisition. To further tease apart these relationships, a more extensive evaluation of autistic traits is required, so that the predictive value of each specific trait can be examined reliably and validly. The Subthreshold Autistic Traits Questionnaire (SATQ; Kanne et al., 2011) is a potentially suitable instrument

for such an analysis. The SATQ was also designed to measure autistic traits but it captures a broader range of autism characteristics than the AQ (Kanne et al., 2011; Nishiyama et al., 2014). It has also been suggested that the SATQ is more sensitive to features of the female autism phenotype given that the AQ is argued to be biased towards the male autism phenotype (Murray et al., 2017; Ruzich et al., 2015). Therefore, validating the current findings with the SATQ could provide evidence for their generalisability.

An alternative explanation of why AQ scores did not predict programming skill acquisition is that autistic traits may be a better predictor of people's 'fit' or preference to pursue a career in programming, rather than their ability to learn and acquire programming skill per se. Coles and Phalp (2016) also suggested this possibility following findings that the difference between SQ and EQ was related to degree choice, but not to programming skill when assessed using course grades. In the current study, the participant population did score above average on autistic traits. Given that, from a theoretical perspective, autistic traits are presumably stable dispositional characteristics, it is possible that these traits made students more likely to pursue training in this domain. The fact that autistic traits did not correlate with the amount of previous programming experience (r = -.087, p = .145) may simply be due to a limited opportunity for these students to gain such experience. In addition, for many students the current course was part of their mandatory study program. Therefore, the higher AQ scores in our sample may be due to people choosing engineering or mathematics majors that have programming prerequisites, rather than a specific choice to learn about programming, or to pursue a career path that involves programming. To disentangle this relationship, future studies should examine autistic traits across diverse groups of novice programmers within student populations and compare those who have enrolled in degrees which do and do not involve programming coursework. One interesting comparison group would be arts and humanity students who have to take programming subjects (e.g., for data analyses). If autistic traits were found to be predictive of course choice, and possibly course enjoyment, the AQ as a questionnaire could be used by individuals and counsellors to assist informed decisions about career pathways and associated vocational training and education.

Finally, it is possible that autistic traits do not predict programming skill in the current study because programming skill was operationalised as performance under

academic assessment conditions. These assessments may be confounded with stress and anxiety and may therefore underestimate the degree to which someone learned (and to a variable extent across participants). If success were to be more broadly operationalised using long-term criteria such as employer and employee satisfaction, different results may be found. In this, less immediate but more ecologically valid, context, the personality traits associated with autism (e.g., satisfaction derived from completing repetitive and specialised tasks) may lead employees to be more focused, and motivated – which leads to better long-term performance outcomes. An interesting future direction would be to measure the autistic traits of programmers in the workplace and examine the relationship of this with subjective measures of performance and satisfaction from the perspective of programmers, employers and co-workers.

In sum, the current study suggests that autistic traits are not as reliably related to programming skill as stereotypes suggest. However, there may be specific autistic traits or cognitive strengths associated with autism that have a more direct relationship with programming skill. For example, the cognitive features of autism may relate to programming skill, while personality or preference aspects of autistic traits relate to career choice and workplace performance. These possibilities require further empirical investigation using more specific measures and more diverse samples. There is also a need to explore other outcome measures of 'programming skill' which should include measures of specific skills but also long-term measures of workplace success from both employee and employer perspectives. This line of enquiry will provide more information on the relationship between autism and autistic traits, job performance and satisfaction. This can lead to a wide range of benefits in society, for example by informing careers counselling, destigmatising autism and encouraging the employment of autistic people.

Chapter 5

Processing of violations in human and computer languages: An EEG study

5.1 INTRODUCTION

The role that natural human language plays when learning to computer program is still to be determined. Some researchers have argued that programming is, at least partly, a language skill (e.g., Fedorenko et al., 2019; O'Regan, 2012; Paulson, 2007). It has also been suggested that, from a theoretical point of view, there are many similarities between programming and natural language writing. Specifically, it has been argued that both skills rely on specific structure and style rules (Hermans & Aldewereld, 2017) and that these rules might be similar to some extent, especially because modern programming languages were intentionally designed to resemble natural languages (Fedorenko et al., 2019; Paulson, 2007). This natural language-like design of modern programming languages may facilitate transfer between natural language and programming skills, leading to the possibility that natural language skills may be a predictor of programming success. However, other researchers have argued that programming is essentially a problem-solving skill that should be classed as a Science Technology Engineering and Mathematics (STEM) subject (Fedorenko et al., 2019). Understanding how both STEM and language skills relate to programming is important for optimising programming education (Fedorenko et al., 2019).

Experimentally, only a handful of studies have looked into the role of language when learning to program. In an earlier study, we found that vocabulary learning skills predicted programming performance, but grammar learning skills did not (see Chapter 3). A second study on this topic, by Prat et al. (2020), showed that scores on language aptitude tests explained some of the variance in learned programming skill. In addition, three neuro-imaging studies, using functional Magnetic Resonance Imaging (fMRI), have investigated the relationship between programming languages and natural languages. The first, by Siegmund et al. (2014), showed that reading computer code for comprehension activated similar brain areas to silent reading of a natural language. In contrast, Floyd et al. (2017) showed differences in activation patterns between reading of natural and programming languages. However, this difference was only found in beginning programmers. Thus, it is possible that a programming language is processed in a similar way as natural language once a programmer has become sufficiently proficient. In a recent fMRI study Ivanova et al. (2020) found that reading a programming language depended much less on language systems in the brain than reading sentence problems. They argued that reading and

understanding programming languages was more reliant on the multiple demand system, which is the brain network most associated with mathematics. However, their results suggested that although the language system was not activated in the same way for programming languages as for natural languages, parts of the language system were still involved when reading code. They tested two programming languages: Python, which is text-based, and Scratch Junior, which is a more graphical programming language, and therefore is structured less like a natural language. Although they did not directly compare these programming languages, their results suggest that the language system was more involved for the text-based programming language Python, compared to the graphical language Scratch Junior. Therefore, there still seems to be a role for language in programming, and this role appears to be related to the exact nature of the programming language.

In sum, the previous studies suggest that natural language processing is involved when reading programming languages. What remains unclear is which aspects of language relate to programming skills. One can argue that the structural rules in a programming language, referred to as 'programming syntax', have a similar role to grammar in a natural language, as both prescribe the rules that a writer has to follow to create correct and interpretable text or code (Hermans & Aldewereld, 2017). However, in the study reported in Chapter 3, we saw that programming skills were not predicted by artificial grammar learning. This raises the question to what extent learning the syntax of a programming language resembles learning grammar in a natural language. To answer this question, further research into the linguistic properties of programming languages is necessary. Specifically, it is relevant to investigate whether there are fundamental differences between processing programming syntax and natural language grammar in the brain.

5.1.1 Event-Related Potentials (ERP) and language processing

One way to study language processing in natural languages is by analysing brain responses to different kinds of violations (e.g., semantic or morphosyntactic) in written or spoken language (Osterhout & Holcomb, 1995). For natural languages, a considerable amount of research has been performed using such violations with Event-Related Potentials (ERPs; Carreiras & Clifton, 2004). In these studies, the electrical brain responses evoked by these

violations are recorded with electroencephalography (EEG) and compared to those evoked by stimuli without violations (Friederici et al., 2002). Previous studies have shown that different types of violations elicit different types of responses (e.g., Carreiras & Clifton, 2004; Friederici et al., 2002).

For grammatical violations in natural language, most studies have shown two types of brain responses: a left-anterior negativity (LAN), which occurs approximately between 300 and 500ms (Carreiras & Clifton, 2004), and a positive deflection in the signal at approximately 500ms post-onset of the stimulus, which is called the P600 (Carreiras & Clifton, 2004; Friederici et al., 2002; Hagoort et al., 1993; Osterhout & Holcomb, 1992). The LAN effect (when present) is believed to signal recognition of a violation, while the P600 response is usually considered to reflect repair and reanalysis (Friederici et al., 2002). The P600 indicates that when this type of violation occurs, the reader has to exert additional mental effort to repair and/or reanalyse the sentence. Molinaro et al. (2008) suggest that in the first stage of the P600, around 500 to 700ms, the incongruence of the sentence is registered. In the second stage, from 700 to 900ms, the reanalysis of the stimulus occurs. Studies have shown clear P600 effects for, for example, subject-verb violations, where the form of the verb does not match the number or person of the subject in the sentence, for example: *"He are a good swimmer." (e.g., Gouvea et al., 2010; Kaan, 2002; Nevins et al., 2007).

Previous studies have also shown that violations that are not grammatical in nature elicit different responses. Specifically, semantic violations (where the grammar of the sentence is correct, but a word is presented that does not fit in the conceptual context of the sentence; e.g., "The cloud was buried.") have been shown to elicit a negative deflection at approximately 400ms, called the N400 (Friederici, 1995; 2002; Holcomb, 1993; Kutas & Hillyard, 1980). The N400 has also been elicited in non-language contexts, for example, for errors in mathematics (e.g., Jost et al., 2004; Szucs & Csépe, 2004; Wang et al., 2000).

Several studies have shown earlier positive deflections at around 300ms. The interpretation of this effect compared to the P600 effect is still contested. While some studies interpret this effect as an early P600, arguing that it is still a language specific effect (Vissers et al., 2006), others argue that this should be considered as a separate component

(P300; Osterhout, 1999; Osterhout et al., 1996). The P300 is found in response to a wide variety of unexpected, task relevant stimuli, both within and outside language contexts, and is typically seen in the frontocentral and centroparietal regions (Donchin, 1981; Osterhout et al., 1996). The amplitude and the duration of the P300 effect are thought to reflect the amount of attentional resources allocated to a stimulus, with stimuli that demand more attentional resources eliciting longer P300s with larger amplitudes (Suárez-Pellicioni et al., 2013). Coulson et al. (1998) and Sassenhagen and Fiebach (2019) argue that the P600 is part of the same family as the P300. They argue that both the P300 and the P600 reflect a domain-general brain response to salience and are not language specific, supported by Sassenhagen et al.'s (2014) finding that the P600, like the P300, is reaction-time aligned. The P300 has also been found in language paradigms, for example by Osterhout et al. (1996), who found a positive deflection at around 300ms, which they interpreted as a P300 effect, when they presented a word in uppercase while the rest of the sentence was in lowercase. Vissers et al. (2006) found similar effects for orthographic violations (spelling errors) in expected words, which could also have been interpreted as a P300, but they interpreted the effect as an early, language-related P600. Overall, regardless of the name assigned to this early positive deflection, the literature suggests that the early positive response (either a P300 or an early P600) in a language context is related to unexpected formats or spellings, rather than grammatical or semantic violations.

5.1.2 The current study

In the current study, we aimed to test whether syntax errors in a programming language are processed similarly to grammatical violations in a natural language. If they are, then they would be predicted to elicit a P600 effect. We used the programming language Java, as this is a text-based programming language that is widely used and has strict syntax rules, and, thus, allows for unambiguous violations (Gosling et al., 2000). We chose to present a typical violation that is programming specific, and grammatical in nature: a violation in bracket-use in "if" and "while" statements. For example, presenting "while {x==1}" which uses the incorrect type of brackets, while the correct structure would be "while (x==1)".

"If" and "while" statements are very common structures that occur in almost all programming languages. Both structures start a conditional instruction for the execution of

the code that follows. For example, “if (x==1)” means “if variable x is currently equal to 1, execute the following code”. Hence, the code that follows is only executed if variable x is equal to 1. Similarly, “while (x<10)” instructs the computer to execute the following code repeatedly until variable x is no longer smaller than 10. In this structure, variable x will typically change with every execution of the following code, causing it to eventually reach or exceed 10; then the computer will stop running that section of the code.

Although “if” and “while” statements occur in almost every programming language, the exact syntax rules to present such a statement vary across languages. For example, in Java and C++ the correct formulation is “if (x==1)”, while in Python the brackets can be omitted: “if x==1”. The absence or presence of the brackets, therefore, does not change the meaning of the statement. Instead, they are a part of a syntax rule that is specific to each programming language, in which they indicate where the conditions of the “if” or “while” statements start. Although the use of brackets is somewhat flexible across programming languages, within Java the use of round brackets indicates a strict syntax rule. If the use of brackets is incorrect, the code will not run. Consequently, we propose that a bracket in Java has a function akin to functional elements (e.g., auxiliaries or inflectional morphology) in natural language. Furthermore, it is used in a certain form in conjunction with the “if” or “while” statement to create correct syntax. Therefore, we suggest that this kind of violation is of a grammatical nature and hence might be predicted to elicit a similar P600 response to grammatical violations in natural languages.

In order to closely match the type of violation across the languages, we used subject-verb disagreement as the grammatical violation in Dutch (e.g., **Ik is een geweldige leraar*.: “I is a great teacher.”). This type of violation, like the Java bracket violations, is also detected upon presentation of the second word, and involves a similar type of agreement between a function word (the subject in Dutch, and the “while” or “if” functions in Java) and an incorrect inflection of the following word (the verb in Dutch and the bracket in Java).

When considering the language aspects of programming languages, it is important to take into account that a programming language is never a native language. Therefore, Pandža (2016) suggests that, to further understand the role of language in programming, more research should take a second language learning perspective. Consequently, we

added a third condition: subject-verb disagreements in a second language, in this case, English, which our participant population acquired after the critical age of language acquisition (approximately 10 years old). Second language ERP studies have shown that ERP responses to syntactic violations in a second language vary according to proficiency and age of acquisition (Kotz, 2009). Late learners with low proficiency may even show a complete absence of both the LAN and the P600 effect, while more proficient learners may show no LAN effect but still a P600 effect (Weber-Fox & Neville, 1996). Both the LAN and P600 effects can also be delayed and decreased in amplitude, with typically shorter latencies and higher amplitudes for more proficient L2 speakers (Rossi et al., 2006). If the violation that is tested in the second language also exists in the native language, ERP responses for second language speakers resemble those of native speakers more closely in latency (Tokowicz & MacWhinney, 2005). In general, ERP components for second language speakers tend to have a similar scalp distribution as for native speakers, although some studies have found responses to be more bilateral for second language speakers compared to the more left-lateralised effects for native speakers (Van Hell & Tokowicz, 2010).

In sum, our research question was: Is a programming language processed similarly to a natural language? And, as noted above, to address this question we compared bracket violations in Java with subject-verb violations in Dutch (L1) and English (L2).

For Dutch, based on previous studies on subject-verb violations in natural languages, we predicted a P600 type response to the violations (Chen et al., 2007). On the basis of literature on second language processing and our participants' high proficiency in English, we also predicted a P600 of similar scalp distribution, but that this may be delayed (Kotz, 2009; Rossi et al., 2006; Weber-Fox & Neville, 1996).

This is the first study to study ERP responses to syntactic violations in a programming language. Hence, we had no previous research to guide predictions regarding the effect of violations in Java. However, based on the selection of our stimuli, we expected that the Java bracket violations are of a language-like grammatical nature, and thus, would elicit a P600, that is, an effect comparable in latency and distribution to that for Dutch and English. We predicted that this response would be delayed compared to Dutch (L1), and more similar to English (L2), as a programming language is always a non-native language. However, if the

effect for Java differed in its characteristics from those for Dutch and English, we could not rule out that we might observe a different effect. For example, it is possible that the P300, that can be elicited by a spelling error (Vissers et al. 2006) or a deviant capitalisation (Osterhout et al., 1996), could be evoked from the bracket errors. This P300 effect should then be characterised by an earlier latency, possibly with a shorter duration and a more frontal and bilateral distribution than we would expect from a P600. In order to establish the exact nature of the violations, we also examined the topography of the ERP effects. Specifically, we included anteriority (i.e., how frontal the effect is) and hemisphere as variables and studied the scalp distribution of responses over time.

5.2 METHODS

5.2.1 Ethics statement

The protocol for the current study received ethical approval from the Research Ethics Committee at Groningen University (Reference number: 67378826). We followed the approved protocol where all participants received an information sheet at the start of the experiment and gave their informed consent prior to participation. Participants received 15 Euros for their participation.

5.2.2 Participants

We recruited 12 male participants[1] (mean age 26.58, range 18-39) with at least 2 years of programming experience, and at least basic knowledge of programming in the Java programming language specifically. All participants spoke Dutch as their native language. They had all completed Dutch high school English education and thus had at least B1 level of English (Europees Referentiekader Talen, n.d.). The group consisted of six university students and six participants who worked as programmers professionally. All were right-handed as confirmed with the Edinburgh Handedness Questionnaire (Oldfield, 1971).

[1] The current sample size is smaller than intended because testing was cut short due to COVID-19.

5.2.3 Materials

Demographic questionnaire

A questionnaire was administered in which the participants were asked about general demographic details such as age and gender, as well as about their language and programming backgrounds to ensure that participants did not learn English before the age of 10, and thus could not be considered native English speakers, and to ensure that they all had at least basic knowledge of Java programming.

Stimuli

The experiment consisted of three language blocks (Dutch, English and Java), each containing 80 items. The order of language blocks was counterbalanced across participants. The items were equally divided between grammatical (40 stimuli) and ungrammatical (40 stimuli). In order to prevent participants from seeing one and the same item in both its grammatical and ungrammatical form, we created two lists using the Latin Square design. This means that each participant saw 20 grammatical and 20 ungrammatical sentences for each language. This adds up to 120 experimental items per list. Each participant only saw one list, meaning that half of the participants saw the grammatical version of an item, while the other half saw the ungrammatical version of that item. In addition, there were 120 filler items per list, bringing the total number of items to 240 per participant. The fillers were included to prevent participants from developing a strategy where they would stop reading the whole stimulus, and instead would only focus on one specific violation. Examples of target and filler stimuli are provided in Tables 5.1 and 5.2 respectively, and all stimuli can be found in Appendix B.

Programming language stimuli in Java

For the Java stimuli, the grammatical condition for each participant consisted of 20 “if” and “while” statements with the corresponding round brackets (e.g., *if (x>10)*), while the ungrammatical condition consisted of 20 “if” and “while” statements with incorrect curly brackets (e.g., **if {x>10}*) (see Table 5.1). In our pilot questionnaires, proficient Java programmers indicated that they found these types of Java bracket errors easy to

recognise. Each participant was presented with 20 grammatical and 20 ungrammatical fillers. We used two types of fillers. The first type were "else" statements with grammatical or ungrammatical bracket use. In this case, the grammatical brackets were the curly brackets (e.g., *else{x=5}*) while the ungrammatical strings contained round brackets (e.g., **else(x=5)*). This type of filler ensured that participants could not simply conclude that curly brackets were always ungrammatical. The second type of fillers were "if", "while" and "else" statements with grammatical (e.g., *while(x==50); else{x=5}*) or ungrammatical (e.g., **while(x=50), else{x==5}*) expressions inside the brackets (see Table 5.2). These fillers prevented the participants from only paying attention to the brackets and made sure that they also read the content of the brackets.

First language stimuli in Dutch

For the Dutch stimuli, we manipulated subject-verb agreement. The grammatical condition consisted of 20 sentences beginning with a subject pronoun, followed by a verb agreeing in person and number (e.g., *Hij is*: "He is"); the 20 sentences in the ungrammatical condition started with a subject pronoun with which the verb did not agree (e.g., **Hij ben*: "He am") (see Table 5.1). These first two words of the sentences were critical for the experiment and aligned with the experimental stimuli for Java. After those first two words the experimental sentences were completed with no further violations. As fillers, we used 20 grammatical sentences in which the noun agreed with the preceding adjective and article in gender and number. The 20 ungrammatical sentences contained either gender or number disagreement between the adjective and the noun or number disagreement between the article and the noun (see Table 5.2). The violations for the fillers can be recognised in various positions in the sentences, thereby ensuring that participants would read the whole sentence, and not only focus on the second word, the position in which the violations in the experimental stimuli occurred.

Second language stimuli in English

For English, we also manipulated subject-verb agreement. The grammatical condition consisted of 20 sentences beginning with a subject pronoun and verb agreeing in person and number (e.g., *He is*), while the 20 sentences in the ungrammatical condition started

with a subject pronoun and verb disagreeing in person or number (e.g., **He am*) (see Table 5.1). As fillers for the English condition we used 20 grammatical sentences and 20 ungrammatical sentences with incorrect affixation (pluralised uncountable nouns) or the indefinite article with uncountable (mass) nouns (see Table 5.2). Violations for the fillers can be identified in different positions in the sentences, thereby making sure that participants would read the whole sentence, and not only focus on the second word, in which position the violations in the experimental stimuli occurred.

Table 5.1. Examples of the experimental stimuli.

	(1) Java					
Grammatical	While	(	x	>	10	)
Ungrammatical	While	*{	x	>	10	}
	(2) Dutch					
Grammatical	Ik	ben	een	geweldige	docent.	
	I am a great teacher					
Ungrammatical	Ik	*is	een	geweldige	docent.	
	I is a great teacher.					
	(3) English					
Grammatical	We	have	the	nicest	dog.	
Ungrammatical	We	*has	the	nicest	dog.	

Note: Examples of experimental stimuli for Java, Dutch and English. For the full list of stimuli see Appendix B. The words or symbols are spaced out to show how they were presented in fragments. * indicates where the violation occurs.

Table 5.2. Examples of the fillers.

	(1) Java					
Grammatical	Else	{	x	=	5	}
Ungrammatical	Else	*(	x	=	5	}
Grammatical	While	(	x	==	50	)
Ungrammatical	While	(	x	*=	50	)
	(2) Dutch					
Grammatical	De	lange	touwen	hangen	daar.	
	The long ropes hang there.					
Ungrammatical	De	*lang	touwen	hangen	daar.	
	$The_{ART,DEF}$ $long_{ADJ,INDEF}$ $ropes_{N,PL}$ hang there.					
	(3) English					
Grammatical	They	perform	maintenance	on	roads.	
Ungrammatical	They	perform	*maintenances	on	roads.	

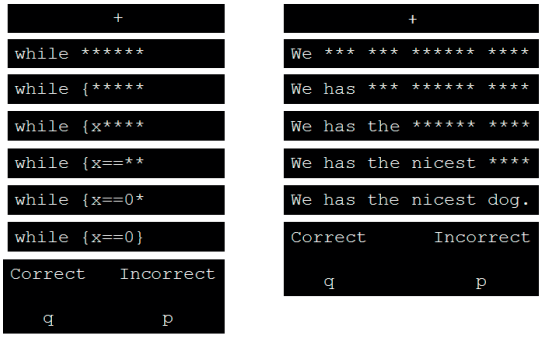
Note: Examples of the different types of fillers for Java, Dutch and English. For all three languages the position of the violation in the sentence/code varied between stimuli. For the full list of fillers see the Appendix B. The words or symbols are spaced out to show how they were presented. * indicated where the violation first occurs.

5.2.4 Procedure

After participants gave informed consent and completed the handedness questionnaire, they were seated at a comfortable distance from the screen (approximately. 80 cm) and the EEG cap was fitted. E-Prime 2.0 (Psychology Software Tools, Inc.) was used as the presentation software. Once the EEG system was fully set up, the experimenter explained that the experiment consisted of three blocks: Dutch, English and Java. The task was to judge a series of sentences or snippets of code for grammaticality using the “q” or “p” key to indicate whether each sentence or snippet had been correct or incorrect. Participants were not told what kind of violations they would encounter but there were five practice items at the start of each block to give them an impression of the experimental stimuli. Instructions were also given on the screen and participants could ask for clarification after the practice items.

Due to the spoken nature of natural languages, people are used to processing sentences serially. Although this is not a common presentation for written language, previous studies have shown that it is possible to present written sentences word by word without readers losing track of the meaning of the sentence (e.g., Carreiras & Clifton, 2004; Friederici et al., 2002). However, programming languages are always presented in written format in such a way that the complete code is visible simultaneously. Programming languages have no spoken form where programmers learn to process code one word or symbol at a time. Therefore, the traditional way of showing stimuli for a reading ERP experiment, with only one word at a time shown in the middle of the screen, would be unnatural and confusing in this condition. To present the programming stimuli in a natural way, while still allowing for accurate timing, for all three languages, we first presented a series of asterisks representing the full length of the stimulus on the screen, and then filled in the words or symbols by replacing the asterisks segment by segment (word by word, or per semantic entity for code; see Figure 5.1 for examples of this method of presentation). Each stimulus started with a fixation cross for 1000ms, followed by a blank screen for 500ms. Then the first segment (word or symbol) appeared with asterisks filling in the space of the following words or symbols for 600ms. Every 600ms another word/symbol was filled in, replacing its asterisk place holder, until the whole sentence or code snippet was on the screen. The sentence was then followed by a blank screen for 500ms and then participants had 3000ms to indicate whether the stimulus was correct or incorrect using the “q” or “p” key. Once a response was given or 3000ms had elapsed, a blank screen appeared for 500ms before the next trial began, starting again with the fixation cross. Stimuli were written in white on a black background (Courier New, 25pt). The assignment of the Correct and Incorrect responses to left or right hands was counterbalanced across participants. The three blocks for the different languages were counterbalanced in order across participants. The task took 30 min on average. After the ERP experiment was completed, participants filled out the demographic questionnaire.

Figure 5.1. Examples of stimulus presentation.



Note: These examples of an ungrammatical stimulus in Java and in English show how each stimulus started with a fixation cross, followed by the sentence or code with the first word written out and the rest of the stimulus length filled in by asterisks. Words or semantic entities then replaced the asterisks one by one, at 600ms intervals. After the full presentation of the stimulus the participant had three seconds to judge whether the previously presented statement had been correct (grammatical) or incorrect (ungrammatical) by pressing the “p” or “q” keys on the keyboard.

5.2.5. EEG Recording and Data processing

The continuous EEG was recorded from 32 Ag/AgCl scalp electrodes (WaveGuard) using the eego system (ANT Neuro Inc, Enschede, The Netherlands). An additional electrode was used (EOG, above the left eye) in order to detect, and allow correction for eye movement artifacts. Impedances were kept below 10 kΩ. Data were acquired at a 500 Hz sampling rate using the common average reference.

The offline processing was carried out in MATLAB using the EEGLAB package. Data were re-referenced to the average of the mastoids. Offline filtering was performed using a band-pass filter (0.1–30 Hz). The data were then segmented into epochs starting 200ms before presentation of the critical word or symbol, until 1200ms post onset. Data were baseline-corrected, with the baseline period of -200-0ms. We then used Independent Component Analysis (ICA) to remove artifacts due to eye movement, muscle activity, heart rate and electrical interference. Only components that were labelled “brain” or “other” by ICA remained. Finally, we used automatic epoch rejection with a threshold of $\pm$ 100 μV to reject epochs that still contained too much noise. In total 6.4% of all trials were excluded

(5.8% for Dutch grammatical, 5.8% for Dutch ungrammatical, 4.2% for English grammatical, 6.7% for English ungrammatical, 9.2% for Java grammatical and 6.7% for Java ungrammatical). There was no difference in the number of retained epochs between conditions ($F(5, 55) = .938$, $p = .464$). Finally, the data were averaged per subject and per condition. Grammaticality judgement accuracy was sufficient for all participants in all conditions (70% correct or higher), so no participants were excluded from any of the analyses. Grammaticality judgements in the current study were used to maintain the participants' attention and to give an indication of the overall recognition of violations, they were not used to exclude data on individual trials.

5.2.6 Analysis

Averaged values (in μV) were extracted per participant, per condition, and per region of interest. Scalp electrodes were divided into 9 regions of interest, each containing either 2 or 3 electrodes: left anterior (F7, F3, FC5), midline anterior (Fz, FC1, FC2), right anterior (F4, F8, FC6), left central (C3, CP5), midline central (Cz, CP1, CP2), right central (C4, CP6), left posterior (P7, P3), midline posterior (Pz, POz), and right posterior (P4, P8). The grand average was computed for each region of interest over the full time-span from 200ms before until 1200ms after presentation of the violation. Topographic maps and ANOVAs were carried out for four time-windows, from 200 to 400ms, 400 to 600ms, 600 to 800ms and 800 to 1000ms.

For the statistical analysis, which was carried out using R, two repeated measures ANOVAs (using the "ez" package in R) were performed for each language and for each of the four time-windows. The first included the lateral regions of interest (Left Anterior, Left Central, Left Posterior, Right Anterior, Right Central and Right Posterior) and had grammaticality (2 levels: grammatical and ungrammatical), anteriority (3 levels: anterior, central, and posterior), and hemisphere (2 levels: left and right hemisphere) as within subject factors. The second repeated measures ANOVA only looked at the regions of interest in the midline (Midline Anterior, Midline Central and Midline Posterior) and had grammaticality (2 levels: grammatical and ungrammatical), and anteriority (3 levels: anterior, central, and posterior) as within subject factors.

Additionally, to analyse whether there were differences in the grammaticality effects across the three languages, we performed similar ANOVAs on the difference waves between the grammatical and ungrammatical levels of each condition. For the lateral regions, the ANOVAs had language (3 levels: Dutch, English and Java), anteriority (3 levels: anterior, central, and posterior), and hemisphere (2 levels: left and right hemisphere), as within subject factors. For the midline regions, the ANOVAs had language (3 levels: Dutch, English and Java), and anteriority (3 levels: anterior, central, and posterior) as within subject factors. The significance level was set to $p < .05$. Post-hoc paired *t*-tests for the individual languages were carried out with those interactions that were significant ($p < .05$) or marginally significant ($.1 > p > .05$), and that included the factor grammaticality or language. As an effect size statistic for the ANOVA analyses, we reported generalised eta squared (η^2G). For the pairwise comparisons, *p*-values were corrected using Bonferroni corrections. If the assumption of sphericity for the ANOVAs was violated at $p < .05$, the Greenhouse and Geisser (1959) correction was applied.

5.3 RESULTS

5.3.1 Accuracy data

Overall accuracy in the grammaticality judgment task was 97.5%. Of the 12 participants, 3 achieved 100% accuracy; the number of errors for the other participants ranged from 1 to 7. Accuracy of grammaticality judgements across languages ranged from 99.58% correct for both grammatical and ungrammatical items in Dutch to 94.58% correct for the ungrammatical items in Java. However, *t*-tests showed that none of these differences in grammaticality judgements across languages reached significance (all *p*-values > .08).

5.3.2 ERP results

For the analysis we compared grammatical and ungrammatical sentences for each of the three languages (Dutch, English and Java). A visual inspection of the waveforms showed a positive effect elicited by ungrammatical stimuli in all three languages. For Dutch (see Figure 5.2), visually, the main deflection of this effect appeared to start at approximately

500ms, lasted until approximately 800ms and occurred mainly in the central and frontal regions. This effect was followed by a smaller ongoing positivity in the posterior regions which was still present at 1000ms.

For the English language (see Figure 5.3), the effect started at approximately 600ms and was broadly distributed, with the strongest initial effect in the frontal left and central regions, lasting until approximately 800ms, and an extended effect in the posterior regions that was still present at 1000ms.

For the programming language Java (see Figure 5.4), the effect started at around 400ms, mainly in the frontal and central areas and lasted until approximately 800ms. There was no extended positive effect after 800ms.

We now discuss the ANOVA results for each language in each of the four time windows. In the text we only report the ANOVA results that were significant or marginally significant. All results, including non-significant results can also be found in Appendix C.

Dutch

In the Dutch language condition, in the time window from **200 to 400ms** there was a significant main effect of grammaticality in the lateral regions ($F(11) = 10.591$, $p = .008$, $\eta^2G = .038$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There were no significant interactions between grammaticality and anteriority or between grammaticality and hemisphere.

In the analysis of the midline regions there was a marginally significant main effect of grammaticality ($F(11) = 4.648$, $p = .054$, $\eta^2G = .040$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and anteriority.

In the time window from **400 to 600ms** there was a marginally significant main effect of grammaticality in the lateral regions ($F(11) = 3.641$, $p = .083$, $\eta^2G = .039$) with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority ($F(22) = 12.076$, $p < .001$, $\eta^2G = .048$). Post hoc *t*-tests for the interaction indicated that the effect of grammaticality was present in the left anterior region ($t(11) = -3.400$, $p = .006$), but not in

the other regions. In this ROI ungrammatical stimuli elicited a more positive response than grammatical stimuli. There was also a marginally significant interaction between grammaticality and hemisphere (F (11) = 3.631, p = .083, η^2G = .014). Post hoc t-tests for this interaction indicated that the effect of grammaticality was significant in the left hemisphere (t(11) = -2.239, p = .047) but not in the right hemisphere.

In the analysis of the midline regions there was no significant main effect of grammaticality. However, there was a significant interaction between grammaticality and anteriority (F(22) = 8.079, p = .013, η^2G = .040). Post hoc t-tests for this interaction indicated that the effect of grammaticality was present in the midline anterior region (t(11) = -2.851, p = .016) but not in the other regions. In this ROI ungrammatical stimuli elicited a more positive response than grammatical stimuli.

In the time window from **600 to 800ms** there was a significant main effect of grammaticality in the lateral regions (F(11) = 15.198, p = .002, η^2G = .208), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority (F(22) = 10.883, p < .001, η^2G = .026). Post hoc t-tests for this interaction indicated that the effect of grammaticality was present in the left anterior region (t(11) = -5.287, p < .001), left central region (t(11) = -3.842, p = .003), right anterior region (t(11) = -3.833, p = .003) and right central region (t(11) = -3.175, p = .009), but not in the posterior regions. In these ROIs ungrammatical stimuli elicited a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and hemisphere.

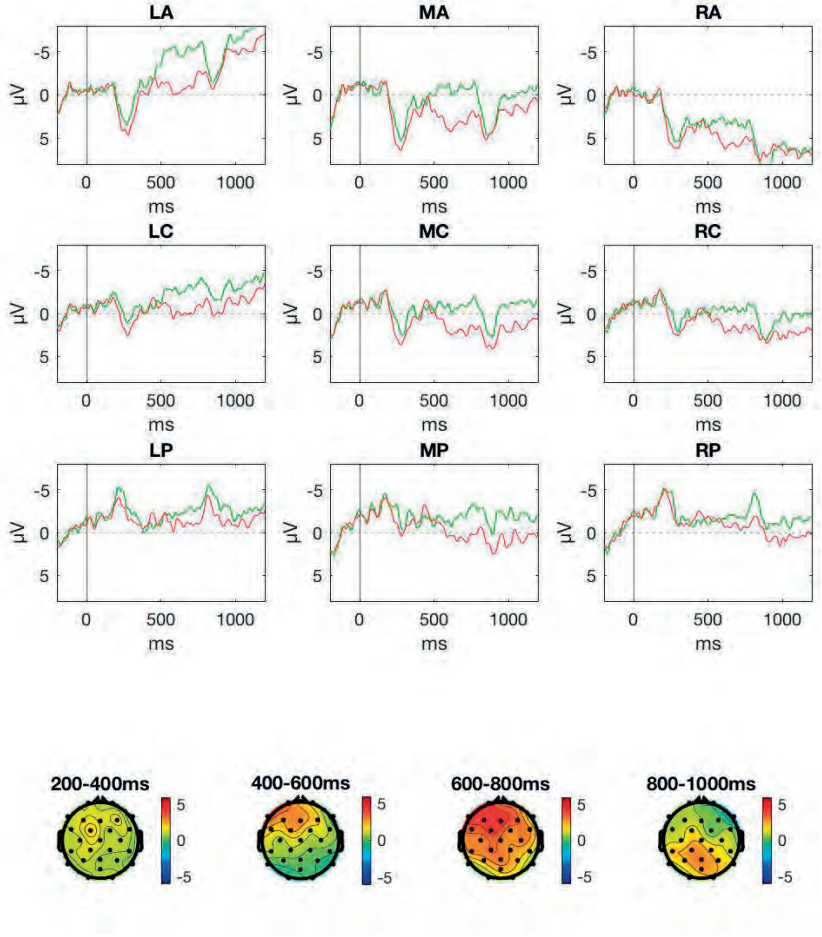
In the analysis of the midline regions there was a significant main effect of grammaticality (F(11) = 9.972, p = .009, η^2G = .281), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and anteriority.

In the time window from **800 to 1000ms** there was a significant main effect of grammaticality in the lateral regions (F(11) = 6.304, p = .029, η^2G = .051), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority (F(22) = 6.452, p = .006, η^2G = .018). Post hoc t-tests for this interaction indicated that the effect of

grammaticality was present in the left central region ($t(11) = -2.489$, $p = .030$), the left posterior region ($t(11) = -3.112$, $p = .010$), and the right posterior region ($t(11) = -4.099$, $p = .002$), but not in the other regions. In these ROIs ungrammatical stimuli elicited a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and hemisphere.

In the analysis of the midline regions there was a significant main effect of grammaticality ($F(11) = 8.228$, $p = .015$, $\eta^2G = .137$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority ($F(22) = 8.656$, $p = .009$, $\eta^2G = .025$). Post hoc *t*-tests for this interaction indicated that the effect of grammaticality was present in the midline central region ($t(11) = -3.022$, $p = .012$) and the midline posterior region ($t(11) = -4.779$, $p < .001$), but not in in the midline anterior region. In these ROIs ungrammatical stimuli elicited a more positive response than grammatical stimuli.

Figure 5.2. Line plots per region of interest and head plots over time for Dutch stimuli.



Note: The green line shows the response to grammatical stimuli, and the red line shows the response to ungrammatical stimuli. L=left, M=midline, R=right, A=anterior, C=Central, P= posterior.

English

In the English language condition, in the time window from **200 to 400ms** there were no significant main effects of grammaticality in either the lateral or midline analysis nor any significant interactions between grammaticality and anteriority, nor between grammaticality and hemisphere (lateral analysis only).

In the time window from **400 to 600ms** there was no significant main effect of grammaticality in the lateral regions. There was also no significant interaction between grammaticality and anteriority. There was a marginally significant interaction between grammaticality and hemisphere ($F(11) = 3.974$, $p = .072$, $\eta^2G = .011$). Post hoc *t*-tests for this interaction indicated that the effect of grammaticality was significant in the left hemisphere ($t(11) = -2.507$, $p = .029$) but not in the right hemisphere.

In the analysis of the midline regions there was no significant main effect of grammaticality. There was also no significant interaction between grammaticality and anteriority.

In the time window from **600 to 800ms** there was a significant main effect of grammaticality in the lateral regions ($F(11) = 11.341$, $p = .006$, $\eta^2G = .188$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There were no significant interactions between grammaticality and anteriority or between grammaticality and hemisphere.

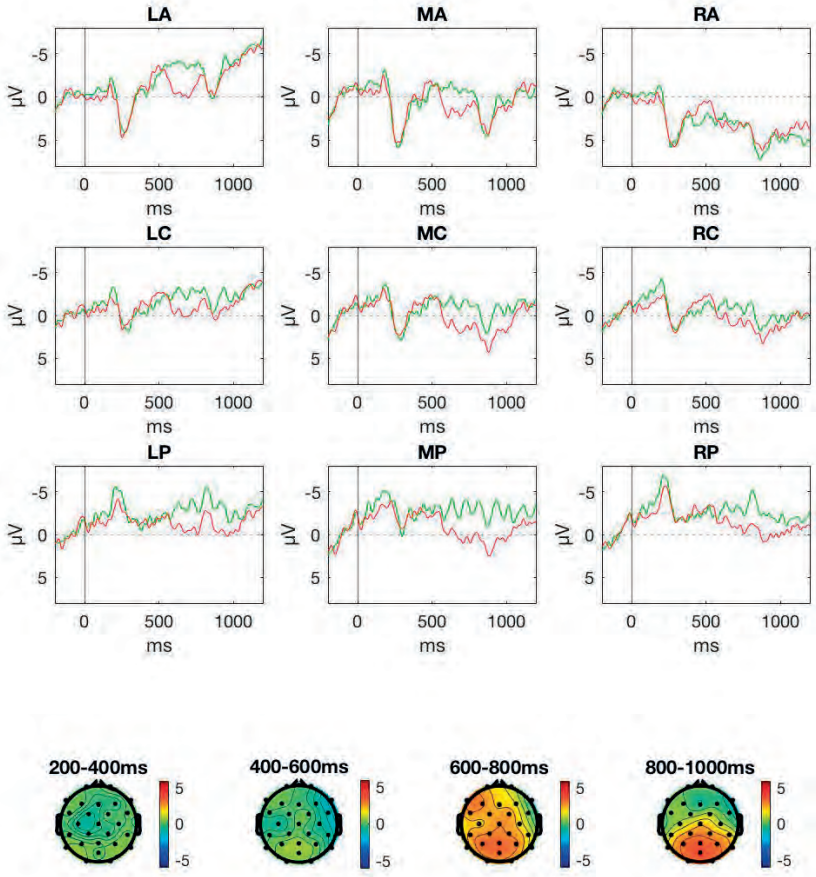
In the analysis of the midline regions there was a significant main effect of grammaticality($F(11) = 13.342$, $p = .004$, $\eta^2G = .296$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and anteriority.

In the time window from **800 to 1000ms** there was a significant main effect of grammaticality in the lateral regions ($F(11) = 5.535$, $p = .038$, $\eta^2G = .079$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority ($F(22) = 12.735$, $p = .002$, $\eta^2G = .063$). Post hoc *t*-tests for this interaction indicated that the effect of grammaticality was present in the left posterior region ($t(11) = -4.409$, $p = .001$) and the right posterior region ($t(11) = -4.339$, $p = .002$), but not in the other regions. In these ROIs,

ungrammatical stimuli elicited a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and hemisphere.

In the analysis of the midline regions there was a significant main effect of grammaticality ($F(11) = 10.428$, $p = .008$, $\eta^2G = .205$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority ($F(22) = 13.739$, $p = .002$, $\eta^2G = .090$). Post hoc *t*-tests for this interaction indicated that the effect of grammaticality was present in the midline central region ($t(11) = -3.585$, $p = .004$) and the midline posterior region ($t(11) = -5.249$, $p < .001$) but not in the midline anterior region. In these ROIs, ungrammatical stimuli elicited a more positive response than grammatical stimuli.

Figure 5.3. Line plots per region of interest and head plots over time for English stimuli.



Note: The green line shows the response to grammatical stimuli, and the red line shows the response to ungrammatical stimuli. L=left, M=midline, R=right, A=anterior, C=Central, P= posterior.

Java

For the Java language condition, in the time window from **200 to 400ms** there was a significant main effect of grammaticality in the lateral regions ($F(11) = 5.444$, $p = .040$, $\eta^2G = .050$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There were no significant interactions between grammaticality and anteriority or between grammaticality and hemisphere.

In the analysis of the midline regions there was a marginally significant main effect of grammaticality ($F(11) = 3.519$, $p = .087$, $\eta^2G = .059$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and anteriority.

In the time window from **400 to 600ms** there was a significant main effect of grammaticality in the lateral regions ($F(11) = 15.200$, $p = .002$, $\eta^2G = .140$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a marginally significant interaction between grammaticality and anteriority in the lateral regions ($F(22) = 3.478$, $p = .078$, $\eta^2G = .019$). Post hoc *t*-tests for this interaction indicated that the effect of grammaticality was present in the left anterior region ($t(11) = -4.548$, $p < .001$), the left central region ($t(11) = -2.985$, $p = .012$), the right anterior region ($t(11) = -3.218$, $p = .008$) and the right central region ($t(11) = -3.842$, $p = .003$), but not in the posterior regions. In these ROIs ungrammatical stimuli elicited a more positive response than grammatical stimuli. There was no significant interaction between grammaticality and hemisphere.

In the analysis of the midline regions there was a significant main effect of grammaticality ($F(11) = 10.890$, $p < .007$, $\eta^2G = .184$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There was also a significant interaction between grammaticality and anteriority ($F(22) = 14.160$, p = .001, $\eta 2G = .037$) with post hoc *t*-tests for this interaction indicating that the effect of grammaticality was present in all three ROIs: the midline anterior region ($t(11) = -4.089$, $p = .002$), the midline central region ($t(11) = -2.636$, $p = .023$) and the midline posterior region ($t(11) = -2.441$, $p = .033$). In all ROIs, ungrammatical stimuli elicited a more positive response than grammatical stimuli.

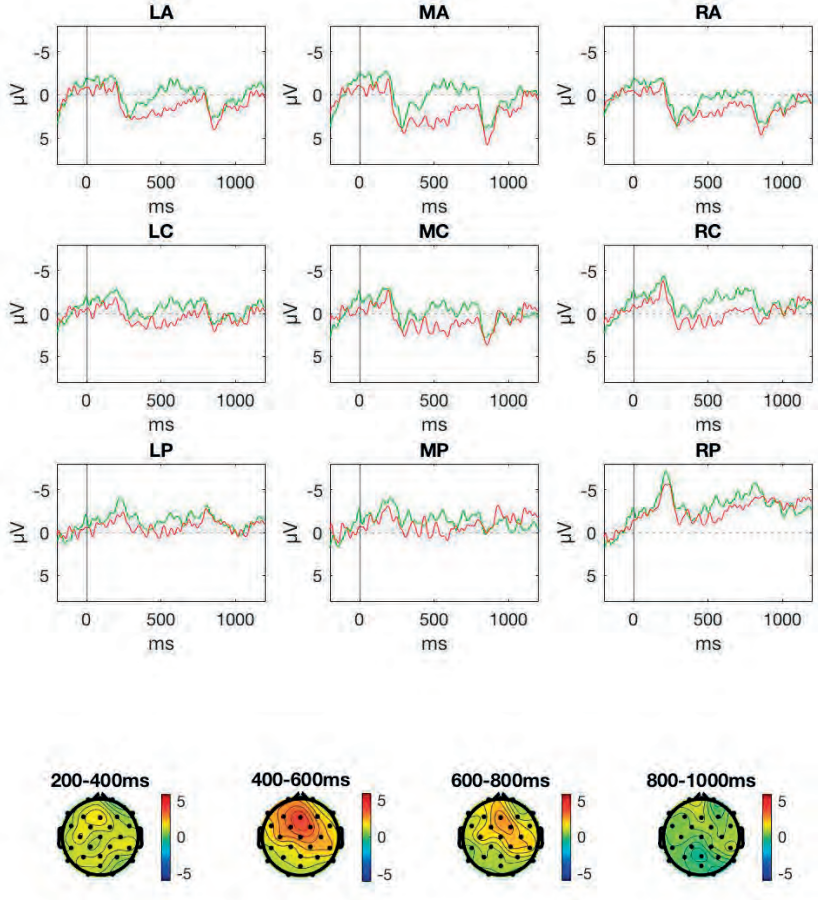
In the time window from **600 to 800ms** there was a marginally significant main effect of grammaticality in the lateral regions ($F(11) = 4.327$, $p = .062$, $\eta^2G = .052$), with ungrammatical stimuli eliciting a more positive response than grammatical stimuli. There were no significant interactions between grammaticality and anteriority or between grammaticality and hemisphere.

In the analysis of the midline regions there was no significant main effect of grammaticality, but there was a significant interaction between grammaticality and anteriority ($F(22) = 6.224$, $p = .007$, $\eta^2G = .015$), with post hoc *t*-tests for this interaction indicating that the effect of grammaticality was present in the midline anterior region ($t(11) = -2.915$, $p = .014$) but not in the other regions. In this ROI ungrammatical stimuli elicited a more positive response than grammatical stimuli.

In the time window from **800 to 1000ms** there was no significant main effect of grammaticality in the lateral regions. There were also no significant interactions between grammaticality and anteriority or between grammaticality and hemisphere.

In the analysis of the midline regions there was no significant main effect of grammaticality, but there was a marginally significant interaction between grammaticality and anteriority ($F(22) = 3.577$, $p = .074$, $\eta^2G = .024$). However, post hoc *t*-tests for this interaction revealed no significant effects of grammaticality in any of the ROIs.

Figure 5.4. Line plots per region of interest and head plots over time for Java stimuli.



Note: The green line shows the response to grammatical stimuli, and the red line shows the response to ungrammatical stimuli. L=left, M=midline, R=right, A=anterior, C=Central, P= posterior.

Comparison of the ERP patterns per language

When comparing the difference waves between grammatical and ungrammatical conditions between the three languages, in the time window from **200 to 400ms** there were no significant main effects of language in either the lateral or midline analysis nor any significant interactions between language and anteriority, nor between language and hemisphere (lateral analysis only).

In the time window from **400 to 600ms** there was a significant main effect of language in the lateral regions ($F(22) = 5.820$, $p = .009$, $\eta^2G = .146$). Post hoc *t*-tests indicated that the effect in response to Java was stronger than the effects in response to Dutch ($t(71) = -2.896$, $p = .015$) and English ($t(71) = -5.637$, $p < .001$) while the effect in response to Dutch was also stronger than the effect in response to English ($t(71) = 3.301$, $p = .005$). There was also a significant interaction between language and anteriority ($F(44) = 4.716$, $p = .020$, $\eta^2G = .055$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the left central region where the effect in response to Java was stronger than the effect in response to English ($t(11) = -2.823$, $p = .050$), in the right anterior region where the effect in response to Java was stronger than the effect in response to English ($t(11) = -3.316$, $p = .021$), and the effect in response to Dutch was stronger than the effect in response to English ($t(11) = 3.274$, $p = .022$) and in the right central region where the effect in response to Java was stronger than the effect in response to Dutch ($t(11) = -3.580$, $p = .013$) and stronger than the effect in response to English ($t(11) = -3.259$, $p = .023$). There was no significant interaction between language and hemisphere.

In the analysis of the midline regions, there was a marginally significant main effect of language ($F(22) = 3.098$, $p = .065$, $\eta^2G = .088$). Post hoc *t*-tests indicated that the effect in response to Java was stronger than the effects in response to Dutch ($t(35) = -2.527$, $p < .048$) and English ($t(35) = -3.232$, $p = .008$). There was also a significant interaction between language and anteriority ($F(44) = 6.017$, $p = .017$, $\eta^2G = .054$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the midline anterior region where the effect in response to Java was stronger than the effect in response to English ($t(11) = -2.867$, $p = .046$).

In the time window from **600 to 800ms** there was a marginally significant main effect of language in the lateral regions ($F(22) = 2.719$, $p = .088$, $\eta^2G = .055$). In contrast to the earlier time windows, post hoc *t*-tests indicated that the effect in response to Java was *weaker* than the effects in response to Dutch ($t(71) = 4.002$, $p < .001$) and English ($t(71) = 2.825$, $p = .018$. There was also a significant interaction between language and anteriority ($F(44) = 4.420$, $p = .004$, $\eta^2G = .025$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the left anterior region, where the response to Dutch was stronger than the response to Java ($t(11) = 2.873$, $p = .045$). There was no significant interaction between language and hemisphere.

In the analysis of the midline regions, there was a marginally significant main effect of language ($F(22) = 2.736$, $p = .087$, $\eta^2G = .076$). Post hoc *t*-tests indicated that the effect in response to Java was weaker than the effects in response to Dutch ($t(35) = 3.660$, $p = .003$) and English ($t(35) = 2.822$, $p = .023$). There was also a significant interaction between language and anteriority ($F(44) = 4.816$, $p = .012$, $\eta^2G = .020$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the midline posterior region where the effect in response to English was stronger than the effect in response to Java ($t(11) = 3.092$, $p = .031$).

In the time window from **800 to 1000ms** there was no significant main effect of language in the lateral regions, but there was a significant interaction between language and anteriority ($F(44) = 6.400$, $p = .005$, $\eta^2G = .069$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the left posterior region where the effect in response to English was stronger than the effect in response to Java ($t(11) = 2.953$, $p = .039$). There was no interaction between language and hemisphere.

In the analysis of the midline regions, there was a significant main effect of language ($F(22) = 4.331$, $p = .026$, $\eta^2G = .162$). Post hoc *t*-tests indicated that, once again, the effect in response to Java was weaker than the effects in response to Dutch ($t(35) = 3.379$, $p = .005$) and English ($t(35) = 3.347$, $p = .006$). There was also a significant interaction between language and anteriority ($F(44) = 10.180$, $p < .001$, $\eta^2G = .101$). Post hoc *t*-tests for this interaction indicated that there was a significant effect of language only in the midline

posterior region where the effect in response to Java was weaker than the effects in response to Dutch ($t(11) = 3.082$, $p = .031$) and English ($t(11) = 4.002$, $p = .006$).

5.3.3 Summary of ERP results

The statistical ERP results are summarised in Tables 5.3 and 5.4. Overall, we find a positive deflection in responses to ungrammatical sentences for all three languages that is present between 400 and 800ms. Statistically, the effects for Java and Dutch start first, reaching significance in the time window from 200ms to 400ms, followed by English which reaches significance in the left hemisphere in the time window from 400-600ms, and reaches overall significance in the time window from 600 to 800ms. Visually, the main deflection for Java appears to start first, at around 400ms. For Dutch, although there is already an early statistically significant effect of grammaticality, the main deflection appears to start at around 500ms. The effect for English appears to start last, at around 600ms. We see that English and Dutch show prolonged positive effects in the time window from 800 to 1000ms as well, while this effect is absent for Java. The effects for Dutch and Java start more frontally, while the effect for English is more broadly distributed. The effects for English and Dutch in the time window from 400-600ms appear to occur mostly in the left hemisphere. The late effect for Dutch and English from 800 to 1000ms is mainly found in the central and posterior regions.

Table 5.3. Summary of ANOVA results by language.

		200-400ms	400-600ms	600-800ms	800-1000ms
Dutch	Main effect	✓		✓	✓
	Anterior		✓	✓	
	Midline		✓	✓	✓
	Posterior				✓
	Left hemisphere		✓		
	Right hemisphere				
English	Main effect			✓	✓
	Anterior				
	Midline				✓
	Posterior				✓
	Left hemisphere		✓		
	Right hemisphere				
Java	Main effect	✓	✓		
	Anterior		✓	✓	
	Midline		✓		
	Posterior		✓		
	Left hemisphere				
	Right hemisphere				

Note: This table shows in which time windows there were significant main effects of grammaticality for each language, and, if there was a significant interaction with anteriority in either the lateral or the midline regions, whether effects were present in the anterior, midline or posterior regions. If there was an interaction between grammaticality and hemisphere the table showes whether the effects were present in the left or the right hemisphere.

Table 5.4. Summary of statistical results comparing grammaticality effects between the languages.

		200-400ms	400-600ms	600-800ms	800-1000ms
Main effects			Java > Dutch&English		Dutch&English > Java
Grammaticality * Anteriority	Anterior		Dutch&Java > English	Dutch > Java	
	Central		Java > Dutch & English		
	Posterior			English>Java	Dutch&English > Java

Note: This table shows in which time windows there were main effects of language, and the directions of those effects. If there was an interaction between language and anteriority in either the midline or lateral regions, the table also shows in which region an effect was found (anterior, central or posterior) and what the direction of the effect was in that region.

5.4 DISCUSSION

This study aimed to examine whether a programming language is processed similarly to natural languages. This was done by examining brain responses elicited by syntax errors in a programming language and comparing those to responses elicited by grammatical violations in participants' first (Dutch) and second (English) natural languages. Specifically, we compared ERP effects in response to bracket errors in "if" and "while" statements in Java to responses in reaction to subject-verb disagreements in Dutch and English. We visually and statistically inspected the responses per language over four time windows in nine different ROIs.

We found a positive deflection in response to violations for all three languages. This suggests that participants were sufficiently proficient in all three languages to process the presented violations as errors. This was confirmed by the high accuracy on the grammaticality judgment questions. Additionally, finding the current ERP responses confirms that the violations in all three languages were not semantic in nature, as semantic

violations are known to elicit a negative wave around 400ms after the violation (Friederici, 1995; 2002; Kutas & Hillyard, 1980). However, the differences in latency, offset and scalp distribution between Java, Dutch and English was unexpected. In this section we summarise these differences and discuss various possible explanations.

The early frontal effect for Dutch that reached significance between 200 and 400ms was unexpected. Visually, the main deflection of the effect appeared to start at around 500ms, with a broad scalp distribution. It is possible that the early significant effect was noise due to the small sample size of the study. However, the early positivity seems to precede the main P600 effect, and may be a separate effect related to visual processing and expectancy (Potts, 2004), perhaps brought on by the way stimuli were presented and their relative brevity. The visual and statistical effects for Dutch and English at around 500-600ms are in line with the P600 effect in response to sentences with a violation or unpreferred use of syntax (Friederici et al., 2002; Hagoort & Brown, 2000). Additionally, the prolonged late positivity effects in both Dutch and English in the posterior regions were in line with a typical P600 effect (Hagoort & Brown, 2000), and probably reflect reanalysis of the stimulus (Molinaro et al., 2008).

For Java the effect occurred early. It was statistically detectable from 200ms and became visually prominent at around 300ms. It occurred mainly in the frontal and central regions. Unlike for Dutch, the effect seemed to be uniform (consisting of only one *peak* or component) throughout its entire duration (200-800ms). Visually, as well as statistically, the effect for Java had a shorter duration and ended earlier than the effects for Dutch and English. Furthermore, the effect for Java was never predominantly distributed in the posterior regions, which did happen for the effects for Dutch and English and is expected for the later part of a P600 effect (Hagoort & Brown, 2000). Therefore, the effect for Java cannot be unequivocally interpreted as an early P600 effect, as its characteristics are more in line with the P300 effect (Coulson et al., 2010; Osterhout, 1999; Osterhout et al., 1996; Sassenhagen & Fiebach, 2019; Sassenhagen et al. 2014). Based on the early latency, the short duration and the frontocentral distribution, the effect favours a P300 interpretation (Osterhout, 1999; Osterhout et al., 1996). In this case, the bracket error in Java is considered as a more superficial violation of expectations rather than a grammatical violation. This violation can still be language-related, such as incorrect spelling, or it can be a formatting

error that is not cognitively integrated in the same way as in natural languages. When comparing our findings for Java to the findings by Osterhout et al. (1996) for unexpected capitalisation and by Vissers et al. (2006) for orthographic (spelling) violations in English, we see that the ERP responses to orthographic errors resembled the responses to our Java violations most closely. The orthographic errors elicited an early response, starting around 400ms (which can also be classified as a P300). This response was also fronto-centrally distributed and, thus, matches the current response in Java in both latency and topography. Hence, interpretation of the early positive wave as a P300 effect is a plausible explanation for the differences in onset, offset and topography between Java and the natural languages in the current study.

However, we cannot completely rule out the possibility that the effect is an early P600 effect, especially given the fact that the P600 for Dutch also had an earlier component. If we were to interpret the effect as the P600, we have to address several more issues, mainly related to its latency. Since Dutch was the participants' native language, we expected the violations in Dutch to elicit the earliest ERP response, which would then be followed by a later onset for English and Java. Friederici (1995) proposed that the latency of the P600 effect reflects the complexity of processing necessary for the revision of the initially preferred reading. In the current study, there are two factors that may have influenced the ease of processing across the languages. The first is the level of fluency in each language. Based on the literature, we expect the most fluent language to elicit the earliest response (Kotz, 2009; Rossi et al., 2006; Weber-Fox & Neville, 1996). We assumed that Dutch, being the native language, was also the most fluent language. However, it is possible that although Dutch was the programmers' native language, and English was typically learned before Java, Java is currently the written language to which they are exposed the most. Professional programmers and programming students spend the majority of their worktime working with code, rather than reading texts in natural languages such as Dutch or English. Unless these programmers read regularly in their free time, it is possible that they have been primarily exposed to written Java rather than written texts in Dutch or English for multiple years. This higher exposure and proficiency in Java reading could explain why the errors were processed more quickly and, therefore, elicited an earlier P600 compared to Dutch and English.

A second factor that may have influenced the ease of processing for the Java violations is stimulus length. The violation in Java becomes clear when a single bracket appears, while for Dutch and English the violation becomes clear at the appearance of a multiple-letter word, that is decoded and then processed. It is possible that the single character length of the bracket versus the multiple character words makes processing of the Java violations quicker than processing of the violations in Dutch and English, leading to an earlier P600 effect in Java. We cannot exclude this possibility with the current results. However, future studies could investigate this by conducting a similar experiment where the Java error consists of multiple characters.

If the effect in Java does indeed reflect an (early) P600, it only elicits the first stage of registering, where the reader identifies the violation, without later reanalysis (Molinaro et al, 2008). This may be because the bracket violation in Java is more easily recognised and resolved than the violations in Dutch and English. Programmers may assume that the "if" or "while" function was chosen deliberately, and, thus, the curly bracket must simply be replaced with a round one (rather than replacing "if" or "while" with, for example, "else" which requires a curly bracket). In Dutch and English this resolution may be more complicated because the error can be resolved in several ways. For example, readers may consider either the inflection on the verb or the pronoun as incorrect. The resolution of this violation is more dependent on the context of the sentence and, therefore, may require more in-depth processing from the reader. A lack of reanalysis is also expected if Java bracket errors are indeed comparable to orthographic violations. In that case, they are quickly resolved as a spelling error with no alternative meaning. Therefore, a lack of the repair and reanalysis stage would explain the short duration of the effect, as well as its lack of a posterior distribution, which is mainly related to repair and reanalysis (Hagoort & Brown, 2000).

5.4.1 Conclusion

Based on previous studies, the effect in response to Java bracket violations can be interpreted either as an early P600 or as a P300 effect. Regardless of the exact classification of the effect, the latency and topography of the Java response suggest that this type of error is similar to an orthographic violation in natural language. Based on the current

results, the most likely conclusion is, thus, that the bracket violation in Java is language related, but not of a grammatical nature. However, we cannot exclude the possibility that the Java response reflects a P300 effect that is unrelated to language, or that it reflects an early P600 that is related to grammar. This means that, based on the current experiment, we are unable to conclude whether syntax in a programming language is processed similarly to grammar in a natural language. Future studies with different stimuli will be necessary to disentangle these different possibilities.

The current study lays the foundations for these future studies by showing that processing of programming languages can be studied through ERPs. Future research could build on this by studying a wide variety of violations in different programming languages, aiming to determine the linguistic nature of different elements in these types of languages. Through multiple ERP studies, we should eventually be able to determine which elements of a programming language are of a grammatical, orthographic, semantic or entirely un-languagelike nature. This will help us understand the way in which people process and understand programming languages, and the way in which programming languages differ from or resemble natural languages.

Chapter 6

General Discussion

6.1 OVERVIEW

The research reported in this thesis aimed to contribute to a better understanding of computer programming as a skill in modern society and education. Three relevant areas of research within the field of cognition were identified: cognitive skills, personality traits, and processing in the brain. To study programming in the first two areas, it was necessary to lay methodological foundations to be able to measure programming skill. Specifically, the aim of Chapter 2 was to answer the methodological research question 1) Can we create two reliable parallel short versions of the Second Computer Science 1 (SCS1) programming test? Subsequent chapters then aimed to answer three further research questions: 2) Which cognitive skills predict success at the end of a programming course? 3) Do autistic traits predict success in a programming course? and 4) Is a programming language processed similarly to a natural language?

In this chapter, I first discuss the answers to these four research questions. Then, I focus on the findings across the different studies regarding the relationship between programming and language. Next, I reflect on implications of the findings for education. Finally, I give suggestions for future research, and discuss the broader impact of the outcomes of the thesis.

6.2 ANSWERS TO RESEARCH QUESTIONS

(1) Can we create two reliable parallel short versions of the Second Computer Science 1 (SCS1) programming test?

Chapter 2 reports the development of two shortened versions of the Second Computer Science 1 programming test (SCS1-S). The results showed that these versions could not be considered fully parallel, and that Version 1 was of questionable quality. However, Version 2 was of comparable reliability and validity to the full SCS1 that was validated by Parker et al. (2016) and Xie, Davidson et al. (2019). Hence, this research has resulted in a validated short version, allowing this test to be used in a wider variety of studies that require shorter testing time.

In addition, it was clear that the test was difficult for the current student population: they showed low accuracy and did not complete many items. This is in line with the findings

of both Parker et al. (2016) and Xie, Davidson et al. (2019). It suggests that our Australian undergraduate population performed similarly to the American undergraduate students, showing that this test can be used to measure programming skill across different courses, institutions and countries. However, this finding also means that the test is limited in its ability to measure differences in programming skill for populations with very little programming experience, and that the test cannot be used as a pre-test for computer science courses or research studies when students have no previous programming experience.

Based on the findings in Chapter 2 I give two suggestions for future use of the SCS1 and the SCS1-S. First, I suggest only using Version 2 of the shortened SCS1 test, as this version was found to be of similar quality as the full SCS1 test. Unfortunately, because testing for the studies in Chapters 3 and 4 happened simultaneously with the validation study in Chapter 2, Version 1 was still used in the current studies. This may have caused more noise in the independent programming measure, and therefore may have led to lower correlations than if we had only used Version 2. Second, I suggest that both the full SCS1 and the short versions of this test are most suitable to be used with students who are slightly more advanced than the students in the current studies. This means that the need for an easier test of basic programming skills still remains. Future studies could simplify the questions of the SCS1 or design an entirely new programming test that is more sensitive to differences in early gains in programming skill following the start of programming training.

(2) Which cognitive skills predict success at the end of a programming course?

In the study in Chapter 3, we examined the extent to which five cognitive skills, measured at the start of an Australian undergraduate programming course, correlated with programming performance on the SCS1-S, as an independent programming measure, and with the course grades at the end of the semester. A task measuring logical reasoning was the most consistent predictor, significantly predicting programming performance on both the course exam and the SCS1-S. Algebra and vocabulary learning tasks were also found to predict programming performance, but only on the independent programming test. This supports the idea that the predictive value of a cognitive skill depends on the way programming is assessed, as the testing format may emphasise certain aspects of

programming skills over others. When measuring programming skills with a stricter test and with more time pressure, rather than a more flexible format, participants may rely more on their problem-solving speed and their memorisation of the programming language, therefore increasing the predictive values of algebra and vocabulary skills. Additionally, the language component (consisting of grammar learning and vocabulary learning) did not predict programming skill on either outcome measure, while the algorithmic/mathematics component (consisting of pattern recognition, logical reasoning and algebra) predicted performance on both measures. For a more extensive discussion of the relationship between language and programming, see the section on "The role of natural language skills in programming" later in this chapter.

(3) Do autistic traits predict success in a programming course?

In the study in Chapter 4, we measured autistic traits at the start of an Australian undergraduate programming course and correlated these with programming performance on the independent SCS1-S measure and the course-related grades at the end of the semester. Autistic traits were found to be higher in the current student population compared to the general population. However, they did not predict programming skill. There were also no significant correlations between autistic traits and the cognitive skills, nor were there any significant correlations between the individual subscales of the Autism Spectrum Quotient (AQ) and programming skill. These results suggest that despite popular stereotypes of programmers as people with autistic characteristics, autistic traits do not play a role in predicting programming aptitude.

(4) Is a programming language processed similarly to a natural language?

In the study in Chapter 5, we studied Event Related Potentials (ERPs) measured with Electroencephalography (EEG) in response to violations in participants' native language (Dutch), second language (English) and a programming language (Java). Violations in all three languages elicited positive ERP deflections. However, there were differences between the languages with regard to onset, offset and scalp distribution of the effects. Specifically, the early onset and offset of the effect in Java, as well as its frontal and bilateral scalp distribution suggested that this type of bracket violation was processed differently to the subject-verb violations in the two natural languages. Based on both timing and scalp

distribution, the effect for Java could be interpreted as either an early P600 or a late P300 effect, and therefore we cannot draw a conclusion as to whether this effect was language related. However, the effect in response to the bracket violations was similar to that observed in response to orthographic violations in natural language (Vissers et al., 2006), suggesting that programmers may perceive it as incorrect spelling. In sum, based on the results of the current study, we cannot conclusively answer the question if Java is processed similarly to a natural language.

6.3 THE ROLE OF NATURAL LANGUAGE SKILLS IN PROGRAMMING

Based on the levels of the PGK-hierarchy model of the programming process (named after Perrenet, Groote and Kaasenbrood who first defined it; Perrenet et al., 2005) as described by Armoni (2013), I speculated that the first two levels of the model, the problem level and the object level, rely most strongly on algorithmic/mathematical skills. For the third level of the model, the program level, I speculated that this level relies more strongly on knowledge of the programming language and, thus, on language skills. It was thus expected that language skills would be predictive of programming ability. However, the results from Chapter 3 suggest that programming skill, following the course undertaken by our participants, relied more on algorithmic/mathematical skills and less on language skills, with only vocabulary learning predicting performance only on the SCS1-S. In Chapter 5, the Java bracket errors were found to be processed differently from Dutch and English subject-verb violations. This means that the results of the research in this thesis show only a limited relationship between language and programming. Based on the results of the current thesis and previous studies, I will now present two possible explanations for the limited role for natural language as found in the studies reported in Chapters 3 and 5.

First, it is possible that programming languages do resemble natural languages and are processed similarly, but that the specific type of programming language violation tested in Chapter 5 was just not grammatical in nature. If programming languages do indeed resemble natural languages, then the lack of predictive value for language skills in the study in Chapter 3 may have been due to teaching focus. Just like many modern-day programming courses, the undergraduate course taken by our participants did not teach the programming language explicitly (Hermans, 2020; Portnoff, 2018), and, therefore, focussed

on the first two levels of the PGK-hierarchy but not on level 3. Therefore, programming performance in the current course may have relied more on algorithmic/mathematical skills and less on language skills. It is possible that language skills would be more predictive of outcomes from a course that teaches programming languages more explicitly. Additionally, it is also possible that language skills are more predictive of the speed of the learning process, rather than end performance. This is supported by the findings of Prat et al. (2020), who measured both how well people performed at the end of the course, and how quickly they progressed throughout the course. They showed that language skills were most predictive of how quickly people learned. The study in Chapter 3 did not include a measure of learning speed. It is possible that the current language skills measured in Chapter 3 (grammar learning & vocabulary learning) would have been more predictive of such a measure.

However, it is also possible that we found little predictive value for language skills because programming languages are different from natural languages and are thus processed differently. Based on the results of Chapter 5 there are two possible options. First, it is possible that a programming language is processed in the same way as natural languages by the brain, but that its linguistic properties are different. For example, programming languages may contain more orthographic and fewer syntactic rules than natural languages. However, it is also possible that programming languages are not processed like a language at all, and that programming is an altogether different skill from language. This was suggested in a recent study by Ivanova et al. (2020), where they found that reading a programming language activated different brain areas to reading sentence problems. However, their results still showed that reading programs did require parts of the language system, and therefore could not exclude that a programming language is in part processed like a language. In order to determine which of these explanations is correct, future ERP studies will have to study a wider variety of linguistic elements of programming languages. More specific suggestions for future ERP studies can be found in the upcoming section “Suggestions for future studies”.

6.4 POTENTIAL IMPLICATIONS FOR EDUCATION

The studies reported in this thesis aimed to unravel the relationship between programming and various cognitive skills. The ultimate aim of this field of research is to guide programming education. Although more experimental research is still needed, this thesis does give some indicators which may be helpful for educators to consider.

First, in Chapter 3, results show a convincing role for logical reasoning skills when predicting programming skill at the end of a typical undergraduate course. Therefore, educators could use logical reasoning assessments to identify students who are not strong in this area and so may have difficulties during the course and, conversely, those with strong logical reasoning skills, who may do well. Additionally, it is possible that more explicit teaching of logical reasoning could improve programming skills. This would be in line with suggestions from educational researchers stating that teaching computational thinking prior to teaching programming would be beneficial for the learning process (e.g., Lu & Fletcher, 2009). Grover et al. (2019) showed that for middle schoolers, using non-programming activities to teach programming skills and concepts such as Boolean logic, loops and variables did indeed improve their programming performance in Scratch. Future training studies will have to confirm whether such teaching methods, including explicit teaching of logical reasoning, also improve programming performance in undergraduate students. In the meantime, it may already be worth considering unplugged teaching activities in educational programs.

Second, cognitive skills that predicted programming performance varied depending on the way in which programming skill was assessed. It is important for educators to note that not just the content, but also the format, of assignments and exams may favour students with certain skills over others. For example, assessments with formats like the SCS1, with strict limitations regarding the use of outside sources and with time pressure, may favour students with relatively good vocabulary learning and algebra skills. Therefore, it is important for educators to carefully consider the skills that they wish to measure, and to design their assessments accordingly.

Finally, at present, empirical results, both of the current thesis and of previous studies, are inconclusive when it comes to the relationship between programming and

natural language. Some studies suggest that language skills do predict programming performance, and that programming languages are processed similarly to natural languages in the brain (e.g., Floyd et al., 2017; Prat et al., 2020; Siegmund et al., 2014). Other studies show differences in processing between natural languages and programming languages and argue that these cognitive skills are distinct (e.g., Ivanova et al., 2020). Based on the current state of the findings in the literature and on the findings of the current thesis, I argue that language skills may play a role in programming, but that this role could be influenced by the way that programming is taught. In current educational systems programming is typically taught with a constructivist approach, in which students are expected to learn the programming language by using it, without much direct instruction (Hermans, 2020; Portnoff, 2018). This lack of explicit instruction may diminish the emphasis on language in programming courses. Some researchers have argued for more explicit programming language teaching (e.g., Hermans, 2020; Hermans & Aldewereld, 2017; Portnoff, 2018). The exact role of language in programming needs to be studied further to inform decisions regarding the best methods to teach programming languages.

6.5 SUGGESTIONS FOR FUTURE STUDIES

Based on the experimental chapters in the current thesis, in this section I will discuss four directions for future research: 1) Interactions between course content, test format, and cognitive skills, 2) Further disentangling the relationships between cognitive skills and programming, 3) Autistic traits and programming, and 4) How to take advantage of the newly developed ERP presentation method.

With all of these all suggestions for future research, it is important to note that programming languages vary widely in their syntax, rules and other properties. The current thesis studied two languages: Java and Processing. Consequently, based on the studies in this thesis, we are unable to determine the extent to which the results would generalise to other programming languages. Therefore, it will be important for future studies to extend the work reported here by investigating the same issues in a wide variety of programming languages.

6.5.1 Interactions between course content, test format, and cognitive skills

In the current thesis I make two important points about course content and test format. First, results suggest that course content influences performance on the SCS1-S. Second, results suggest that course content and test format may influence which cognitive skills are predictive of programming skill. In this section I give recommendations for future research on these two points.

The results presented in Chapter 2 show that the SCS1 items are able to successfully measure programming skills across different institutions and in different countries. However, the results also suggest that course content or format can influence scores on the SCS1 items to some extent: some items were easier while other items were more difficult for our population compared to the populations studied by Parker et al. (2016) and Xie, Davidson et al. (2019). To get a better idea of the relationship between course content and performance on the SCS1 test items, it would be beneficial to validate this test in a wider variety of courses. In addition, researchers should carefully consider *a priori* how the test relates to the curriculum used in the study.

Additionally, the results of the experiments in Chapter 3 suggest that course content and test format may influence which skills are predictive of programming performance at the end of the course. Specifically, algebra and vocabulary learning skills were found to be more predictive of performance on a programming test with a strict format than on a test with a more flexible format. In addition, we argued that the way in which programming syntax is taught may influence the importance of language skills in the learning process. Even in ERP experiments, like the one conducted in Chapter 5, it is possible that the way a programming language is processed also depends on the way it was taught. For example, if educational programs focus more explicitly on the syntax rules of the programming languages, violations may be processed more as syntactic violations. This hypothesis is supported by findings of natural language studies that show that the method of second language instruction can influence the size, shape and scalp distribution of ERP effects (Morgan-Short, 2012). Therefore, it is relevant for future studies to investigate whether different methods of instruction (e.g., explicit vs implicit syntax teaching), influence 1) the relationship between programming performance and cognitive skills, and 2) the brain

responses to specific syntactic elements. This will help to further clarify the ways in which instruction may or may not influence the role of different cognitive skills in programming.

6.5.2 Further disentangling the relationships between cognitive skills and programming

The results presented in the current thesis show that there are relationships between cognitive skills and acquiring programming skill. In Chapter 3 we showed these relationships within a correlational study design. This design is suitable and convenient for exploratory studies aiming to get an impression of which subskills are important for learning to program. Also, this design allows us to test hypotheses in a natural learning context. However, future studies are still necessary to determine the directions of these relationships. This could be achieved by testing the hypotheses from the current thesis with experimental training studies. These studies could include a control condition or manipulation of instruction methods. For example, studies can test whether more explicit teaching of logical reasoning skills improves programming performance.

Secondly, to further refine the PGK-hierarchy (Armoni, 2013; Perrenet et al., 2005), it would be interesting to test whether specific cognitive skills relate to specific sub-skills of programming. This could be done by splitting programming assessments into validated assessments of programming subskills (e.g., problem solving, programming syntax knowledge, etc.), corresponding to the three levels described in the model, and testing which cognitive skills correlate with which specific level.

Additionally, to further disentangle the relationship between programming and language skills, as mentioned previously, future studies should first investigate whether teaching formats with more emphasis on programming syntax show a stronger relationship with language-related cognitive skills. Future studies could also use the ERP methodology established in the current thesis (see “How to take advantage of the newly developed ERP presentation method” later in this chapter) to test more different elements in different programming languages to determine the extent to which different elements of programming languages have language-like properties. Specifically, based on the results of a recently published study by Ivanova et al. (2020), programming languages with a more

text-based structure, such as Python, are expected to show stronger linguistic resemblance to natural languages than graphical languages such as Scratch Junior.

Finally, future ERP studies could compare ERP effects across programmers with different levels of expertise to examine how proficiency in a programming language is reflected in the linguistic processing. Floyd et al.'s (2017) functional Magnetic Resonance Imaging (fMRI) study found that, for expert programmers, brain activity whilst reading code was virtually indistinguishable from brain activity during natural language processing, while for less experienced programmers brain activity during the different types of language processing was more distinct. These differences in expertise may also be visible in EEG signals of participants of varying levels of proficiency, where we may expect more proficient natural second language speakers to show ERPs that are more similar in timing and distribution to those in a native natural language (Kotz, 2009; Rossi et al., 2006; Weber-Fox & Neville, 1996). The study in Chapter 5 did not have a large enough sample size with enough variation in proficiency to test this hypothesis for programmers, but it is possible that the more proficient the programmer, the more the response to Java would resemble the response to Dutch or English.

6.5.3 Autistic traits and programming

The results of the study presented in Chapter 4 did not show any evidence for a relationship between autistic traits and programming skill. Therefore, if aiming to predict programming performance, it is probably more useful to focus on cognitive skills in future studies. However, it is possible that the lack of a relationship between autistic traits and programming in the current study was due to the limited reliability of the AQ subscales used in Chapter 4. Therefore, future similar studies could be conducted using a test that is more suitable to measure separate autistic traits. Specifically, in Chapter 4, the Subthreshold Autistic Traits Questionnaire (SATQ; Kanne et al., 2011) is suggested as a potentially appropriate measure. This test is more sensitive to different autistic traits than the AQ, allowing investigation of the relationship between individual autistic traits and programming skill more precisely. This may either show that there is a relationship between specific autistic traits and programming skills which was missed with the AQ, or it may confirm that there is indeed no relationship.

Secondly, it is possible that there is a relationship between autistic traits and interest (rather than aptitude) in programming. This could be studied by administering questionnaires asking students about their reasons for choosing a course at the start, and about course enjoyment at the end of programming courses. Here we may expect students who take the course as an elective to show higher autistic traits than students who take it as a compulsory course. We may also expect students with higher autistic traits to express more course enjoyment than students with lower autistic traits.

6.5.4 How to take advantage of the newly developed ERP presentation method

In Chapter 5, an ERP experiment was conducted to study responses to grammatical and ungrammatical snippets of a programming language, and to compare these responses to responses to grammatical violations in Dutch (first language) and English (second language). This required the adaptation of typical ERP stimulus presentation to the programming context. Specifically, the full length of the stimulus was presented in asterisks on the screen at once, after which the words or symbols were filled in by replacing the asterisks word by word (or per semantic entity for code). Stimuli for all three languages (Java, Dutch, and English) were presented in this way. The effects for Dutch and English were similar to the effects found in previous studies, thus suggesting that this new method of presentation for ERP stimuli elicits similar ERP effects as the traditional method, whilst presenting stimuli in a way that is more natural for programming languages. This method can also be considered for ERP studies with natural languages, as this way of presentation resembles the natural reading experience more closely than some currently used methodologies. To ensure that this method of presentation is reliable for eliciting the whole range of ERP effects, it would be advisable to replicate and extend the results found in Chapter 5 with various types of violations in different languages.

6.6 OVERALL CONCLUSION AND IMPACT

Based on the results of the studies in this thesis, I conclude that logical reasoning is the most reliable cognitive skill to predict programming performance following an introductory programming course. Additionally, algebra and vocabulary learning also seem to be of importance, but only when programming skill is measured with a test that is administered

under time pressure and that limits use of outside sources. Autistic traits did not predict programming skill, but I hypothesised that they may predict an interest in programming.

Based on the value of vocabulary learning in predicting programming skill, and the finding that programming language processing shows some similarities to of natural language processing, I suggest that natural language skill *does* play a role in programming acquisition. However, I suggest that this role is dependent on the way the programming language is taught.

The results of this thesis provide us with essential new knowledge on the cognitive nature of programming and facilitate future research into this area. Ultimately, the findings in this field will shape our understanding of programming as a skill.

Appendix A

Table with item details that were used to split the SCS1 programming test into two versions

Table A.1 Items assigned to SCS1-S1 with the level of difficulty as estimated according to the first validation of the SCS1 by Parker et al. (2016) and according to our pilot work.

Question number SCS1-S1	Original question from SCS1	Topic	Type of question	Difficulty according to Parker et al. (2016) 0-0.50 = Hard 0.50-0.85 = Moderate >0.85 = Easy	Difficulty according to our pilot study	Level of discrimination according to Parker et al. (2016) <0.1 = poor 0.1-0.3 = fair >0.3 = good	Level of Discrimination according to our pilot study
				SCS1-S1			
1	1	for	definitional	moderate	.18	good	.13
2	2	logical operator	tracing	moderate	.41	good	.29**
3	5	function return values	tracing	hard	.12	poor	.14
4	6	if	definitional	hard	.46	fair	.31**
5	9	while	tracing	hard	.29	fair	.43***
6	12	basics	definitional	hard	.50	fair	.42***
7	13	for	code completion	hard	.16	fair	.18
8	14	recursion	definitional	hard	.35	good	.49***
9	17	arrays	code completion	hard	.23	fair	.50***
10	18	recursion	code completion	hard	.22	poor	.18
11	22	arrays	tracing	hard	.32	fair	.54***
12	25	basics	code completion	hard	.46	fair	.60***

13	27	function parameters	tracing	hard	.18	poor	.23
				SCS1-S2			
1	3	while	definitional	moderate	.61	good	.39***
2	4	arrays	definitional	hard	.23	fair	.46***
3	7	while	code completion	hard	.23	fair	.52***
4	8	for	Tracing	hard	.33	poor	.37***
5	10	logical operator	definitional	hard	.45	fair	.33***
6	11	function return values	definitional	hard	.20	fair	.21*
7	16	function parameters	code completion	hard	.21	fair	.45***
8	19	if	tracing	moderate	.52	good	.58***
9	20	function parameters	definitional	hard	.16	poor	.24*
10	21	if	code completion	hard	.29	fair	.55***
11	23	basics	tracing	moderate	.57	fair	.62***
12	24	recursion	tracing	hard	.18	poor	.15
13	26	logical operator	code completion	hard	.28	fair	.39***

Note: Difficulty is defined as the proportion of correct attempts for each question. Discriminability is defined by the point biserial correlation between the score on each item and total SCS1-S score**.** Unanswered questions were counted as incorrect.

Appendix B

Stimuli for the Event Related Potential experiment

Experimental stimuli

Dutch

Grammatical

List 1	**List 2**
Het heeft veel tijd nodig.	Het deed er niet toe.
Hij doet een goede zet.	Het kan elk moment gebeuren.
Hij is een slechte kok.	Hij durft alles te vragen.
Hij kon slecht vogels vangen.	Hij heeft een gebroken been.
Ik doe dit nooit meer.	Hij kan goed bloemen tekenen.
Ik durf alleen te reizen.	Hij was een goede fotograaf.
Ik kan de was doen.	Ik ben een goede student.
Ik was een geweldige docent.	Ik heb veel mooie verhalen.
Jij bent een aardige jongen.	Ik kan goed aardappels schillen.
Jij deed dat elke dag.	Jij doet altijd zo gek.
Jij kunt niet goed duiken.	Jij durft dit te bespreken.
Jullie doen heel belangrijk werk.	Jij hebt even pauze nodig.
Jullie durven wel te springen.	Jij hebt veel dure kleding.
Jullie hebben een lange vakantie.	Jij kon heel goed rekenen.
Wij durven morgen te protesteren.	Jullie hadden veel films gemaakt.
Wij hadden vroeger een paard.	Jullie kunnen heel goed dansen.
Wij hebben een mooi standbeeld.	Jullie zijn een gezellige club.
Wij konden goed huiswerk maken.	Wij deden het moeilijke examen.
Wij kunnen dat heel goed.	Wij doen ons werk dagelijks.
Zij is een goede zwemster.	Wij zijn een beetje moe.

Ungrammatical

List 1	**List 2**
Het deden er niet toe.	Het hebben veel tijd nodig.
Het kunnen elk moment gebeuren.	Hij bent een slechte kok.
Hij durven alles te vragen.	Hij doe een goede zet.
Hij hebben een gebroken been.	Hij konden slecht vogels vangen.
Hij kunt goed bloemen tekenen.	Ik doet dit nooit meer.
Hij waren een goede fotograaf.	Ik durven alleen te reizen.
Ik bent een goede student.	Ik kunt de was doen.
Ik hebben veel mooie verhalen.	Ik waren een geweldige docent.
Ik kunnen goed aardappels schillen.	Jij deden dat elke dag.
Jij doe altijd zo gek.	Jij kunnen niet goed duiken.
Jij durven dit te bespreken.	Jij zijn een aardige jongen.
Jij heeft even pauze nodig.	Jullie doet heel belangrijk werk.
Jij heeft veel dure kleding.	Jullie durft wel te springen.
Jij konden heel goed rekenen.	Jullie heeft een lange vakantie.
Jullie had veel films gemaakt.	Wij durft morgen te protesteren.
Jullie is een gezellige club.	Wij had vroeger een paard.
Jullie kan heel goed dansen.	Wij heeft een mooi standbeeld.
Wij deed het moeilijke examen.	Wij kan dat heel goed.
Wij doet ons werk dagelijks.	Wij kon goed huiswerk maken.
Wij is een beetje moe.	Zij ben een goede zwemster.

English

Grammatical

List 1	**List 2**
He is an expert programmer.	He dares to jump far.
He needs to practice drawing.	He does the dirty dishes.
He was a clever journalist.	He has a rare disease.
I am an excellent painter.	I dare to cycle fast.
I have many tasty snacks.	I do a silly dance.
I need some more potatoes.	I was a terrible singer.
It dares to sneak inside.	It does that every day.
It needs to happen now.	It has a long tail.
She dares to say anything.	It is a bad choice.
She does her stretches daily.	She is a great dancer.
She has a big nose.	She needs a long holiday.
They have many famous books.	They are a friendly group.
They need a short break.	They dare to climb high.
They were always very mean.	They do their difficult task.
We are a bit late.	We dare to fight them.
We do the expensive groceries.	We have a beautiful painting.
You dare to present this.	We need a strong coffee.
You do all the chores.	We were a great team.
You have so many shoes.	You are a good friend.
You were a bad swimmer.	You need to eat more.

Ungrammatical

List 1	**List 2**
He dare to jump far.	He are an expert programmer.
He do the dirty dishes.	He need to practice drawing.
He have a rare disease.	He were a clever journalist.
I dares to cycle fast.	I has many tasty snacks.
I does a silly dance.	I is an excellent painter.
I were a terrible singer.	I needs some more potatoes.
It are a bad choice.	It dare to sneak inside.
It do that every day.	It need to happen now.
It have a long tail.	She dare to say anything.
She are a great dancer.	She do her stretches daily.
She need a long holiday.	She have a big nose.
They am a friendly group.	They has many famous books.
They dares to climb high.	They needs a short break.
They does their difficult task.	They was always very mean.
We dares to fight them.	We am a bit late.
We has a beautiful painting.	We does the expensive groceries.
We needs a strong coffee.	You dares to present this.
We was a great team.	You does all the chores.
You is a good friend.	You has so many shoes.
You needs to eat more.	You was a bad swimmer.

Java

Grammatical

List 1	List 2
if (a>=5)	if (a<10)
if (a>2)	if (a==100)
if (b!=1)	if (b<=107)
if (b==40)	if (b<22)
if (i<7)	if (i!=10)
if (j==110)	if (i==2)
if (x<=23)	if (x<27)
if (x<20)	if (x>5)
if (z!=14)	if (z!=0)
if (z<32)	if (z>=2)
while (a!=6)	while (a==0)
while (b==4)	while (b!=4)
while (j>15)	while (b==10)
while (k>5)	while (i==7)
while (p==55)	while (i>1)
while (v<2)	while (j<20)
while (x<26)	while (j<22)
while (x==0)	while (k>=50)
while (y<=6)	while (x<10)
while (z>37)	while (x>0)

Ungrammatical

List 1	**List 2**
if {a<10}	if {a>=5}
if {a==100}	if {a>2}
if {b<=107}	if {b!=1}
if {b<22}	if {b==40}
if {i!=10}	if {i<7}
if {i==2}	if {j==110}
if {x<27}	if {x<=23}
if {x>5}	if {x<20}
if {z!=0}	if {z!=14}
if {z>=2}	if {z<32}
while {a==0}	while {a!=6}
while {b!=4}	while {b==4}
while {b==10}	while {j>15}
while {i==7}	while {k>5}
while {i>1}	while {p==55}
while {j<20}	while {v<2}
while {j<22}	while {x<26}
while {k>=50}	while {x==0}
while {x<10}	while {y<=6}
while {x>0}	while {z>37}

Fillers

Dutch

Grammatical

Lists 1 and 2

Daar stroomt een grote rivier.

Dat is een officieel instituut.

Dat was een ernstig incident.

De afgekeurde manuscripten werden vernietigd.

De mooie etiketten worden verkocht.

De noodzakelijke vetten zijn gezond.

De turnster had een uitdaging.

De uitgebreide documenten worden besproken.

De verplichte formulieren liggen daar.

Er hing een zeldzame viool.

Er is een giftig gas.

Er ontstond een acute crisis.

Het huidige adres is onbekend.

Het ingewikkelde recept is lekker.

Het moderne concert klonk goed.

Het nieuwe paspoort is geldig.

Hij gebruikte een klassiek instrument.

Hij ontdekte een nieuw continent.

Hij verdiende een goed salaris.

Zij heeft een indrukwekkend fort.

Ungrammatical

Lists 1 and 2

Daar is een diepe meer.

Dat was een gevaarlijke park.

De lang touwen hangen daar.

De leeg nesten waren verspreid.

De radicaal fronten bevochten elkaar.

Er is een nieuw klok.

Er ligt een ongebruikte apparaat.

Er stond een hoge hek.

Het beschadigde netten werden vervangen.

Het bot mes werkt niet.

Het donkere vlakken moeten gewit.

Het moeilijke vakken werden gehaat.

Het piepende wielen werden gesmeerd.

Het was een grote avontuur.

Hier is een lange lint.

Hij gaf een grote compliment.

Hij heeft een sterk zoon.

Hij verliet een prachtige dorp.

Zij bezochten een prachtige plein.

Zij sloten een geheime pact.

English

Grammatical

List 1 and 2

Everyone likes to receive pay.

He loved magic a lot.

He played with the sand.

His dad earns some money.

I can't follow the logic.

I have so much homework.

I should gather my courage.

I was quickly granted permission.

It is a simple question.

It must have been tourism.

It's relatively easy to win.

Plants don't stop producing oxygen.

She handled most financial tasks.

She has a strong faith.

She has knowledge of health.

She never drinks enough water.

The hotel offered cheap transportation.

The professor discussed some research.

There was not much evidence.

We don't rely on agriculture.

Ungrammatical

List 1 and 2

A good health is important.

He didn't make a progress.

His leader demanded a progress.

I have never played hockeys.

I ran a finance today.

I will offer an advice.

I've access to an information.

My laptop generates a heats.

She packed all the luggages.

She really values a privacy.

She shows a good participation.

That is an important knowledge.

That was very much funs.

The car had a damage.

The greeds is not good.

The informations are not correct.

The team made many progresses.

The weathers have been terrible.

They perform maintenances on roads.

They stood on some gravels.

Java

Grammatical

List 1 and 2

else {j=i}

while (z==b)

if (b>=-5)

else {x=z}

else {x=2}

else {i=j}

while (y!=x)

else {j=100}

while (x=<i)

else {v=-5}

else {v=0}

if (t==x)

else {y=1}

if (a==x)

else {i=0}

else {j=28}

else {v=1}

else {a=b}

else {x=10}

else {y=13}

Ungrammatical

List 1 and 2

else (i=x)

else (v=100)

else (x==y)

else (x=0)

else (z==5)

else (z=y)

else {i>=14}

else {j==-10}

else {v<5}

else {x>z}

if (a=2)

if (b=-1)

if (j>>3)

if (x=40)

if (y=<10)

while (i=1)

while (x<>6)

while (y=-10)

while (z<<0)

while (z>=<23)

Appendix C

Tables with all ANOVA and post hoc *t*-test results

Table C.1. Results of repeated measures ANOVAs for Dutch language stimuli.

Time window	200-400	400-600	600-800	800-1000
		Lateral regions		
Grammaticality	$F(11) = 10.591$, $p = .008$**, $\eta^2G = .038$	$F(11) = 3.641$, $p = .083$, $\eta^2G = .039$	$F(11) = 15.198$, $p = .002$**, $\eta^2G = .208$	$F(11) = 6.304$, $p = .029$*, $\eta^2G = .051$
Grammaticality*anteriority	$F(22) = .517$, $p = .511$, $\eta^2G = .002$	$F(22) = 12.076$, $p < .001$***, $\eta^2G = .048$	$F(22) = 10.883$, $p < .001$***, $\eta^2G = .026$	$F(22) = 6.452$, $p = .006$**, $\eta^2G = .018$
Grammaticality*hemisphere	$F(11) = 0.049$, $p = .829$, $\eta^2G < .001$	$F(11) = 3.631$, $p = .083$, $\eta^2G = .014$	$F(11) = 1.483$, $p = .249$, $\eta^2G = .006$	$F(11) = .443$, $p = 520$, $\eta^2G = .001$
		Midline regions		
Grammaticality	$F(11) = 4.648$, $p = .054$, $\eta^2G = .040$	$F(11) = 2.133$, $p = .172$, $\eta^2G = .049$	$F(11) = 9.972$, $p = .009$**, $\eta^2G = .281$	$F(11) = 8.228$, $p = .015$*, $\eta^2G = .137$
Grammaticality*anteriority	$F(22) = 0.586$, $p = .472$, $\eta^2G = .003$	$F(22) = 8.079$, $p = .013$*, $\eta^2G = .040$	$F(22) = 2.581$, $p = .134$, $\eta^2G = .013$	$F(22) = 8.656$, $p = .009$**, $\eta^2G = .025$

Note: The top part of the table shows the results for the ANOVAs with the lateral regions of interest and had grammaticality (2 levels: grammatical and ungrammatical), hemisphere (2 levels: left and right hemisphere), and anteriority (3 levels: anterior, central, and posterior) as between subject factors. The second part of the table shows the repeated measures ANOVAs only looked at the regions of interest in the midline and had grammaticality (2 levels: grammatical and ungrammatical), and anteriority (3 levels: anterior, central, and posterior). ANOVAs were run per 200ms time increment from 200-1000ms. As an effect size statistic for the ANOVA analyses, we report generalised eta squared (η2G). *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.2. Results of post hoc *t*-tests for the interaction between grammaticality and anteriority and grammaticality and hemisphere for the Dutch language stimuli.

Time window	200-400ms	400-600ms	600-800ms	800-1000ms
		Lateral regions		
Grammaticality * Anteriority				
LA	X	$t(11) = -3.400$, $p = .006$**	$t(11) = -5.287$, $p < .001$***	$t(11) = -0.801$, $p = .440$
LC	X	$t(11) = -1.837$, $p = .093$	$t(11) = -3.842$, $p = .003$**	$t(11) = -2.498$, $p = .030$*
LP	X	$t(11) = 0.065$, $p = .949$	$t(11) = -1.647$, $p = .128$	$t(11) = -3.112$, $p = .010$**
RA	X	$t(11) = -1.507$, $p = .160$	$t(11) = -3.833$, $p = .003$**	$t(11) = 0.399$, $p = .698$
RC	X	$t(11) = -1.091$, $p = .299$	$t(11) = -3.174$, $p = .009$**	$t(11) = -1.956$, $p = .076$
RP	X	$t(11) = 1.042$, $p = .320$	$t(11) = -1.850$, $p = .091$	$t(11) = -4.099$, $p = .002$***
Grammaticality * Hemisphere				
Left	X	$t(11) = -2.239$, $p = .047$*	X	X
Right	X	$t(11) = -2.147$, $p = .055$	X	X
		Midline regions		
Grammaticality * Anteriority				
MA	X	$t(11) = -2.851$, $p = .016$*	X	$t(11) = -1.043$, $p = .320$
MC	X	$t(11) = -1.202$, $p = .255$	X	$t(11) = -3.022$, $p = .012$*
MP	X	$t(11) = 0.014$, $p = .989$	X	$t(11) = -4.779$, $p < .001$***

Note: Post hoc *t*-tests were only performed for time windows where an interaction was found between grammaticality and anteriority or grammaticality and hemisphere. *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.3. Results of repeated measures ANOVAs for English language stimuli.

Time window	200-400	400-600	600-800	800-1000
		Lateral regions		
Grammaticality	$F(11) = 0.274$, $p = .6112$, $\eta 2G = .002$	$F(11) = 0.390$, $p = .545$, $\eta 2G = .005$	$F(11) = 11.341$, $p = .006$**, $\eta 2G = .188$	$F(11) = 5.535$, $p = .038$*, $\eta 2G = .079$
Grammaticality*anteriority	$F(22) = 0.500$, $p = .523$, $\eta 2G = .002$	$F(22) = 0.620$, $p = .472$, $\eta 2G = .003$	$F(22) = 0.341$, $p = .579$, $\eta 2G = .002$	$F(22) = 12.735$, $p = .002$**, $\eta 2G = .063$
Grammaticality*hemisphere	$F(11) = 0.091$, $p = .768$, $\eta^2G < .001$	$F(11) = 3.974$, $p = .072$, $\eta 2G = .011$	$F(11) = 3.036$, $p = .109$, $\eta 2G = .016$	$F(11) = 0.215$, $p = .652$, $\eta 2G = .279$
		Midline regions		
Grammaticality	$F(11) = 0.003$, $p = .955$, $\eta^2G < .001$	$F(11) = 0.353$, $p = .564$, $\eta 2G = .007$	$F(11) = 13.342$, $p = .004$**, $\eta 2G = .296$	$F(11) = 10.428$, $p = .008$**, $\eta 2G = .205$
Grammaticality*anteriority	$F(22) = 0.916$, $p = .373$, $\eta 2G = .005$	$F(22) = 0.730$, $p = .425$, $\eta 2G = .007$	$F(22) = 0.715$, $p = .428$, $\eta 2G = .005$	$F(22) = 13.739$, $p = .002$**, $\eta 2G = .090$

Note: The top part of the table shows the results for the ANOVAs with the lateral regions of interest and had grammaticality (2 levels: grammatical and ungrammatical), hemisphere (2 levels: left and right hemisphere), and anteriority (3 levels: anterior, central, and posterior) as between subject factors. The second part of the table shows the repeated measures ANOVAs only looked at the regions of interest on the midline and had grammaticality (2 levels: grammatical and ungrammatical), and anteriority (3 levels: anterior, central, and posterior). ANOVAs were run per 200ms time increment from 200-1000 ms. As an effect size statistic for the ANOVA analyses, we report generalised eta squared (η2G). *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.4. Results of post hoc *t*-tests for the interaction between grammaticality and anteriority and grammaticality and hemisphere for the English language stimuli.

Time window	200-400ms	400-600ms	600-800ms	800-1000ms
		Lateral		
Grammaticality * Anteriority				
LA	X	X	X	$t(11) = -0.516$, $p = .616$
LC	X	X	X	$t(11) = -2.022$, $p = .068$
LP	X	X	X	$t(11) = -4.409$, $p = .001$**
RA	X	X	X	$t(11) = 0.842$, $p = .418$
RC	X	X	X	$t(11) = -1.917$, $p = .082$
RP	X	X	X	$t(11) = -4.339$, $p = .002$**
Grammaticality * Hemisphere				
Left	X	$t(11) = -2.507$, $p = .029$*	X	X
Right	X	$t(11) = -1.635$, $p = .130$	X	X
		Midline		
Grammaticality * Anteriority				
MA	X	X	X	$t(11) = -0.407$, $p = .692$
MC	X	X	X	$t(11) = -3.585$, $p = .004$**
MP	X	X	X	$t(11) = -5.249$, $p < .001$**

Note: Post hoc *t*-tests were only performed for time windows where an interaction was found between grammaticality and anteriority or grammaticality and hemisphere. *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.5. Results of repeated measures ANOVAs for Java programming language stimuli.

Time window	200-400	400-600	600-800	800-1000
		Lateral regions		
Grammaticality	$F(11) = 5.444$, $p = .040$*, $\eta 2G = .050$	$F(11) = 15.200$, $p = .002$**, $\eta 2G = .140$	$F(11) = 4.327$, $p = .062$, $\eta 2G = .052$	$F(11) = .191$, $p = .671$, $\eta 2G = .003$
Grammaticality*anteriority	$F(22) = .595$, $p = .471$, $\eta 2G = .003$	$F(22) = 3.478$, $p = .078$, $\eta 2G = .019$	$F(22) = 1.287$, $p = .286$, $\eta 2G = .008$	$F(22) = 1.164$, $p = .314$, $\eta 2G = .009$
Grammaticality*hemisphere	$F(11) = .438$, $p = .522$, $\eta 2G = .001$	$F(11)<.001$, $p = .989$, $\eta^2G < .001$	$F(11) = 1.264$, $p = .285$, $\eta 2G = .002$	$F(11) = .166$, $p = .692$, $\eta 2G = .001$
		Midline regions		
Grammaticality	$F(11) = 3.519$, $p = .087$, $\eta 2G = .059$	$F(11) = 10.890$, $p = .007$**, $\eta 2G = .184$	$F(11) = 2.601$, $p = .135$, $\eta 2G = .053$	$F(11) = .070$, $p = .796$, $\eta 2G = .002$
Grammaticality*anteriority	$F(22) = 1.282$, $p = .292$, $\eta 2G = .003$	$F(22) = 14.160$, $p = .001$**, $\eta 2G = .037$	$F(22) = 6.224$, $p = .007$**, $\eta 2G = .015$	$F(22) = 3.577$, $p = .074$, $\eta 2G = .024$

Note: The top part of the table shows the results for the ANOVAs with the lateral regions of interest and had grammaticality (2 levels: grammatical and ungrammatical), hemisphere (2 levels: left and right hemisphere), and anteriority (3 levels: anterior, central, and posterior) as between subject factors. The second part of the table shows the repeated measures ANOVAs only looked at the regions of interest on the midline and had grammaticality (2 levels: grammatical and ungrammatical), and anteriority (3 levels: anterior, central, and posterior). ANOVAs were run per 200ms time increment from 200-1000 ms. As an effect size statistic for the ANOVA analyses, we report generalised eta squared (η2G). *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.7. Results of post hoc *t*-tests for the interaction between grammaticality and anteriority and grammaticality and hemisphere for the Java programming language stimuli.

Time window	200-400ms	400-600ms	600-800ms	800-1000ms
		Lateral		
Grammaticality * Anteriority				
LA	X	$t(11) = -4.548$, $p < .001$***	X	X
LC	X	$t(11) = -2.985$, $p = .012$*	X	X
LP	X	$t(11) = -1.413$, $p = .186$	X	X
RA	X	$t(11) = -3.218$, $p = .008$**	X	X
RC	X	$t(11) = -3.842$, $p = .003$**	X	X
RP	X	$t(11) = -1.667$, $p = .124$	X	X
Grammaticality * Hemisphere				
Left	X	X	X	X
Right	X	X	X	X
		Midline		
Grammaticality * Anteriority				
MA	X	$t(11) = -4.089$, $p = .002$**	$t(11) = -2.915$, $p = .014$*	$t(11) = -1.292$, $p = .223$
MC	X	$t(11) = -2.636$, $p = .023$*	$t(11) = -1.300$, $p = .220$	$t(11) = 0.378$, $p = .713$
MP	X	$t(11) = -2.441$, $p = .033$*	$t(11) = -0.613$, $p = .553$	$t(11) = 1.014$, $p = .333$

Note: Post hoc *t*-tests were only performed for time windows where an interaction was found between grammaticality and anteriority or grammaticality and hemisphere. *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.8. Results of repeated measures ANOVAs with the difference scores between the grammatical and ungrammatical stimuli for each language.

Time window	200-400	400-600	600-800	800-1000
		Lateral regions		
Language	$F(22) = 1.227$, $p = .312$, $\eta 2G = .039$	$F(22) = 5.820$, $p = .009$**, $\eta 2G = .146$	$F(22) = 2.719$, $p = .088$, $\eta 2G = .055$	$F(22) = 1.564$, $p = .232$, $\eta 2G = .044$
Language*anteriority	$F(44) = 1.016$, $p = .380$, $\eta 2G = .014$	$F(44) = 4.716$, $p = .020$*, $\eta 2G = .055$	$F(44) = 4.420$, $p = .004$**, $\eta 2G = .025$	$F(44) = 6.400$, $p = .005$**, $\eta 2G = .069$
Language*hemisphere	$F(22) = .098$, $p = .907$, $\eta 2G = .001$	$F(22) = 1.450$, $p = .256$, $\eta 2G = .012$	$F(22) = 2.308$, $p = .123$, $\eta 2G = .018$	$F(22) = .254$, $p = .778$, $\eta 2G = .004$
		Midline regions		
Language	$F(22) = 1.270$, $p = .301$, $\eta 2G = .062$	$F(22) = 3.098$, $p = .065$, $\eta 2G = .088$	$F(22) = 2.736$, $p = .087$, $\eta 2G = .076$	$F(22) = 4.331$, $p = .026$*, $\eta 2G = .162$
Language*anteriority	$F(44) = 1.419$, $p = .262$, $\eta 2G = .018$	$F(44) = 6.017$, $p = .017$*, $\eta 2G = .054$	$F(44) = 4.816$, $p = .012$*, $\eta 2G = .020$	$F(44) = 10.180$, $p < .001$***, $\eta 2G = .101$

Note: The top part of the table shows the results for the ANOVAs with the lateral regions of interest and had language (3 levels: Dutch, English and Java), hemisphere (2 levels: left and right hemisphere), and anteriority (3 levels: anterior, central, and posterior) as between subject factors. The second part of the table shows the repeated measures ANOVAs only looked at the regions of interest on the midline and had language (3 levels: Dutch, English and Java), and anteriority (3 levels: anterior, central, and posterior). ANOVAs were run per 200ms time increment from 200-1000 ms. As an effect size statistic for the ANOVA analyses, we report generalised eta squared (η2G). *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

Table C.9. Results of post hoc *t*-tests for the main effects of language, the interactions between language and anteriority and language and hemisphere for the for the difference scores between grammatical and ungrammatical stimuli for each language.

Time window		200-400ms	400-600ms	600-800ms	800-1000ms
			Lateral		
Main					
	Dutch vs Java	X	$t(71) = -2.896$, $p = .015$*	$t(71) = 4.002$, $p < .001$***	X
	English vs Java	X	$t(71) = -5.637$, $p < .001$***	$t(71) = 2.853$, $p = .018$*	X
	Dutch vs English	X	$t(71) = 3.301$, $p = .005$**	$t(71) = .738$, $p = .999$	X
Grammaticality * Anteriority					
LA	Dutch vs Java	X	$t(11) = .221$, $p = .999$	$t(11) = 2.873$, $p = .045$*	$t(11) = 0.232$, $p = .999$
	English vs Java	X	$t(11) = -2.131$, $p = .170$	$t(11) = 1.901$, $p = .251$	$t(11) = -0.083$, $p = .999$
	Dutch vs English	X	$t(11) = 2.209$, $p = .150$	$t(11) = 0.922$, $p = .999$	$t(11) = 0.346$, $p = .999$
RA	Dutch vs Java	X	$t(11) = -1.084$, $p = .904$	$t(11) = 1.503$, $p = .480$	$t(11) = -0.997$, $p = .999$
	English vs Java	X	$t(11) = -3.316$, $p = .021$*	$t(11) = -0.756$, $p = .999$	$t(11) = -1.102$, $p = .880$
	Dutch vs English	X	$t(11) = 3.274$, $p = .022$*	$t(11) = 2.407$, $p = .100$	$t(11) = 0.565$, $p = .999$
LC	Dutch vs Java	X	$t(11) = -0.844$, $p = .999$	$t(11) = 2.344$, $p = .120$	$t(11) = 2.034$, $p = .200$
	English vs Java	X	$t(11) = -2.823$, $p = .050$	$t(11) = 2.077$, $p = .190$	$t(11) = 1.531$, $p = .460$
	Dutch vs English	X	$t(11) = 1.467$, $p = .510$	$t(11) = 0.460$, $p = .999$	$t(11) = 0.633$, $p = .999$
RC	Dutch vs Java	X	$t(11) = -3.580$, $p = .013$*	$t(11) = 0.978$, $p = .999$	$t(11) = 0.768$, $p = .999$
	English vs Java	X	$t(11) = -3.259$, $p = .023$*	$t(11) = 0.351$, $p = .999$	$t(11) = 0.779$, $p = .999$

	Dutch vs English	X	$t(11) = 1.597$, $p = .415$	$t(11) = 0.498$, $p = .999$	$t(11) = -0.162$, $p = .999$
LP	Dutch vs Java	X	$t(11) = -1.139$, $p = .840$	$t(11) = 1.128$, $p = .849$	$t(11) = 1.830$, $p = .284$
	English vs Java	X	$t(11) = -0.646$, $p = .999$	$t(11) = 2.683$, $p = .064$	$t(11) = 2.953$, $p = .039$*
	Dutch vs English	X	$t(11) = -0.671$, $p = .999$	$t(11) = -1.922$, $p = .243$	$t(11) = -1.048$, $p = .951$
RP	Dutch vs Java	X	$t(11) = -2.438$, $p = .099$	$t(11) = 0.857$, $p = .999$	$t(11) = 1.914$, $p = .250$
	English vs Java	X	$t(11) = -1.607$, $p = .409$	$t(11) = 1.447$, $p = .530$	$t(11) = 2.300$, $p = .130$
	Dutch vs English	X	$t(11) = -0.508$, $p = .999$	$t(11) = -2.007$, $p = .210$	$t(11) = -1.707$, $p = .350$
Hemisphere * Anteriority					
Left	Dutch vs Java	X	X	X	X
	English vs Java	X	X	X	X
	Dutch vs English	X	X	X	X
Right	Dutch vs Java	X	X	X	X
	English vs Java	X	X	X	X
	Dutch vs English	X	X	X	X
			Midline		
Main					
	Dutch vs Java	X	$t(35) = -2.527$, $p = .048$*	$t(35) = 3.660$, $p = .003$**	$t(35) = 3.379$, $p = .005$**
	English vs Java	X	$t(35) = -3.023$, $p = .008$**	$t(35) = 2.822$, $p = .023$*	$t(35) = 3.347$, $p = .006$**
	Dutch vs English	X	$t(35) = 1.184$, $p = .734$	$t(35) = -0.065$, $p = .999$	$t(35) = -0.426$, $p = .999$
Grammaticality * Anteriority					
MA	Dutch vs Java	X	$t(11) = -1.383$, $p = .582$	$t(11) = 2.079$, $p = .190$	$t(11) = 0.373$, $p = .999$
	English vs Java	X	$t(11) = -2.867$, $p = .046$*	$t(11) = 0.536$, $p = .999$	$t(11) = -0.200$, $p = .999$
	Dutch vs English	X	$t(11) = 2.405$, $p = .105$	$t(11) = 1.211$, $p = .750$	$t(11) = 0.662$, $p = .999$

MC	Dutch vs Java	X	$t(11) = -1.344$, $p = .620$	$t(11) = 1.894$, $p = .250$	$t(11) = 2.368$, $p = .112$
	English vs Java	X	$t(11) = -1.894$, $p = .250$	$t(11) = 1.589$, $p = .420$	$t(11) = 2.578$, $p = .077$
	Dutch vs English	X	$t(11) = 0.669$, $p = .999$	$t(11) = -0.350$, $p = .999$	$t(11) = -0.575$, $p = .999$
MP	Dutch vs Java	X	$t(11) = -1.555$, $p = .440$	$t(11) = 2.188$, $p = .153$	$t(11) = 3.082$, $p = .031$*
	English vs Java	X	$t(11) = -0.542$, $p = .999$	$t(11) = 3.092$, $p = .031$*	$t(11) = 4.002$, $p = .006$**
	Dutch vs English	X	$t(11) = -1.182$, $p = .790$	$t(11) = -1.195$, $p = .772$	$t(11) = -1.030$, $p = .976$

Note: Post hoc *t*-tests were only performed for time windows where a significant or marginally significant main effect of language was found, or if there was a significant or marginally significant interaction between grammaticality and anteriority or grammaticality and hemisphere. The table presents *p*-values after correction for multiple comparisons with Bonferroni corrections. *indicates *p*-value below .05, ** indicates *p*-value below .01, *** indicates *p*-value below .001.

References

References

Aho, A. V., Lam, M. S., Sethi, R., & Ullman, J. D. (2007). *Compilers: Principles, techniques, and tools. Second edition*. New York: Pearson Education Addison Wesley.

American Psychiatric Association. (2013). *Diagnostic and statistical manual of mental disorders* (5th ed.). Arlington, VA: Author.

Armoni, M. (2013). On teaching abstraction in CS to novices. *Journal of Computers in Mathematics and Science Teaching*, *32*(3), 265-284.

Austin, E. J. (2005). Personality correlates of the broader autism phenotype as assessed by the Autism Spectrum Quotient (AQ). *Personality and Individual Differences*, *38*(2), 451–460. https://doi.org/10.1016/j.paid.2004.04.022

Bailey, J., & Mitchell, R. B. (2006). Industry perceptions of the competencies needed by computer programmers: technical, business, and soft skills. *Journal of Computer Information Systems*, *47*(2), 28-33. https://doi.org/10.1080/08874417.2007.-11645951

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

Baron-Cohen, S. (2006). The hyper-systemizing, assortative mating theory of autism. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, *30*(5), 865–872. https://doi.org/10.1016/j.pnpbp.2006.01.010

Baron-Cohen, S. (2008). *Autism and Asperger syndrome*. Oxford, UK: Oxford University Press.

Baron-Cohen, S. (2012). Autism and the technical mind. *Scientific American, 307*(5), 72–75.

Baron-Cohen, S., Wheelwright, S., Burtenshaw, A., & Hobson, E. (2007). Mathematical Talent is Linked to Autism. *Human Nature*, *18*(2), 125–131. https://doi.org/10.1007/s12110-007-9014-0

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

Barrick, M. R., Mount, M. K., & Judge, T. A. (2001). Personality and performance at the beginning of the new millennium: What do we know and where do we go next? *International Journal of Selection and Assessment*, *9*(1 & 2), 9–30. https://doi.org/10.1111/1468-2389.00160

Bednarik, R., & Tukiainen, M. (2006). An eye-tracking methodology for characterizing program comprehension processes. *Proceedings of the 2006 symposium on eye tracking research & applications*, 125-132. https://doi.org/10.1145/1117309.1117356

Bergin Jr, T. J., & Gibson Jr, R. G. (1996). *History of programming languages---II*. New York, NY: ACM Press and Addison-Wesley. https://doi.org/10.1145/234286

Borzovs, J., Kozmina, N., Niedrite, L., Solodovnikova, D., Straujums, U., Zuters, J., & Klavins, A. (2017). Can SQ and EQ Values and Their Difference Indicate Programming Aptitude to Reduce Dropout Rate? In M. Kirikova, K. Nørvåg, G. A. Papadopoulos, J. Gamper, R. Wrembel, J. Darmont, & S. Rizzi (Eds.), *New Trends in Databases and Information Systems 767,* 285–293. Cham, CH: Springer International Publishing. https://doi.org/10.1007/978-3-319-67162-8_28

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies, 9*(6), 737-751. doi:https://doi.org/10.1016/S0020-7373(77)80039-4

Brooks, R. (1978). Using a behavioral theory of program comprehension in software engineering. *Proceedings of the 3rd International Conference on Software Engineering,* 196-201.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal Man-Machine Studies, 18*(6), 543-554. doi:https://doi.org/10.1016/S0020-7373(83)80031-5

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini-languages: a way to learn programming principles. *Education and Information Technologies*, *2*(1), 65-83. https://doi.org/10.1023/A:1018636507883

Carreiras, M., & Clifton Jr, C. (2004). *The on-line study of sentence comprehension: Eyetracking, ERPs and beyond*. New York, NY: Psychology Press.

Catherine, B.-C., & Wheeler, D. D. (1994). The Myers-Briggs Personality Type and Its Relationship to Computer Programming. *Journal of Research on Computing in Education*, *26*(3), 358–370. https://doi.org/10.1080/08886504.1994.10782096

Chen, L., Shu, H. U. A., Liu, Y., Zhao, J., & Li, P. (2007). ERP signatures of subject–verb agreement in L2 learning. *Bilingualism: Language and Cognition*, *10*(2), 161-174. https://doi.org/10.1017/S136672890700291X

Coles, M., & Phalp, K. T. (2016). *Brain type as a programming aptitude predictor.* PPIG 2016 – 27th Annual Workshop.

Coulson, S., King, J. W., & Kutas, M. (1998). Expect the unexpected: Event-related brain response to morphosyntactic violations. *Language and Cognitive Processes*, *13*(1), 21-58. https://doi.org/10.1080/016909698386582

Dasgupta, S. (2014). *It began with Babbage: The genesis of computer science*. Oxford, UK: Oxford University Press.

den Houting, J. (2019). Neurodiversity: An insider's perspective. *Autism, 23*(2), 271–273. https://doi.org/10.1177/1362361318820762

Donchin, E. (1981). Surprise! . . . Surprise? *Psychophysiology*, *18*, 493–513. https://doi.org/10.1111/j.1469-8986.1981.tb01815.x

European commission (2018). *Communication from the commission to the European parliament, the council, the European economic and social committee and the committee of the regions on the Digital Education Action Plan.* https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=COM%3A2018%3A22%3AFIN

Europees Referentiekader Talen. (n.d.). Niveaus havo/vwo. https://erk.nl/docent/streefniveaus/havo/

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

Essinger, J. (2004). *Jacquard's web: How a hand-loom led to the birth of the information age*. Oxford, UK: OUP Oxford.

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)*, 175-186. IEEE. https://doi.org/10.1109/ICSE.2017.24

Focquaert, F., Steven, M. S., Wolford, G. L., Colden, A., & Gazzaniga, M. S. (2007). Empathizing and systemizing cognitive traits in the sciences and humanities. *Personality and Individual Differences*, *43*(3), 619-625. https://doi.org/10.1016/j.paid.2007.01.004

Friederici, A. D. (1995). The time course of syntactic activation during language processing: A model based on neuropsychological and neurophysiological data. *Brain and Language*, *50*(3), 259-281. https://doi.org/10.1006/brln.1995.1048

Friederici, A.D. (2002). Towards a neural basis of auditory sentence processing. *Trends in Cognitive Sciences, 6*(2), 78-84. https://doi.org/10.1016/S1364-6613(00)01839-8

Friederici, A. D., Hahne, A., & Saddy, D. (2002). Distinct neurophysiological patterns reflecting aspects of syntactic complexity and syntactic repair. *Journal of Psycholinguistic Research*, *31*(1), 45-63. https://doi.org/10.1023/A:1014376204525

Fuegi, J., & Francis, J. (2003). Lovelace & Babbage and the creation of the 1843 'notes'. *IEEE Annals of the History of Computing*, *25*(4), 16-26. https://doi.org/10.1109/MAHC.2003.1253887

Gazzaniga, M.S., Ivry, R.B., & Mangin, G.R. (2009). *Cognitive neuroscience: The biology of the mind* (3rd ed.). New York, NY: W.W. Norton & Company.

Golding, P., Facey-Shaw, L., & Tennant, V. (2006). Effects of peer tutoring, attitude and personality on academic performance of first year introductory programming students. *Proceedings. Frontiers in Education. 36th Annual Conference*, 7–12. https://doi.org/10.1109/FIE.2006.322662

Gosling, J., Joy, B., Steele, G., & Bracha, G. (2000). *The Java language specification*. Boston, MA: Addison-Wesley Professional.

Gouvea, A. C., Phillips, C., Kazanina, N., & Poeppel, D. (2010). The linguistic processes underlying the P600. *Language and Cognitive Processes*, *25*(2), 149-188. https://doi.org/10.1080/01690960902965951

Gray, K. E., & Flatt, M. (2003). ProfessorJ: A gradual introduction to Java through language levels. *Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, 170-177. https://doi.org/10.1145/949344.949394

Greenhouse, S. W., & Geisser, S. (1959). On methods in the analysis of profile data. *Psychometrika*, *24*(2), https://doi.org/95-112.10.1007/BF02289823

Grover, S., Jackiw, N., & Lundh, P. (2019). Concepts before coding: non-programming interactives to advance learning of introductory programming concepts in middle school. *Computer Science Education*, *29*(2-3), 106-135. https://doi.org/10.1080/08993408.2019.1568955

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

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, 8(6), 1-165. https://doi.org/10.2200/S00684ED1V01Y201511HCI033

Guzdial, M. (2019, August 26). Holding ourselves to a higher standard: “Language-independent” just doesn’t cut it. *Computing Education Research Blog.* https://computinged.wordpress.com/2019/08/26/the-temptation-in-computing-education-research-to-claim-language-independence/

Guzdial, M., & du Boulay, B. (2019). The history of computing education research. *The Cambridge handbook of computing education research* (pp. 11–39). Cambridge, UK: Cambridge University Press.

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

Hagoort, P., & Brown, C. M. (2000). ERP effects of listening to speech compared to reading: the P600/SPS to syntactic violations in spoken sentences and rapid serial visual presentation. *Neuropsychologia*, *38*(11), 1531-1549. https://doi.org/10.1016/S0028-3932(00)00053-1

Hagoort, P., Brown, C., & Groothusen, J. (1993). The syntactic positive shift (SPS) as an ERP measure of syntactic processing. *Language and Cognitive Processes*, *8*(4), 439-483. https://doi.org/10.1080/01690969308407585

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. (2020). Hedy: A Gradual Language for Programming Education. *Proceedings of the 2020 ACM Conference on International Computing Education Research*, 259-270. https://doi.org/10.1145/3372782.3406262

Hermans, F., & Aldewereld, M. (2017). Programming is writing is programming. *Companion to the First International Conference on the Art, Science and Engineering of Programming*, 1–8. https://doi.org/10.1145/3079368.3079413

Hoekstra, R. A., Bartels, M., Cath, D. C., & Boomsma, D. I. (2008). Factor Structure, Reliability and Criterion Validity of the Autism-Spectrum Quotient (AQ): A Study in Dutch Population and Patient Groups. *Journal of Autism and Developmental Disorders*, *38*(8), 1555–1566. https://doi.org/10.1007/s10803-008-0538-x

Holcomb, P. J. (1993). Semantic priming and stimulus degradation: Implications for the role of the N400 in language processing. *Psychophysiology*, *30*(1), 47-61. https://doi.org/10.1111/j.1469-8986.1993.tb03204.x

Hurst, R. M., Mitchell, J. T., Kimbrel, N. A., Kwapil, T. K., & Nelson-Gray, R. O. (2007). Examination of the reliability and factor structure of the Autism Spectrum Quotient (AQ) in a non-clinical sample. *Personality and Individual Differences*, *43*(7), 1938–1949. https://doi.org/10.1016/j.paid.2007.06.012

IBM (Ed.) 1968 *Aptitude test for programmer personnel.* New York: International Business Machines Corporation.

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'reilly, U. M., ... & Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive resources. *BioRxiv*. https://doi.org/10.1101/2020.04.16.045732

Jost, K., Henninghausen, E., & Rosler, F. (2004). Comparing arithmetic and semantic fact retrieval: Effects of problem size and sentence constraint on event-related brain potentials. *Psychophysiology, 41*, 46–59. https://doi.org/10.1111/1469-8986.00119

Kaan, E. (2002). Investigating the effects of distance and number interference in processing subject-verb dependencies: An ERP study. *Journal of Psycholinguistic Research*, *31*(2), 165-193. https://doi.org/10.1023/A:1014978917769

Kanne, S. M., Wang, J., & Christ, S. E. (2011). The Subthreshold Autism Trait Questionnaire (SATQ): Development of a brief self-report measure of subthreshold autism traits. *Journal of Autism and Developmental Disorders*, *42*(5), 769–780. https://doi.org/ 10.1007/s10803-011-1308-8

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

Kline, R. (2005). *Principles and practices of structural equation modeling (2n ed.).* New York: Guilford Press.

Kotz, S. A. (2009). A critical review of ERP and fMRI evidence on L2 syntactic processing. *Brain and Language*, *109*(2-3), 68-74. https://doi.org/10.1016/j.bandl.2008.06.002

Knuth, D. E., & Pardo, L. T. (1980). The early development of programming languages. In *A history of computing in the twentieth century* (pp. 197-273). New York, NY: Marcel Dekker. https://doi.org/10.1016/B978-0-12-491650-0.50019-8

Kutas, M., & Hillyard, S. A. (1980). Reading senseless sentences: Brain potentials reflect semantic anomaly. *Science, 207*, 203-205. https://doi.org/10.1126/science.7350657

Lance, C. E., Butts, M. M., & Michels, L. C. (2006). The sources of four commonly reported cutoff criteria: What did they really say? *Organizational Research Methods*, *9*(2), 202-220. https://doi.org/10.1177/1094428105284919

Landry, O., & Chouinard, P. A. (2016). Why we should study the broader autism phenotype in typically developing populations. *Journal of Cognition and Development*, *17*(4), 584–595. https://doi.org/10.1080/15248372.2016.1200046

Lenroot, R. K., & Yeung, P. K. (2013). Heterogeneity within autism spectrum disorders: What have we learned from neuroimaging studies? *Frontiers in Human Neuroscience*, *7*. https://doi.org/10.3389/fnhum.2013.00733

Lord, C., Cook, E. H., Leventhal, B. L., & Amaral, D. G. (2000). Autism spectrum disorders. *Neuron*, *28*(2), 355–363.

References

Lu, J. J., & Fletcher, G. H. (2009). Thinking about computational thinking. *Proceedings of the 40th ACM technical symposium on computer science education*, 260-264. https://doi.org/10.1145/1508865.1508959

Lukey, F. J. (1980). Understanding and debugging programs. *International Journal of Man-Machine Studies*, *12*(2), 189-202. https://doi.org/10.1016/S0020-7373(80)80017-4

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

Marston, D., Fuchs, L. S., & Deno, S. L. (1986). Measuring pupil progress: A comparison of standardized achievement tests and curriculum-related measures. *Diagnostique*, *11*(2), 77-90. https://doi.org/10.1177/073724778601100203

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y. B. D., ... & Wilusz, T. (2001). A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *ACM SIGCSE Bulletin 33*(4), 125-180. https://doi.org/10.1145/572133.572137

Milton, D. E. M. (2012). On the ontological status of autism: The 'double empathy problem.' *Disability & Society*, *27*(6), 883–887. https://doi.org/10.1080/09687599.2012.710008

Mitchell, P., Cassidy, S., & Sheppard, E. (2019). The double empathy problem, camouflage, and the value of expertise from experience. *Behavioral and Brain Sciences*, *42*, e100. https://doi.org/10.1017/S0140525X18002212

Molinaro, N., Vespignani, F., & Job, R. (2008). A deeper reanalysis of a superficial feature: An ERP study on agreement violations. *Brain Research*, *1228*, 161-176. https://doi.org/10.1016/j.brainres.2008.06.064

Morgan-Short, K., Steinhauer, K., Sanz, C., & Ullman, M. T. (2012). Explicit and implicit second language training differentially affect the achievement of native-like brain activation patterns. *Journal of Cognitive Neuroscience*, *24*(4), 933-947. https://doi.org/10.1162/jocn_a_00119

Mulyanto, H., Gunarhadi, G., & Indriayu, M. (2018). The effect of problem based learning model on student mathematics learning outcomes viewed from critical thinking skills. *International Journal of Educational Research Review*, *3*(2), 37-45. https://doi.org/10.24331/ijere.408454

Murray, A. L., McKenzie, K., Kuenssberg, R., & Booth, T. (2017). Do the Autism Spectrum Quotient (AQ) and Autism Spectrum Quotient Short Form (AQ-S) primarily reflect general ASD traits or specific ASD traits? A bi-factor analysis. *Assessment*. https://doi.org/10.1177/1073191115611230

Nelson, G. L., Xie, B., & Ko, A. J. (2017). Comprehension first: Evaluating a novel pedagogy and tutoring system for program tracing in CS1. *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 2-11. https://doi.org/10.1145/3105726.3106178

Nevins, A., Dillon, B., Malhotra, S., & Phillips, C. (2007). The role of feature-number and feature-type in processing Hindi verb agreement violations. *Brain Research*, *1164*, 81-94. https://doi.org/10.1016/j.brainres.2007.05.058

Nishiyama, T., Suzuki, M., Adachi, K., Sumi, S., Okada, K., Kishino, H., ... & Kanne, S. M. (2014). Comprehensive comparison of self-administered questionnaires for measuring quantitative autistic traits in adults. *Journal of Autism and Developmental Disorders*, *44*(5), 993-1007. https://doi.org/ 10.1007/s10803-013-2020-7

Oldfield, R. C. (1971). The assessment and analysis of handedness: The Edinburgh inventory. *Neuropsychologia*, *9*(1), 97-113. https://doi.org/10.1016/0028-3932(71)90067-4

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

Osterhout, L. (1999). A superficial resemblance does not necessarily mean you are part of the family: Counterarguments to Coulson, King and Kutas (1998) in the P600/SPS-P300 debate. *Language and Cognitive Processes*, *14*(1), 1-14. https://doi.org/10.1080/016909699386356

Osterhout, L., & Holcomb, P. J. (1992). Event-related brain potentials elicited by syntactic anomaly. *Journal of Memory and Language*, *31*(6), 785-806. https://doi.org/10.1016/0749-596X(92)90039-Z

Osterhout, L., & Holcomb, P. J. (1995). *Event related potentials and language comprehension.* In M. D. Rugg & M. G. H. Coles (Eds.), *Oxford psychology series, No. 25. Electrophysiology of mind: Event-related brain potentials and cognition* (p. 171–215). Oxford, UK: Oxford University Press.

Osterhout, L., McKinnon, R., Bersick, M., & Corey, V. (1996). On the language specificity of the brain response to syntactic anomalies: Is the syntactic positive shift a member of the P300 family?. *Journal of Cognitive Neuroscience*, *8*(6), 507-526.

https://doi.org/10.1162/jocn.1996.8.6.507

Pandža, N. B. (2016). Computer Programming as a Second Language. In *Advances in human factors in cybersecurity* (pp. 439–445). Cham, CH: 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*, 93-101. https://doi.org/10.1145/2960310.2960316

Parker, M. C., Solomon, A., Pritchett, B., Illingworth, D. A., Marguilieux, L. E., & Guzdial, M. (2018). Socioeconomic status and computer science achievement: Spatial ability as a mediating variable in a novel model of understanding. *Proceedings of the 2018 ACM Conference on International Computing Education Research*, 97-105. https://doi.org/10.1145/3230977.3230987

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. https://doi.org/10.1145/1151954.1067467

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

Portnoff, S. R. (2018). The introductory computer programming course is first and foremost a language course. *ACM Inroads*, *9*(2), 34-52. https://doi.org/10.1145/3152433

Potts, G. F. (2004). An ERP index of task relevance evaluation of visual stimuli. *Brain and Cognition*, *56*(1), 5-13. https://doi.org/10.1016/j.bandc.2004.03.006

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C. H. (2020). Relating natural language aptitude to individual differences in learning programming languages. *Scientific Reports*, *10*(1), 1-10. https://doi.org/10.1038/s41598-020-60661-8

Qualtrics (Version 2019). (2005). [Survey software]. Qualtrics. https://www.qualtrics.com.

Rayner, K. (2009). Eye movements in reading: Models and data. *Journal of Eye Movement Research, 2*(5), 1-10.

Robins, A., Margulieux, L. E., & Morrison, B. B. (2019). Cognitive sciences for computing education. In S. Fincher & A. Robins (Eds.), *Handbook of computing education research* (pp. 231--275). Cambridge, UK: Cambridge University Press.

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). http://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*. Available at: https://www.edutopia.org/blog/code-literacy-21st-century-requirement-douglas-rushkoff

Rossi, S., Gugler, M. F., Friederici, A. D., & Hahne, A. (2006). The impact of proficiency on syntactic second-language processing of German and Italian: Evidence from event-related potentials. *Journal of Cognitive Neuroscience*, *18*(12), 2030-2048. https://doi.org/10.1162/jocn.2006.18.12.2030

Ruzich, E., Allison, C., Smith, P., Watson, P., Auyeung, B., Ring, H., & Baron-Cohen, S. (2015). Measuring autistic traits in the general population: A systematic review of the Autism-Spectrum Quotient (AQ) in a nonclinical population sample of 6,900 typical adult males and females. *Molecular Autism*, *6*(1), 2. https://doi.org/10.1186/2040-2392-6-2

Sassenhagen, J., & Fiebach, C. J. (2019). Finding the P3 in the P600: Decoding shared neural mechanisms of responses to syntactic violations and oddball targets. *NeuroImage*, *200*, 425-436. https://doi.org/10.1016/j.neuroimage.2019. 06.048

Sassenhagen, J., Schlesewsky, M., & Bornkessel-Schlesewsky, I. (2014). The P600-as-P3 hypothesis revisited: Single-trial analyses reveal that the late EEG positivity following linguistically deviant material is reaction time aligned. *Brain and Language*, *137*, 29-39. https://doi.org/10.1016/j.bandl.2014.07.010

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., ... & Brechmann, A. (2014). Understanding source code with functional magnetic resonance imaging. *Proceedings of the 36th International Conference on Software Engineering*, 378-389. ACM. 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

Stevenson, J. L., & Hart, K. R. (2017). Psychometric properties of the Autism-Spectrum Quotient for assessing low and high levels of autistic traits in college students. *Journal of Autism and Developmental Disorders*, *47*(6), 1838–1853. https://doi.org/10.1007/s10803-017-3109-1

Suárez-Pellicioni, M., Núñez-Peña, M. I., & Colomé, A. (2013). Mathematical anxiety effects on simple arithmetic processing efficiency: An event-related potential study. *Biological Psychology*, *94*(3), 517-526. https://doi.org/10.1016/j.biopsycho. 2013.09.012

Szucs, D., & Csépe, V. (2004). Access to numerical information is dependent on the modality of stimulus presentation in mental addition: A combined ERP and behavioral study. *Cognitive Brain Research*, *19,* 10–27. https://doi.org/10.1016/j.cogbrainres .2003.11.002

Tew, A.E. 2010. *Assessing fundamental introductory computing concept knowledge in a language independent manner.* [Doctoral dissertation, Georgia Institute of Technology]. Georgia Tech Library. https://smartech.gatech.edu/handle/1853/37090

Tew, A.E. & Guzdial, M. (2011). The FCS1: A language independent assessment of CS1 knowledge. *Proceedings of the 42nd ACM technical symposium on computer science education* (2011), 111–116. https://doi.org/10.1145/1953163.1953200

Tirre, W. C., & Pena, C. M. (1993). Components of quantitative reasoning: General and group ability factors. *Intelligence*, *17*(4), 501–521. https://doi.org/10.1016/0160-2896(93)90015-W

Tokowicz, N., & MacWhinney, B. (2005). Implicit and explicit measures of sensitivity to violations in second language grammar: An event-related potential investigation. *Studies in Second Language Acquisition*, *27*(2), 173-204. https://doi.org/10.1017/S0272263105050102

Van Hell, J. G., & Tokowicz, N. (2010). Event-related brain potentials and second language learning: Syntactic processing in late L2 learners at different L2 proficiency levels. *Second Language Research*, *26*(1), 43-74. https://doi.org/10.1177/ 0267658309337637

Vissers, C. T. W., Chwilla, D. J., & Kolk, H. H. (2006). Monitoring in language perception: The effect of misspellings of words in highly constrained sentences. *Brain Research*, *1106*(1), 150-163. https://doi.org/10.1016/j.brainres.2006.05.012

Wang, Y., Kong, J., Tang, D., Zhuang, D., & Li, S. (2000). Event-related potential N270 is elicited by mental conflict processing in human brain. *Neuroscience Letters*, *293,* 17–20. https://doi.org/10.1016/S0304-3940(00)01480-4

Weber-Fox, C. M., & Neville, H. J. (1996). Maturational constraints on functional specializations for language processing: ERP and behavioral evidence in bilingual speakers. *Journal of Cognitive Neuroscience*, *8*(3), 231-256. https://doi.org/10.1162/jocn.1996.8.3.231

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

Wheelwright, S., Baron-Cohen, S., Goldenfeld, N., Delaney, J., Fine, D., Smith, R., Weil, L., & Wakabayashi, A. (2006). Predicting Autism Spectrum Quotient (AQ) from the Systemizing Quotient-Revised (SQ-R) and Empathy Quotient (EQ). *Brain Research*, *1079*(1), 47–56. https://doi.org/10.1016/j.brainres.2006.01.012

Wray, S. (2007). SQ Minus EQ can predict programming aptitude. *Proceedings of the PPIG 19th Annual Workshop,* 243-254. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.222.8951&rep=rep1&type=pdf

Xie, B., Davidson, M. J., Li, M., & Ko, A. J. (2019). An item response theory evaluation of a language-independent CS1 knowledge assessment. *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 699-705. https://doi.org/10.1145/3287324.3287370

References

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., ... & Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, *29*(2-3), 205-253. https://doi.org/10.1080/08993408.2019.1565235

Xie, B., Nelson, G. L., Akkaraju, H., Kwok, W., & Ko, A. J. (2020). The effect of informing agency in self-directed online learning environments. *Proceedings of the Seventh ACM Conference on Learning@ Scale*, 77-89. https://doi.org/10.1145/3386527.3405928

Summary

This thesis examines computer programming from a cognitive perspective. Programming is a relatively new skill that has become increasingly important throughout society in recent years. In order to be able to teach this skill optimally, it is essential to understand the cognitive nature of programming. Specifically, it is important to investigate whether this skill relates to other cognitive skills, personality traits and brain processing, and if so how. This thesis covers all three of these areas.

Chapter 1 provides an overview of the history of programming research. It covers the main findings in the three areas of interest and discusses methodologies that are commonly used in these fields. It then states the main research questions of the thesis and introduces the topics covered in of each of the following chapters.

Chapter 2 of the thesis lays the foundations for Chapters 3 and 4 by validating two short versions of the SCS1, a programming test for beginning undergraduate students in computer science. We found that the two short versions could not be considered fully parallel, and that one version was of questionable quality. However, the other version was of comparable reliability and validity to the original, full-length, test, that had been validated in previous studies. Hence, this study results in a validated short version, allowing this test to be used in other research that is only able to allocate short amounts of time to testing.

Chapter 3 explores which of the cognitive skills tested at the start of a semester-long undergraduate programming course, predicted programming performance at the end of that course. We used two measures of programming skill: course-related programming skill, which was measured using the students' course grades, and generalised programming skill, which was measured using the short versions of the programming test described in Chapter 2. We found that logical reasoning skill was the most reliable predictor of programming skill, as it predicted both course-related and generalised programming performance. Algebra and vocabulary learning skills only predicted generalised programming performance. Grammar learning and pattern recognition skills did not predict programming performance on either of the measures.

Chapter 4 investigates whether autistic traits in undergraduate students predict their programming skill at the end of the course. We used the same participants and the

same programming measures as in Chapter 3. We found that the students in our course had higher autistic traits than the general population. However, autistic traits did not predict performance on either of the programming measures. They also did not correlate with any of the cognitive skills described in Chapter 3. This led to the conclusion that autistic traits do not predict programming aptitude. However, it is possible that autistic traits may relate to an interest in programming. This would explain the higher autistic traits in our student population. Future studies will need to investigate this hypothesis further.

In Chapter 5 investigates whether the brain processes syntax violations in a programming language (Java) in a similar way to grammar violations in a native (Dutch) and a foreign (English) natural language. We used Event-Related Potentials (ERPs) as a method to measure electrical brain responses to stimuli. In all three languages, sentences with violations elicited more positive brain responses than sentences without violations. However, there were differences between the languages with regard to the onset, offset and scalp distribution of the effects. Specifically, there was an early onset and offset of the effect in Java, as well a frontal and bilateral scalp distribution. This suggests that this type of bracket violation is processed differently to the subject-verb violations in the two natural languages. Based on both the timing and the scalp distribution, it is inconclusive whether the effect for Java reflects a processing mechanism such as that used for natural languages. However, the effect in response to the bracket violations is similar to that observed in the past in response to orthographic violations in natural language, suggesting that programmers may perceive these violations as incorrect spelling rather than incorrect syntax.

Chapter 6 discusses and integrates the findings of the separate studies and gives suggestions for future studies and for education. Based on the results of the separate studies, logical reasoning seems to be the most reliable predictor of programming performance. Future studies will have to further disentangle the relationships between cognitive skills, teaching and testing methods and programming skills. In particular, they should study whether language skills play a larger role in programming courses where syntax is taught and assessed more explicitly. For education, it may be beneficial to teach logical reasoning explicitly. Additionally, both educators and researchers should be aware

that teaching and testing methods may favour students with certain cognitive strengths over others.

The results of this thesis provide essential new knowledge on the cognitive nature of programming and facilitate future research into this area. Ultimately, the findings in this field will shape our understanding of programming as a skill.

Samenvatting

Dit proefschrift onderzoekt het computerprogrammeren vanuit een cognitieve invalshoek. Programmeren is een relatief nieuwe vaardigheid die de afgelopen decennia steeds belangrijker is geworden in onze samenleving. Om deze vaardigheid optimaal te kunnen doceren is het belangrijk om de cognitieve aard van het programmeren te begrijpen. Het is dan ook belangrijk om te onderzoeken of en hoe deze vaardigheid verband houdt met andere cognitieve vaardigheden, persoonlijkheidskenmerken en verwerking in de hersenen. Dit proefschrift behandelt alle drie deze gebieden.

Hoofdstuk 1 geeft een overzicht van de geschiedenis van programmeeronderzoek. Het behandelt de belangrijkste bevindingen in de drie aandachtsgebieden en bespreekt methodologieën die vaak worden gebruikt om programmeervaardigheden te onderzoeken. Vervolgens worden de belangrijkste onderzoeksvragen van het proefschrift geformuleerd en worden de onderwerpen van de volgende hoofdstukken geïntroduceerd.

Hoofdstuk 2 van het proefschrift legt de basis voor de Hoofdstukken 3 en 4 door twee verkorte versies van de SCS1 (een programmeertest voor beginnende informaticastudenten) te valideren. De twee verkorte versies bleken niet als volledig gelijk te kunnen worden beschouwd. Daarnaast was één versie van twijfelachtige kwaliteit. De andere versie had echter een vergelijkbare betrouwbaarheid en validiteit als de originele, volledige test, die in eerdere onderzoeken was gevalideerd. Deze studie resulteert dan ook in een gevalideerde korte versie die kan worden gebruikt in ander onderzoek waar kortere testtijden noodzakelijk zijn.

In Hoofdstuk 3 wordt bestudeerd welke van de cognitieve vaardigheden, getest aan het begin van een semesterlang programmeervak, de programmeerprestaties aan het einde van dit semester voorspellen. Hierbij werden programmeervaardigheden op twee manier gemeten: vakgerelateerde programmeervaardigheden, die werden gemeten met de cijfers van de studenten voor het vak, en algemene programmeervaardigheden, die werden gemeten met behulp van de korte versies van de programmeertest zoals beschreven in Hoofdstuk 2. We ontdekten dat logisch redeneren de meest betrouwbare voorspeller was van programmeervaardigheden, aangezien het zowel vakgerelateerde vaardigheden als algemene programmeervaardigheden voorspelde. Algebraïsche vaardigheden en het vermogen om nieuwe woorden te leren voorspelden alleen algemene

programmeervaardigheden. Het leren van grammaticale vaardigheden en patroonherkenning voorspelden geen van beide soorten programmeervaardigheden.

In Hoofdstuk 4 wordt onderzocht of autistische eigenschappen bij bachelorstudenten hun programmeervaardigheden aan het einde van een programmeervak voorspellen. We gebruikten dezelfde deelnemers en dezelfde programmeertests als in Hoofdstuk 3. We ontdekten dat de studenten van dit vak meer autistische eigenschappen hadden dan de gemiddelde bevolking. Autistische eigenschappen hadden echter geen voorspellende waarde voor vakgerelateerde vaardigheden of algemene programmeervaardigheden aan het einde van het vak. De eigenschappen correleerden ook niet met de cognitieve vaardigheden zoals beschreven in Hoofdstuk 3. Het is echter mogelijk dat autistische kenmerken wel verband houden met interesse in programmeren. Dit zou de bovengemiddelde autistische eigenschappen in onze studentenpopulatie verklaren. Toekomstig onderzoek zal deze hypothese verder moeten onderzoeken.

In Hoofdstuk 5 wordt onderzocht of de hersenen fouten in de syntaxis van een programmeertaal (in Java) op een vergelijkbare manier verwerken als grammaticale fouten in hun moedertaal (Nederlands) en een vreemde, natuurlijke taal (Engels). We gebruikten *Event-Related Potentials (ERP's)* als methode om elektrische hersenactiviteit in reactie op stimuli te meten. In alle drie de talen veroorzaakten zinnen met fouten meer positieve hersenactiviteit dan zinnen zonder fouten. Er waren echter wel verschillen tussen de talen met betrekking tot het begin, de duur en de locatie in de hersenen van de effecten. Het effect voor Java begon vroeg, duurde kort en had een frontale en bilaterale verdeling over de schedel. Dit suggereert dat dit soort fouten in de code anders worden verwerkt dan fouten met persoon en getal van werkwoordsvervoeging in de twee natuurlijke talen. Op basis van zowel het tijdsverloop als de topografie van het effect voor Java blijft onduidelijk of dit soort fouten op dezelfde manier verwerkt worden als fouten in natuurlijke talen. Het effect in reactie op de fouten in de programmeersyntaxis lijkt wel op het effect dat in het verleden is waargenomen als reactie op orthografische fouten in een natuurlijke taal. Dit suggereert dat programmeurs de haakjesfouten wellicht als onjuiste spelling beschouwen, in plaats van als onjuiste syntaxis.

In Hoofdstuk 6 worden de bevindingen van de afzonderlijke studies besproken en worden suggesties gegeven voor toekomstig onderzoek en voor het onderwijs. Op basis van de resultaten van de afzonderlijke onderzoeken lijkt logisch redeneren de meest betrouwbare voorspeller van programmeervaardigheden. Toekomstig onderzoek zal de verbanden tussen cognitieve vaardigheden, onderwijs- en testmethoden, en programmeervaardigheden verder moeten verduidelijken. Daarbij is het vooral van belang om te onderzoeken of taalvaardigheid een grotere rol speelt in programmeercursussen waar syntaxis explicieter wordt onderwezen en beoordeeld. Voor het onderwijs kan het nuttig zijn om logisch redeneren expliciet te onderwijzen. Ten slotte moeten zowel docenten als onderzoekers zich ervan bewust zijn dat, afhankelijk van de gekozen onderwijs- en testmethoden, studenten met bepaalde cognitieve talenten in het voordeel kunnen zijn ten opzichte van anderen.

De bevindingen van dit proefschrift leveren belangrijke nieuwe kennis op over de cognitieve aard van programmeren en bevorderen toekomstig onderzoek op dit gebied. Uiteindelijk zullen de bevindingen in dit onderzoeksveld onze kennis van programmeren als vaardigheid verder vormgeven.

GRODIL

Groningen Dissertations in Linguistics

1. Henriëtte de Swart (1991). Adverbs of Quantification: A Generalized Quantifier Approach.
2. Eric Hoekstra (1991). Licensing Conditions on Phrase Struc¬ture.
3. Dicky Gilbers (1992). Phonological Networks. A Theory of Segment Represen¬tation.
4. Helen de Hoop (1992). Case Configuration and Noun Phrase Interpretation.
5. Gosse Bouma (1993). Nonmonotonicity and Categorial Unifi¬ca¬tion Grammar.
6. Peter I. Blok (1993). The Interpretation of Focus.
7. Roelien Bastiaanse (1993). Studies in Aphasia.
8. Bert Bos (1993). Rapid User Interface Development with the Script Language Gist.
9. Wim Kosmeijer (1993). Barriers and Licensing.
10. Jan-Wouter Zwart (1993). Dutch Syntax: A Minimalist Appro¬ach.
11. Mark Kas (1993). Essays on Boolean Functions and Negative Polarity.
12. Ton van der Wouden (1994). Negative Contexts.
13. Joop Houtman (1994). Coordination and Constituency: A Study in Categorial Grammar.
14. Petra Hendriks (1995). Comparatives and Categorial Grammar.
15. Maarten de Wind (1995). Inversion in French.
16. Jelly Julia de Jong (1996). The Case of Bound Pronouns in Peripheral Romance.
17. Sjoukje van der Wal (1996). Negative Polarity Items and Negation: Tandem Acquisition.
18. Anastasia Giannakidou (1997). The Landscape of Polarity Items.
19. Karen Lattewitz (1997). Adjacency in Dutch and German.
20. Edith Kaan (1997). Processing Subject-Object Ambiguities in Dutch.
21. Henny Klein (1997). Adverbs of Degree in Dutch.
22. Leonie Bosveld-de Smet (1998). On Mass and Plural Quantification: The case of French ‘des’/‘du’-NPs.
23. Rita Landeweerd (1998). Discourse semantics of perspective and temporal structure.
24. Mettina Veenstra (1998). Formalizing the Minimalist Program.
25. Roel Jonkers (1998). Comprehension and Production of Verbs in aphasic Speakers.
26. Erik F. Tjong Kim Sang (1998). Machine Learning of Phonotactics.
27. Paulien Rijkhoek (1998). On Degree Phrases and Result Clauses.
28. Jan de Jong (1999). Specific Language Impairment in Dutch: Inflectional Morphology and Argument Structure.
29. H. Wee (1999). Definite Focus.
30. Eun-Hee Lee (2000). Dynamic and Stative Information in Temporal Reasoning: Korean tense and aspect in discourse.
31. Ivilin P. Stoianov (2001). Connectionist Lexical Processing.

32. Klarien van der Linde (2001). Sonority substitutions.
33. Monique Lamers (2001). Sentence processing: using syntactic, semantic, and thematic information.
34. Shalom Zuckerman (2001). The Acquisition of "Optional" Movement.
35. Rob Koeling (2001). Dialogue-Based Disambiguation: Using Dialogue Status to Improve Speech Understanding.
36. Esther Ruigendijk (2002). Case assignment in Agrammatism: a cross-linguistic study.
37. Tony Mullen (2002). An Investigation into Compositional Features and Feature Merging for Maximum Entropy-Based Parse Selection.
38. Nanette Bienfait (2002). Grammatica-onderwijs aan allochtone jongeren.
39. Dirk-Bart den Ouden (2002). Phonology in Aphasia: Syllables and segments in level-specific deficits.
40. Rienk Withaar (2002). The Role of the Phonological Loop in Sentence Comprehension.
41. Kim Sauter (2002). Transfer and Access to Universal Grammar in Adult Second Language Acquisition.
42. Laura Sabourin (2003). Grammatical Gender and Second Language Processing: An ERP Study.
43. Hein van Schie (2003). Visual Semantics.
44. Lilia Schürcks-Grozeva (2003). Binding and Bulgarian.
45. Stasinos Konstantopoulos (2003). Using ILP to Learn Local Linguistic Structures.
46. Wilbert Heeringa (2004). Measuring Dialect Pronunciation Differences using Levenshtein Distance.
47. Wouter Jansen (2004). Laryngeal Contrast and Phonetic Voicing: A Laboratory Phonology.
48. Judith Rispens (2004). Syntactic and phonological processing in developmental dyslexia.
49. Danielle Bougaïré (2004). L'approche communicative des campagnes de sensibilisation en santé publique au Burkina Faso: Les cas de la planification familiale, du sida et de l'excision.
50. Tanja Gaustad (2004). Linguistic Knowledge and Word Sense Disambiguation.
51. Susanne Schoof (2004). An HPSG Account of Nonfinite Verbal Complements in Latin.
52. M. Begoña Villada Moirón (2005). Data-driven identification of fixed expressions and their modifiability.
53. Robbert Prins (2005). Finite-State Pre-Processing for Natural Language Analysis.
54. Leonoor van der Beek (2005) Topics in Corpus-Based Dutch Syntax.
55. Keiko Yoshioka (2005). Linguistic and gestural introduction and tracking of referents in L1 and L2 discourse.

56. Sible Andringa (2005). Form-focused instruction and the development of second language proficiency.
57. Joanneke Prenger (2005). Taal telt! Een onderzoek naar de rol van taalvaardigheid en tekstbegrip in het realistisch wiskundeonderwijs.
58. Neslihan Kansu-Yetkiner (2006). Blood, Shame and Fear: Self-Presentation Strategies of Turkish Women's Talk about their Health and Sexuality.
59. Mónika Z. Zempléni (2006). Functional imaging of the hemispheric contribution to language processing.
60. Maartje Schreuder (2006). Prosodic Processes in Language and Music.
61. Hidetoshi Shiraishi (2006). Topics in Nivkh Phonology.
62. Tamás Biró (2006). Finding the Right Words: Implementing Optimality Theory with Simulated Annealing.
63. Dieuwke de Goede (2006). Verbs in Spoken Sentence Processing: Unraveling the Activation Pattern of the Matrix Verb.
64. Eleonora Rossi (2007). Clitic production in Italian agrammatism.
65. Holger Hopp (2007). Ultimate Attainment at the Interfaces in Second Language Acquisition: Grammar and Processing.
66. Gerlof Bouma (2008). Starting a Sentence in Dutch: A corpus study of subject- and object-fronting.
67. Julia Klitsch (2008). Open your eyes and listen carefully. Auditory and audiovisual speech perception and the McGurk effect in Dutch speakers with and without aphasia.
68. Janneke ter Beek (2008). Restructuring and Infinitival Complements in Dutch.
69. Jori Mur (2008). Off-line Answer Extraction for Question Answering.
70. Lonneke van der Plas (2008). Automatic Lexico-Semantic Acquisition for Question Answering.
71. Arjen Versloot (2008). Mechanisms of Language Change: Vowel reduction in 15th century West Frisian.
72. Ismail Fahmi (2009). Automatic term and Relation Extraction for Medical Question Answering System.
73. Tuba Yarbay Duman (2009). Turkish Agrammatic Aphasia: Word Order, Time Reference and Case.
74. Maria Trofimova (2009). Case Assignment by Prepositions in Russian Aphasia.
75. Rasmus Steinkrauss (2009). Frequency and Function in WH Question Acquisition. A Usage-Based Case Study of German L1 Acquisition.

76. Marjolein Deunk (2009). Discourse Practices in Preschool. Young Children's Participation in Everyday Classroom Activities.
77. Sake Jager (2009). Towards ICT-Integrated Language Learning: Developing an Implementation Framework in terms of Pedagogy, Technology and Environment.
78. Francisco Dellatorre Borges (2010). Parse Selection with Support Vector Machines.
79. Geoffrey Andogah (2010). Geographically Constrained Information Retrieval.
80. Jacqueline van Kruiningen (2010). Onderwijsontwerp als conversatie. Probleemoplossing in interprofessioneel overleg.
81. Robert G. Shackleton (2010). Quantitative Assessment of English-American Speech Relationships.
82. Tim Van de Cruys (2010). Mining for Meaning: The Extraction of Lexico-semantic Knowledge from Text.
83. Therese Leinonen (2010). An Acoustic Analysis of Vowel Pronunciation in Swedish Dialects.
84. Erik-Jan Smits (2010). Acquiring Quantification. How Children Use Semantics and Pragmatics to Constrain Meaning.
85. Tal Caspi (2010). A Dynamic Perspective on Second Language Development.
86. Teodora Mehotcheva (2010). After the fiesta is over. Foreign language attrition of Spanish in Dutch and German Erasmus Student.
87. Xiaoyan Xu (2010). English language attrition and retention in Chinese and Dutch university students.
88. Jelena Prokić (2010). Families and Resemblances.
89. Radek Šimík (2011). Modal existential wh-constructions.
90. Katrien Colman (2011). Behavioral and neuroimaging studies on language processing in Dutch speakers with Parkinson's disease.
91. Siti Mina Tamah (2011). A Study on Student Interaction in the Implementation of the Jigsaw Technique in Language Teaching.
92. Aletta Kwant (2011).Geraakt door prentenboeken. Effecten van het gebruik van prentenboeken op de sociaal-emotionele ontwikkeling van kleuters.
93. Marlies Kluck (2011). Sentence amalgamation.
94. Anja Schüppert (2011). Origin of asymmetry: Mutual intelligibility of spoken Danish and Swedish.
95. Peter Nabende (2011). Applying Dynamic Bayesian Networks in Transliteration Detection and Generation.

96. Barbara Plank (2011). Domain Adaptation for Parsing.
97. Cagri Coltekin (2011).Catching Words in a Stream of Speech: Computational simulations of segmenting transcribed child-directed speech.
98. Dörte Hessler (2011). Audiovisual Processing in Aphasic and Non-Brain-Damaged Listeners: The Whole is More than the Sum of its Parts.
99. Herman Heringa (2012). Appositional constructions.
100. Diana Dimitrova (2012). Neural Correlates of Prosody and Information Structure.
101. Harwintha Anjarningsih (2012). Time Reference in Standard Indonesian Agrammatic Aphasia.
102. Myrte Gosen (2012). Tracing learning in interaction. An analysis of shared reading of picture books at kindergarten.
103. Martijn Wieling (2012). A Quantitative Approach to Social and Geographical Dialect Variation.
104. Gisi Cannizzaro (2012). Early word order and animacy.
105. Kostadin Cholakov (2012). Lexical Acquisition for Computational Grammars. A Unified Model.
106. Karin Beijering (2012). Expressions of epistemic modality in Mainland Scandinavian. A study into the lexicalization-grammaticalization-pragmaticalization interface.
107. Veerle Baaijen (2012). The development of understanding through writing.
108. Jacolien van Rij (2012). Pronoun processing: Computational, behavioral, and psychophysiological studies in children and adults.
109. Ankelien Schippers (2012). Variation and change in Germanic long-distance dependencies.
110. Hanneke Loerts (2012).Uncommon gender: Eyes and brains, native and second language learners, & grammatical gender.
111. Marjoleine Sloos (2013). Frequency and phonological grammar: An integrated approach. Evidence from German, Indonesian, and Japanese.
112. Aysa Arylova. (2013) Possession in the Russian clause. Towards dynamicity in syntax.
113. Daniël de Kok (2013). Reversible Stochastic Attribute-Value Grammars.
114. Gideon Kotzé (2013). Complementary approaches to tree alignment: Combining statistical and rule-based methods.
115. Fridah Katushemererwe (2013). Computational Morphology and Bantu Language Learning: an Implementation for Runyakitara.
116. Ryan C. Taylor (2013). Tracking Referents: Markedness, World Knowledge and Pronoun Resolution.

117. Hana Smiskova-Gustafsson (2013). Chunks in L2 Development: A Usage-based Perspective.
118. Milada Walková (2013). The aspectual function of particles in phrasal verbs.
119. Tom O. Abuom (2013). Verb and Word Order Deficits in Swahili-English bilingual agrammatic speakers.
120. Gülsen Yılmaz (2013). Bilingual Language Development among the First Generation Turkish Immigrants in the Netherlands.
121. Trevor Benjamin (2013). Signaling Trouble: On the linguistic design of other-initiation of repair in English conversation.
122. Nguyen Hong Thi Phuong (2013). A Dynamic Usage-based Approach to Second Language Teaching.
123. Harm Brouwer (2014). The Electrophysiology of Language Comprehension: A Neurocomputational Model.
124. Kendall Decker (2014). Orthography Development for Creole Languages.
125. Laura S. Bos (2015). The Brain, Verbs, and the Past: Neurolinguistic Studies on Time Reference.
126. Rimke Groenewold (2015). Direct and indirect speech in aphasia: Studies of spoken discourse production and comprehension.
127. Huiping Chan (2015). A Dynamic Approach to the Development of Lexicon and Syntax in a Second Language.
128. James Griffiths (2015). On appositives.
129. Pavel Rudnev (2015). Dependency and discourse-configurationality: A study of Avar.
130. Kirsten Kolstrup (2015). Opportunities to speak. A qualitative study of a second language in use.
131. Güliz Güneş (2015). Deriving Prosodic structures.
132. Cornelia Lahmann (2015). Beyond barriers. Complexity, accuracy, and fluency in long-term L2 speakers' speech.
133. Sri Wachyunni (2015). Scaffolding and Cooperative Learning: Effects on Reading Comprehension and Vocabulary Knowledge in English as a Foreign Language.
134. Albert Walsweer (2015). Ruimte voor leren. Een etnogafisch onderzoek naar het verloop van een interventie gericht op versterking van het taalgebruik in een knowledge building environment op kleine Friese basisscholen.
135. Aleyda Lizeth Linares Calix (2015). Raising Metacognitive Genre Awareness in L2 Academic Readers and Writers.

136. Fathima Mufeeda Irshad (2015). Second Language Development through the Lens of a Dynamic Usage-Based Approach.
137. Oscar Strik (2015). Modelling analogical change. A history of Swedish and Frisian verb inflection.
138. He Sun (2015). Predictors and stages of very young child EFL learners' English development in China.
139 Marieke Haan (2015). Mode Matters. Effects of survey modes on participation and answering behavior.
140. Nienke Houtzager (2015). Bilingual advantages in middle-aged and elderly populations.
141. Noortje Joost Venhuizen (2015). Projection in Discourse: A data-driven formal semantic analysis.
142. Valerio Basile (2015). From Logic to Language: Natural Language Generation from Logical Forms.
143. Jinxing Yue (2016). Tone-word Recognition in Mandarin Chinese: Influences of lexical-level representations.
144. Seçkin Arslan (2016). Neurolinguistic and Psycholinguistic Investigations on Evidentiality in Turkish.
145. Rui Qin (2016). Neurophysiological Studies of Reading Fluency. Towards Visual and Auditory Markers of Developmental Dyslexia.
146. Kashmiri Stec (2016). Visible Quotation: The Multimodal Expression of Viewpoint.
147. Yinxing Jin (2016). Foreign language classroom anxiety: A study of Chinese university students of Japanese and English over time.
148. Joost Hurkmans (2016). The Treatment of Apraxia of Speech. Speech and Music Therapy, an Innovative Joint Effort.
149. Franziska Köder (2016). Between direct and indirect speech: The acquisition of pronouns in reported speech.
150. Femke Swarte (2016). Predicting the mutual intelligibility of Germanic languages from linguistic and extra-linguistic factors.
151. Sanne Kuijper (2016). Communication abilities of children with ASD and ADHD. Production, comprehension, and cognitive mechanisms.
152. Jelena Golubović (2016). Mutual intelligibility in the Slavic language area.
153. Nynke van der Schaaf (2016). "Kijk eens wat ik kan!" Sociale praktijken in de interactie tussen kinderen van 4 tot 8 jaar in de buitenschoolse opvang.
154. Simon Šuster (2016). Empirical studies on word representations.

155. Kilian Evang (2016). Cross-lingual Semantic Parsing with Categorial Grammars.
156. Miren Arantzeta Pérez (2017). Sentence comprehension in monolingual and bilingual aphasia: Evidence from behavioral and eye-tracking methods.
157. Sana-e-Zehra Haidry (2017). Assessment of Dyslexia in the Urdu Language.
158. Srđan Popov (2017). Auditory and Visual ERP Correlates of Gender Agreement Processing in Dutch and Italian.
159. Molood Sadat Safavi (2017). The Competition of Memory and Expectation in Resolving Long-Distance Dependencies: Psycholinguistic Evidence from Persian Complex Predicates.
160. Christopher Bergmann (2017). Facets of native-likeness: First-language attrition among German emigrants to Anglophone North America.
161. Stefanie Keulen (2017). Foreign Accent Syndrome: A Neurolinguistic Analysis.
162. Franz Manni (2017). Linguistic Probes into Human History.
163. Margreet Vogelzang (2017). Reference and cognition: Experimental and computational cognitive modeling studies on reference processing in Dutch and Italian.
164. Johannes Bjerva (2017). One Model to Rule them all. Multitask and Multilingual Modelling for Lexical Analysis: Multitask and Multilingual Modelling for Lexical Analysis.
165. Dieke Oele (2018). Automated translation with interlingual word representations.
166. Lucas Seuren (2018). The interactional accomplishment of action.
167. Elisabeth Borleffs (2018). Cracking the code - Towards understanding, diagnosing and remediating dyslexia in Standard Indonesian.
168. Mirjam Günther-van der Meij (2018). The impact of degree of bilingualism on L3 development English language development in early and later bilinguals in the Frisian context.
169. Ruth Koops van 't Jagt (2018). Show, don’t just tell: Photo stories to support people with limited health literacy.
170. Bernat Bardagil-Mas (2018).Case and agreement in Panará.
171. Jessica Overweg (2018). Taking an alternative perspective on language in autism.
172. Lennie Donné (2018). Convincing through conversation: Unraveling the role of interpersonal health communication in health campaign effectiveness.
173. Toivo Glatz (2018). Serious games as a level playing field for early literacy: A behavioural and neurophysiological evaluation.
174. Ellie van Setten (2019). Neurolinguistic Profiles of Advanced Readers with Developmental Dyslexia.

175. Anna Pot (2019). Aging in multilingual Netherlands: Effects on cognition, wellbeing and health.
176. Audrey Rousse-Malpat (2019). Effectiveness of explicit vs. implicit L2 instruction: a longitudinal classroom study on oral and written skills.
177. Rob van der Goot (2019). Normalization and Parsing Algorithms for Uncertain Input.
178. Azadeh Elmianvari (2019). Multilingualism, Facebook and the Iranian diaspora.
179. Joëlle Ooms (2019). "Don't make my mistake": Narrative fear appeals in health communication.
180. Annerose Willemsen (2019). The floor is yours: A conversation analytic study of teachers' conduct facilitating whole-class discussions around texts.
181. Frans Hiddink (2019). Early childhood problem-solving interaction: Young children's discourse during small-group work in primary school.
182. Hessel Haagsma (2020). A Bigger Fish to Fry: Scaling up the Automatic Understanding of Idiomatic Expressions.
183. Juliana Andrade Feiden (2020). The Influence of Conceptual Number in Coreference Establishing: An ERP Study on Brazilian and European Portuguese.
184. Sirkku Lesonen (2020). Valuing variability: Dynamic usage-based principles in the L2 development of four Finnish language learners.
185. Nathaniel Lartey (2020). A neurolinguistic approach to the processing of resumption in Akan focus constructions.
186. Bernard Amadeus Jaya Jap (2020). Syntactic Frequency and Sentence Processing in Standard Indonesian.
187. Ting Huang (2020). Learning an L2 and L3 at the same time: help or hinder?.
188. Anke Herder (2020). Peer talk in collaborative writing of primary school students: A conversation analytic study of student interaction in the context of inquiry learning.
189. Ellen Schep (2020). Attachment in interaction: A conversation analytic study on dinner conversations with adolescents in family-style group care.
190. Yulia Akinina (2020). Individual behavioural patterns and neural underpinnings of verb processing in aphasia.
191. Camila Martinez Rebolledo (2020). Comprehending the development of reading difficulties in children with SLI.
192. Jakolien den Hollander (2021). Distinguishing a phonological encoding disorder from Apraxia of Speech in individuals with aphasia by using EEG.
193. Rik van Noord (2021). Character-based Neural Semantic Parsing.

194. Anna de Koster (2021). Acting Individually or Together? An Investigation of Children's Development of Distributivity.
195. Frank Tsiwah (2021). Time, tone and the brain: Behavioral and neurophysiological studies on time reference and grammatical tone in Akan.
196. Amélie la Roi (2021). Idioms in the Aging Brain.
197. Nienke Wolthuis (2021). Language impairments and resting-state EEG in brain tumour patients: Revealing connections.
198. Nienke Smit (2021). Get it together: Exploring the dynamics of teacher-student interaction in English as a foreign language lessons.
199. Svetlana Averina (2021). Bilateral neural correlates of treatment-induced changes in chronic aphasia.
200. Wilasinee Siriboonpipattana (2021). Neurolinguistic studies on the linguistic expression of time reference in Thai.
201. Irene Graafsma (2021). Computer programming skills: A cognitive perspective.

GRODIL
Center for Language and Cognition Groningen (CLCG)
P.O. Box 716
9700 AS Groningen
The Netherlands