Focus on Language in Introductory Programming

Zur Erlangung des Grades eines Doktors der Naturwissenschaften (Dr. rer. nat.)
Genehmigte Dissertation von Svana Esche aus Usingen
Tag der Einreichung: 15.01.2024, Tag der Prüfung: 17.04.2024

1. Gutachten: Prof. Dr. Karsten Weihe, Technische Universität Darmstadt
2. Gutachten: Prof. Dr. Ulrik Schroeder, RWTH Aachen
Darmstadt, Technische Universität Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Computer Science
Department

Fachgruppe Algorithmik

Erklärungen laut Promotionsordnung

§ 8 Abs. 1 lit. c PromO

Ich versichere hiermit, dass die elektronische Version meiner Dissertation mit der schriftlichen Version übereinstimmt.

§ 8 Abs. 1 lit. d PromO

Ich versichere hiermit, dass zu einem vorherigen Zeitpunkt noch keine Promotion versucht wurde. In diesem Fall sind nähere Angaben über Zeitpunkt, Hochschule, Dissertationsthema und Ergebnis dieses Versuchs mitzuteilen.

§ 9 Abs. 1 PromO

Ich versichere hiermit, dass die vorliegende Dissertation selbstständig und nur unter Verwendung der angegebenen Quellen verfasst wurde.

§ 9 Abs. 2 PromO

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 15.01.2024

S. Esche

Abstract

This thesis analyzes the relationship between natural language and reasoning when introducing programming in postsecondary education. The particular focus is on the insights that emerge when this relationship is made explicit in teaching-learning situations. As a major result, natural language does indeed play an important role in these teaching-learning situations but deserves further research attention.

Making the described relationship explicit is itself a new meta-perspective for thinking about research and teaching in computing education. Developing this new meta-perspective is an important contribution of this work. As far as we know, this meta-perspective has not yet been synthesized as such and at the same time supported by empirical evidence in postsecondary education. Another contribution is the insight that diagnosing and using students' terms as part of language are particularly fruitful for research and teaching. Further research should pursue this new approach in order to shed light on well-known problems from a new perspective.

For the empirical evidence of these insights, this thesis used a mixed methods research design in which four studies were conducted. The mixed methods research design combined qualitative approaches such as qualitative content analysis with quantitative approaches such as experiments with students as participants. This approach enabled a comprehensive and multi-layered analysis of the results. We included all actors in the teaching-learning situations in order to obtain a more comprehensive analysis.

All four studies were conducted according to the respective gold standard as far as possible. This includes rigorous measurements and large sample sizes. The four studies were divided into an exploratory basis study and three subsequent application studies. The Basis Study focused on students, uncovered the terms they used and indicated a possible relationship between programming language and natural language. In addition, it led to new insights about correct conceptions and misconceptions of programming languages. Furthermore, the methodological approach of this study is

itself a contribution, as it shows how a qualitative analysis of terms can be carried out with large samples. Application Study I examined instructional videos as a teaching method used by instructors. As a basis for instructional videos to support students, the implementation of language focus across multiple representations turned out not to be convincing. Thus, the hypothesis about the positive impact of these language-based videos on students' code writing and self-efficacy could be rejected with reasonable certainty. In the Application Study II, we turn our attention to teaching staff in general and their competences in supporting students. We developed and evaluated a rubric, which is a guide that lists specific criteria for grading by category and level. The rubric serves as an assessment scheme for the quality in their responses, focusing on language. Finally, in the Application Study III, a test of testing programming aptitude was developed and validated based on converting programming thinking into natural language. The resulting Natural Language Computing Test, or NLCT, has proven to be highly accurate, valid, and reliable. As a minor weakness, the NLCT did not prove to be a predictor of success at the end of the course.

To summarize, the four studies conducted have shown that the potential of this new meta-perspective lies in the way in which the relationship described is carried out. The use of terms as a link between programming language and natural language seems to be particularly promising. In contrast, the use of multiple representations of programming language, natural language and visualization has shown to be less promising.

Zusammenfassung

In der vorliegenden Arbeit geht es um die Analyse der Beziehung zwischen natürlicher Sprache und Programmiersprachen in der Hochschullehre der Informatik. Schwerpunkt dieser Arbeit sind die Erkenntnisse, die sich ergeben, wenn diese Beziehung in Lehr-Lern-Situationen explizit gemacht wird. Die Auswertung der Ergebnisse belegt, dass natürliche Sprache in Lehr-Lern-Situationen tatsächlich eine wichtige Rolle spielt, die jedoch weitere Aufmerksamkeit in der Forschung verdient.

Die beschriebene Beziehung explizit zu machen ist selbst eine neue Meta-Perspektive für die Reflexion von Forschung und Lehre in der Informatikdidaktik. Die Entwicklung dieser neuen Meta-Perspektive ist ein wichtiger Beitrag dieser Arbeit. Soweit bekannt wurde diese Meta-Perspektive noch nicht als solche zusammengefasst und gleichzeitig durch empirische Belege in der Hochschuldidaktik der Informatik unterstützt. Ein weiterer Beitrag ist die Erkenntnis, dass die Diagnose und Verwendung von Begriffen, die Studierende verwenden, als Teil von Sprache für Forschung und Lehre besonders fruchtbar sind. Weitere Forschungen könnten diesen neuen Ansatz verfolgen, um bekannte Probleme aus einer neuen Perspektive zu beleuchten.

Für den empirischen Nachweis dieser Erkenntnisse in dieser Arbeit wurde ein methodischer Ansatz aus dem Bereich der Mixed Methods verwendet, welcher insgesamt die Durchführung von vier Studien umfasste. Dieser methodische Ansatz kombinierte qualitative Ansätze wie die qualitative Inhaltsanalyse mit quantitativen Ansätzen wie Experimenten mit Studierenden als Teilnehmende. Ein Vorteil dieses Ansatzes ist, dass eine umfassende und vielschichtige Analyse der Ergebnisse ermöglicht wird. Des Weiteren wurden alle Akteure, die in den üblichen Lehr-Lern-Situationen an Hochschulen involviert sind, einbezogen. Diese Integration zielte darauf ab, eine umfassendere Analyse zu erhalten.

Alle vier Studien wurden so weit wie möglich nach dem jeweiligen wissenschaftlichen Goldstandard durchgeführt. Dazu gehören beispielsweise rigorose Messungen und große

Stichprobengrößen. Die vier Studien bestehen aus einer explorativen Basisstudie und drei anschließenden Anwendungsstudien. Die Basisstudie nahm Studierende als Akteure in den Fokus. Hierbei deckte diese Studie die von den Studierenden verwendeten Begriffe auf und zeigte eine mögliche Beziehung zwischen Programmiersprache und natürlicher Sprache auf. Darüber hinaus führte die Basisstudie zu neuen Erkenntnissen über korrekte Vorstellungen und Fehlvorstellungen im Bereich der Programmiersprachen. Zudem ist der methodische Ansatz dieser Studie ein selbstständiger, wissenschaftlicher Beitrag, da er zeigt, wie eine qualitative Analyse von Begriffen mit großen Stichproben durchgeführt werden kann. Anwendungsstudie I untersuchte Lehrvideos, welche eine von vielen möglichen Lehrmethoden von Dozierenden darstellen. Als didaktische Grundlage für Lehrvideos, die auf die Unterstützung von Studierenden abzielen, erwies sich die Umsetzung des Sprachfokus über die Verwendung mehrerer Repräsentationsebenen als nicht überzeugend. Somit konnte die Hypothese über die positive Wirkung dieser sprachbasierten Videos auf das Schreiben von Programmcode und die Selbstwirksamkeit der Studierenden mit ziemlicher Sicherheit verworfen werden. Anwendungsstudie II fokussierte sich auf das Lehrpersonal im Allgemeinen und deren Kompetenzen bei der Unterstützung der Studierenden. Es wurde ein Rubric als tabellarisches Bewertungsschema entwickelt und evaluiert. Dieses stellt einen Leitfaden dar, der spezifische Kriterien für die Bewertung nach Kategorie und Niveau auflistet. Das Rubric dient als Bewertungsschema für die Qualität der Antworten, wobei gemäß Forschungsthema ein sprachlicher Schwerpunkt gesetzt wurde. Schließlich wurde in der Anwendungsstudie III ein Test zur Messung der Eignung und den Vorerfahrungen zum Programmierlernen entwickelt und validiert, der auf der Umwandlung von Programmieraufgaben in natürliche Sprache basiert. Der daraus resultierende Text namens Natural Language Computing Test (NLCT) hat sich als äußerst genau, valide und reliabel erwiesen. Dennoch erwies sich der NLCT nicht als guter Prädiktor für den Erfolg am Ende des Programmierkurses.

Zusammenfassend lässt sich anführen, dass die vier durchgeführten Studien gezeigt haben, dass das Potenzial dieser neuen Meta-Perspektive in der Art und Weise liegt, wie die beschriebene Beziehung durchgeführt wird. Die Verwendung von Begriffen als Bindeglied zwischen Programmiersprache und natürlicher Sprache scheint besonders vielversprechend zu sein. Im Gegensatz dazu hat sich die Verwendung mehrerer Repräsentationsebenen von Programmiersprache, natürlicher Sprache und Visualisierung als weniger vielversprechend erwiesen.

Publications

This thesis consists of the following publications, which appear in the order in which they were published.

Svana Esche and Karsten Weihe. 2023. Choosing a Didactic Basis for an Instructional Video: What Are the Implications for Novice Programmers?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*, July 8–12, 2023, Turku, Finland. Association for Computing Machinery, New York, NY, USA, 450-456. `https://doi.org/10.1145/3587102.3588795`.

Svana Esche and Karsten Weihe. 2023. Case Study on the Terms Novice Programmers Use to Describe Code Snippets in Java. In *IEEE Transactions on Education*, vol. 66, no. 6, pp. 642-653, 2023, `https://doi.org/10.1109/TE.2023.3290259`.

Svana Esche. 2024. Testing Programming Aptitude through Commonsense Computing. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*, January 29-February 2, 2024, Sydney, NSW, Australia. Association for Computing Machinery, New York, NY, USA, 104-113. `https://doi.org/10.1145/3636243.3636255`

Svana Esche. 2024. Rubric for the Quality of Answers to Student Queries about Code. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE '24)*, March 20-23, Portland, OR, USA. Association for Computing Machinery, New York, NY, USA, 331–337. `https://doi.org/10.1145/3626252.3630918`

Acknowledgements

First and foremost, I would like to express my sincere gratitude to my supervisor Prof. Dr. Karsten Weihe. Without him, I would not have embarked on this academic journey. I am deeply grateful for the mentoring, the opportunity to grow up academically with the freedom to conduct independent research while always having an open ear.

Additionally, this endeavor would not have been possible without the support from the “Qualitätsoffensive Lehrerbildung”, who partially financed my research. This is a joint initiative of the Federal Government and the Länder which is funded by the Federal Ministry of Education and Research.

I would like to extend my sincere thanks to Dr. Guido Rößling for his insightful comments and feedback during the paper writing process and his helpful insights into the computing education community.

Many thanks to the entire working group of Algorithms at the Department of Computer Science. Even if my research topic does not deal with trains, they have been very welcomed to me. Here, a special thanks to Julian Harbath for the shared lunchs and walks during the writing of this thesis.

A special thanks also goes to the doctoral students from the Physics Education group. Despite the different topics, the discussions about STEM education in general as well as methodological issues were very fruitful. Kevin Schmitt, with whom there was a constant exchange from the beginning, deserves special mention here. Thanks also to the doctoral colleagues of the DDI group for the joint writing days in Schweinfurt.

Finally, I would like to thank my family and friends who have supported me in the completion of this thesis. Especially to my mother who takes care of everything every Tuesday. Benny, I am sure you know how much you have supported me and continue to support me. I will always be grateful to you for that. To my son Konrad for his sparkling joy and tireless interest in science.

Contents

Part I.

Synopsis

1. Introduction

> *The Introductory Computer Programming Course is First and Foremost a LANGUAGE Course*
>
> Scott R. Portnoff, *acm Inroads*, 2018

Portnoff here links programming language and natural language, whose relationship is the overarching theme of this thesis. In particular, this thesis explores the question of what new insights arise from teaching-learning situations in post-secondary education on introductory programming when the connection between programming language and natural language is made explicit. This relationship in computer science, and in programming in particular, is the specific form of a more fundamental relationship. Vygotsky has stated the general connection between thought and language earlier [190]. We call this connection the first perspective of a total of three. Two further perspectives from other disciplines support this general connection. The first is the perspective of threshold concepts, which are already used in computing education research; the second is the perspective of linguists. The threshold concept links “to think like a Computer Scientist” [160] with using “appropriate language” [122] for the connection between thinking and language mentioned earlier. Some linguists, however, consider this connection to be one of the three functions of academic language, in which language acts “as a tool of thinking” [69]. The other functions are those of communication and social belonging. These three perspectives reveal the same thing, namely the importance of language.

Language has been a recurring theme in computer education research (CER) from a variety of perspectives. Longer trends related to language have been, for example, language as a cause of misconceptions due to overgeneralization [10], the use of metaphors [191], and language skills as a success factor in introductory programming courses

[102]. However, it is only recently that more attention has been paid to language and its use in teaching and learning. In terms of language and the terms used, there are more unanswered questions than answers in this area [33]. To date, teacher perspectives on language have been explored [14, 166]. Global terminology within the computing community [172] and the terminology of non-computing lecturers outside the computing community [11] have also been considered. However, the learner's perspective is still a gap. This is also true for the impact of an approach that explicitly brings together programming language and natural language. Research on non-native speakers of English and the obstacles they face, on the other hand, has recently increased, e.g. [3, 12, 103].

Research Question

The overarching research question of this thesis is *"What new insights emerge from exemplary teaching-learning situations in introductory programming in postsecondary education when the relationship between programming language and natural language is made explicit?"*

The research question formulates a focus on *exemplary* teaching-learning situations in postsecondary education. It is beyond the scope of this thesis to consider all teaching-learning situations. The selected exemplary teaching-learning situations should be related to the same programming language. In our case, this is Java, which is the programming language of the related Computer Science 1 (CS1) course at the Technical University of Darmstadt. The research question does not refer specifically to multilingual students or non-native speakers of English. Therefore, these areas of research are out of scope.

We define natural language as the language that participants in the CS1 course use to communicate with each other. In the case of this thesis, that is German. Whether it is in written form, as in feedback on homework assignments, or in oral form, as in faculty office hours. We use the term programming language when we need to refer to one.

We present some arguments why this research question and its findings are relevant for computer education research. In general, language on the one hand is related to thinking, communication, and belonging to the other [69]. Language and terms as part of language are thus key factors in thinking, communication, and belonging. Learning, just like daily life in general, involves all these three domains. This applies all the more to learning programming languages. There is some evidence that programming languages are just as alive in the brain as natural languages [150, 170]. Therefore, the significance of this work is based on the relevance of language and on the fact that this work brings this relevance to research practice.

Moreover, answering this overarching research question is a step towards the concept mentioned by Portnoff [150], which was cited in the epigraph. The concept is that CS1 is "first and foremost a language course" [150] and therefore uses teaching and learning strategies rooted in a focus on language. In this study, the focus is on language by making the relationship between programming language and natural language explicit. As empirical evidence, four empirical studies were conducted in this thesis, which are described in the next paragraph.

Methodology

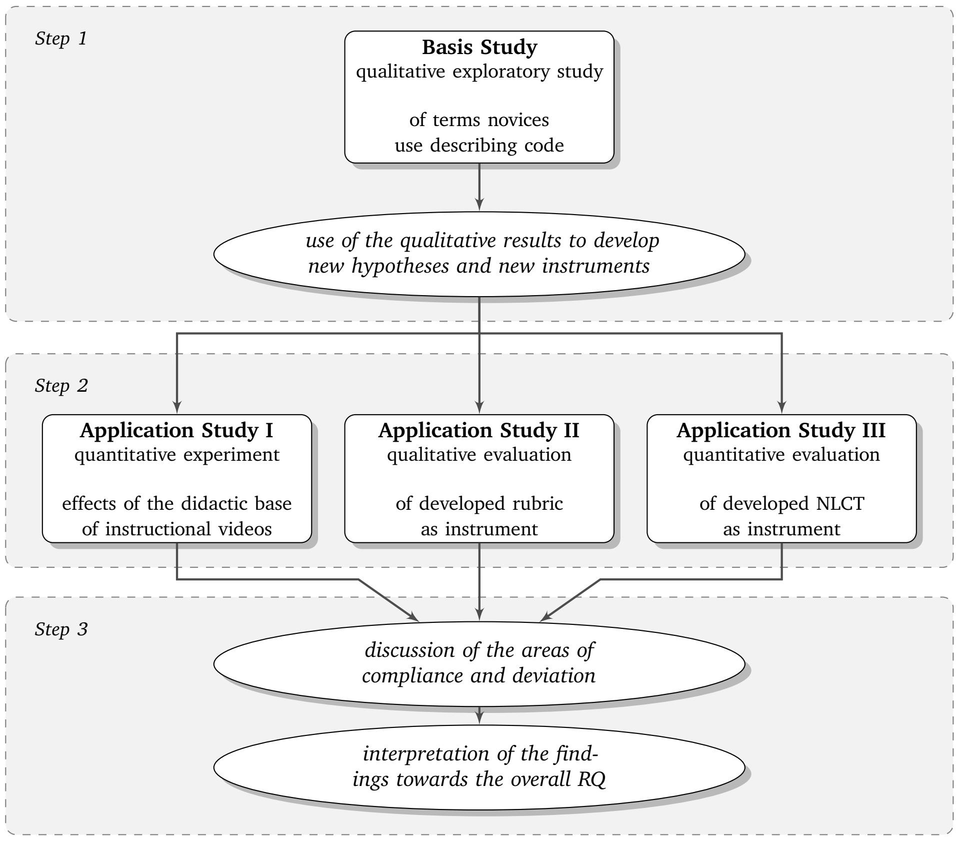
The methodology used for this overarching research question is a collection of four empirical studies with both qualitative and quantitative ones as a mixed methods approach. The interplay between these studies is illustrated in Fig. 1.1. This mixed methods approach with multiple empirical studies was intended to provide a comprehensive and multifaceted response to the research question.

All studies are based on primary analysis of self-collected data. An exploratory qualitative study was the focus at the beginning of the analysis on the research question, see Step 1 in Fig. 1.1. This study – called *Basis Study* in the following – examined terms novices used to describe code snippets in Java using a qualitative content analysis approach. Moreover, it led to hypotheses and ideas about the positive effects of making the relationship between programming language and natural language explicit. These hypotheses and ideas were to be tested in further studies called *Application Studies I to III*, see Step 2 in 1.1.

In *Application Study I*, a randomized experiment was conducted to investigate the effects of connecting natural language and programming language as a didactic basis for instructional videos. The effects were measured on both code writing performance and self-efficacy. Thus, this study involved the teaching-learning situation of instructional videos, which are part of instructors' teaching methods. The focus of the teaching-learning situation studied shifted to teaching staff in general in *Application Study II*. Here, a rubric[1] was developed to assess the quality of supporters' responses to student

[1]A rubric is a guide that lists specific criteria for grading by category and level.

Figure 1.1.: Dissertation research design: interplay of studies and mixed methods approach used for answering the overall research question (RQ).


Step 1
Basis Study
qualitative exploratory study
of terms novices use describing code
use of the qualitative results to develop new hypotheses and new instruments
Step 2
Application Study I
quantitative experiment
effects of the didactic base of instructional videos
Application Study II
qualitative evaluation
of developed rubric as instrument
Application Study III
quantitative evaluation
of developed NLCT as instrument
Step 3
discussion of the areas of compliance and deviation
interpretation of the findings towards the overall RQ

queries about the code. Methodologically, the development of the rubric was theory-based, while it was evaluated qualitatively. The evaluation was carried out both internally through the assessment of written responses and externally through expert interviews. As a final study, a programming aptitude test with tasks based only on natural language was developed and its measurement accuracy was quantitatively analyzed in *Application Study III*. This test was called Natural Language Computing Test, or NLCT for short. So this test was focused on the teaching-learning situation of assessing students' abilities.

In this study, inter-rater agreement analysis and item response theory analysis[2] were used. The latter was also used in Application Study I.

The final step, Step 3, was the synthesis of all study results to answer the overarching research question.

Main Contributions

This thesis has several contributions. The most important one is the development of a new meta-perspective for research and teaching in computing education. Four studies were conducted to empirically demonstrate the usefulness of this meta-perspective. This meta-perspective is the relationship between programming languages and natural languages. As preliminary work, we have identified and synthesized various fragments of research on language in introductory programming as aspects with a common denominator. As far as we are aware, this synthesis has not been done before. As a result of this preliminary work, we have shown that natural language and programming language have always had a relationship and have always been present in research.

Moreover, the thesis contributes by filling two knowledge gaps that have already been identified as such in its field. These include the research call on the terms used, which was contributed by Diethelm and Goschler [33]. To the best of the author's knowledge, the Basis Study is the first empirical study to address the two questions in relation to students in postsecondary education. In fact, this is the case even though the research call was published back in 2015. The second knowledge gap that was addressed was a methodological one, namely testing a theory by conducting a randomized controlled trial, i.e., an experiment, in Application Study I. In introductory programming research, experiments are rare [171] and even recent reviews have formulated calls for conducting more formal experiments [120]. Thus, conducting an experiment is filling a knowledge gap in itself. The same holds for the other two studies included in this thesis, i.e., the Application Studies II and III. Unlike the previous ones, they did not respond to explicitly formulated calls for research. However, assessments in general are a gap in CER which needs more focus in research [112]. The rubric developed in this thesis is the first tool to both describe and assess the quality of responses written by those who teach at different skill levels. The NLCT, in turn, is the first programming aptitude test that directly measures commonsense computing without relying on programming languages.

[2]This is a statistical modeling approach to estimating examinees' abilities based on their responses to test items, see Section 3.2.2.

Outline of the Synopsis

Chapter 2 narrows the literature presented from introduction to programming in post-secondary education in general to reasoning and its relationship to natural language in computing education. Areas of focus as well as gaps are identified. This is followed by a transfer to specific teaching-learning situations. Here, each exemplary teaching-learning situation corresponds to one of the four studies conducted.

In the third chapter, the general methods used are presented, see chapter 3. These are the mixed methods research approach in general and qualitative content analysis, item response theory, and statistical analysis as specific methods. A description of the participants and their context completes this chapter.

Study-specific methods and findings are reported separately in chapter 4. In this chapter, an overview of the methods and results of the publications is provided.

For discussion purposes, the findings are synthesized in terms of answering the over-arching research question, the significance of the findings and strengths are presented, and the limitations and threats are discussed in chapter 5. In addition, this chapter lists specific starting points for future work in teaching and research.

Finally, a general conclusion about the thesis as a whole is drawn in chapter 6.

2. Literature Review

We proceed in four steps to analyze previous research in computing education research (CER) in light of the overarching research question. As a reminder, the research question is as follows: *"What new insights emerge from exemplary teaching-learning situations in introductory programming in postsecondary education when the relationship between programming language and natural language is made explicit?"* First, we turn to the context in which the research question is situated: introductory programming in post-secondary education, see Section 2.1. In Section 2.2 we then turn to the relationship between reasoning such as programming language and natural language. Here we move from the general to the current situation in CER. This is followed by brief overviews of research areas in CER relevant to learning to program in Section 2.3. Finally, we focus on exemplary teaching-learning situations and their location in previous research, see Section 2.4. There we also present the subordinate research questions for the studies conducted.

2.1. Introductory Programming in Postsecondary Education

First, the term of our context – "introductory programming in postsecondary education" – is defined in Section 2.1.1. Then the foci and gaps of research towards introductory programming are highlighted, see Section 2.1.2.

2.1.1. General Definition

In this work, the term "introductory programming" stands as short-form for "introductory programming in postsecondary education." For ease of reading, we refrain from repeatedly mentioning the context of postsecondary education. Similar to the use of the term by Luxton-Reilly et al. [112], the focus on postsecondary education excludes courses and programs that take place either in schools or at out-of-school learning sites. By school, we mean K-12, i.e., formal education from kindergarten through 12th grade, such as college, high school, and others. We also exclude courses explicitly aimed at students outside of computer science, i.e., non-majors.

Similar to the previous understanding, the term CS1 is often used in the literature in addition to or as a substitute for introductory programming. The widespread use of the term CS1 is also evidenced by the fact that CS1 is a separate concept in the ACM Computing Classification System [2]. Because both terms are widely used, CER reviews often use both search terms – introductory programming and CS1 – synonymously, for example [15, 112, 120].

Similar to the not standardized terminology, the content of CS1 courses varies [75, 111]. Hertz describes the situation in drastic terms: "So while we often discuss CS1 and CS2, these courses are taught so differently as to make them nearly meaningless." [75]. As a way out, they recommend including course descriptions and other information to provide context. One glimmer of hope is that the results of one introductory programming course may not be transferable to all other courses, but they are transferable to courses that cover the same content. Nonetheless, much research has been conducted to identify and measure common concepts in introductory programming. The foci and gaps in research on introductory programming are presented in the next section.

2.1.2. Foci and Gaps

For categorization, we use a theoretical framework to examine each aspect and its interactions in teaching-learning situations. The particular theoretical framework used is the didactic triangle, whose origins date back to the 19th century. This framework links the two individuals in a teaching-learning situation, namely the student and the teacher, to the content that is being taught or learned. Thus, the student, the teacher, and the content are the corners of the triangle.
The triangle found its way into CER in the 2010s [17]. Since then, it was used to organize papers [112] and theories [113] in reviews and identify gaps in research [31]. Two further categories are added in this thesis, namely that of assessment and that of research approaches. The former was already introduced before [112]; the

latter was added as an additional category to assess empiricism in the research on introductory programming. Thus foci and gaps are described along the following categories: student, teacher, curriculum, assessment and research approaches. Here we use the term curriculum instead of content, similar to Luxton-Reilly et al. [112]. For a quantitative appraisal, we refer to their data. Luxton-Reilly et al. [112] categorized 1350 papers according to the one area on which a paper focuses. The domains are student (40%), teaching (37%), curriculum (17%), and assessment (6%); percentages are rounded. In addition to the general trends described, we present specific focus areas and gaps described in other reviews [32, 66, 93, 120, 154] specific to CER. The description of foci and gaps is intended to ensure that the studies in this work actually address gaps.

We begin with the area of most research, students. Here we describe the foci and gaps from the perspective of the student alone, the student versus the teacher, and the student versus the curriculum. In 2009, Kinnunen et al. [93] described a gap related to the second perspective. Similar to them, we are unable to cite a reference for this area, indicating a gap that still exists. In the first perspective, the research is mainly about two questions. The first is about who the students are, focusing on gender and underrepresented groups in general. The second question is about what students bring to the table, considering a variety of characteristics and competencies. These include demographic and psychological characteristics such as age and self-efficacy, as well as academic competencies such as prior programming experience and math skills. A particular focus is on predicting learning success and identifying students who are at risk of failing the course [68]. In contrast, the research on the student versus the curriculum makes up the lion's share [112, 93]. Luxton-Reilly et al. [112] describe the focus in that area on measuring students ability and their attitudes. Here, measurement is understood to mean what achievements can be inferred from student performance, as opposed to focusing on the design and approach to measurement. The latter would fall under the category of assessment. Less but not little focus is also put on code literacy, students behavior and their engagement. Sparse research is on developing theories in this field and the experiences of the students. Perceived difficulty has received more anecdotal attention than thorough research [13].

Those who teach fall into the teaching category in the review by Luxton-Reilly et al. [112]. Here, teaching accounts for slightly less than students, who make up the largest proportion. Papers on tools that support teaching and learning and papers on teaching techniques and activities together account for about 80% of the total 497 papers reviewed. The focus is primarily on papers about tools, as they alone account for half of the papers. The aspects of teaching and learning that the tools address are very diverse. Examples include visualization, games, programming environments, progress monitoring, and more. In contrast, models and theories and those who teach are rarely researched, indicating a gap. The lack of research on teaching staff is particularly surprising given that the teacher, as a representative of teaching staff, is one of the three corners of the didactic triangle. Thus, teaching staff play as important a role as students in the teaching-learning situation, and further research on them is needed.

The curriculum area is the second least researched part of the didactic triangle, accounting for 17% of the papers [112]. Luxton-Reilly et al. [112] identified three sub-areas, namely the curriculum itself, programming language choices, and programming paradigms choices. Here, the focus is on the curriculum itself, as it accounts for half of the papers reviewed. Nevertheless, there is a gap in research in this area, as it is mostly experiential [112]. Therefore, the findings to date lack empirical evidence. In programming languages, the focus is on developing programming languages for non-native English speakers, comparing programming languages, and providing advice on selection. In particular, comparison is also a focus in the research on programming paradigms.

The area of assessment is the least researched area [112]. Therefore, assessment as a whole is a gap. Of the 1350 papers, only 89 were categorized as assessment, representing 6%. Luxton-Reilly et al. define assessment broadly as "being set by the teacher and used to assess the student's grasp of the curriculum" [112]. There are several ways to categorize assessment. One categorization distinguishes between formal assessment and summative assessment. The former focuses on diagnosis with the goal of further learning, while the latter, such as exams, aims to provide a summative description of the learner's learning. Another possible categorization is when assessment occurs, namely at the beginning, middle, or end of the course. However, CER's focus is on exams, which fall into the categories of summative assessment and end-of-course assessment. Research on exams includes exam questions and the type of exam, such as paper-and-pencil or lab-based. Another focus in the area of assessment is the automation of assessment and feedback tools and the use of test suites for assessment tools. In this area, references to theory are particularly rare, even if it would strengthen the validity of the assessments.

Finally, we focus on research approaches. The most recent review period includes papers published between 2014 and 2015 and reviewed by Heckman et al. [66]. In these papers, the focus is on quantitative research, which accounts for two-thirds of all papers. The samples examined included a median of 72 participants. There is a notable lack of large-scale studies, with studies involving more than 1000 participants accounting for less than one-tenth. The distinction between interventional and observational approaches is balanced, as these two types account for about half of the papers. An example of an interventional approach is comparing errors in assignments between different integrated development environments (IDEs) and for an observational approach is how often teaching assistants use illustrations to explain something to students. However, Heckman et al. [66] did not specify what proportion of intervention studies were randomized controlled trials, i.e., experiments, as opposed to quasi-experimental designs. In quasi-experimental designs, the study compares groups with different treatments, but group assignment was not randomized. In summary, there are gaps in both replication and meta-analysis. To make matters worse, many papers do not consistently adhere to the reporting standards required for replication. These include, for example, a detailed description of the context of the study, a clear definition of data collection, and the provision of all survey questions.

In conclusion, most of the research focuses on the student and the tools and techniques for teaching, which leaves open many opportunities to fill gaps in CER. Especially when considering assessment as the main topic. In addition, particular gaps occur when the gaps from the different areas are combined. An example of this is research in assessment, which in itself fills a gap, combined with validation and standardization as a gap in research approaches. Again, assessment is a gap, as are competencies and professional development of teaching staff, so assessment of these areas is a particular gap.

2.2. Reasoning and its Relation to Natural Language

After an introduction to research on introductory programming, we take a closer look at the topic of the research question: reasoning and its relationship to natural language. First, this topic is treated in general terms (Section 2.2.1). Then, the current state of research on natural language and reasoning in CER is discussed (Section 2.2.2).

2.2.1. General Relation

> *"But it is obviously not language per se that makes the difference; rather, it seems to be the use of language as an instrument of thinking that matters."*
> [23, p. 14]

Similar to this quote, several theories from different subjects, disciplines, and decades have emphasized the connection between language, thinking, and learning in general. Thus, language is an important factor in academic success.

In general, Vygotsky laid the foundation for the connection between language and thought [190] as early as the 1930s. With respect to teaching, Kempert et al. [90] recently transferred this connection to content learning. Interdisciplinarily, they present language-based scaffolding as an important concept for teaching in general. Scaffolding in teaching is a supportive concept for students that is becoming less needed as they gain experience. However, the extent of the strength of the interaction between language and thinking is described as depending on the subject. Therefore, subject teaching must also use subject-specific methods for teaching. Other perspectives from other disciplines also support this connection between language and thinking. The same connection is made through the idea of threshold concepts [121]. The ability to express oneself "in appropriate language" [122] is part of one's understanding in a particular domain.

Linguists have recognized the interconnectedness of language on the one hand and, for example, thinking on the other in a broader sense. From a linguistic perspective, Morek and Haller consider this interconnection as one of the three functions of academic language, in which language functions "as a tool of thinking" [69]. In addition to this function, according to linguistic theory, language has two other functions, namely "as a medium of knowledge transmission (communicative function)" and "as a a ticket and visiting card (socio-symbolic function)" [69]. Besides, Kempert et al. [90] had additionally described communicative thinking and its importance for teaching. The importance of language as a tool for both thinking and communication was also described by Lemke [104]. Lemke [104] argues that talking about science is doing science through the medium of language. With over 7,500 citations, Lemke's [104] work has greatly influenced the entire field of science education.

Overall, all of these perspectives demonstrate the same two things: both the importance of language and its strong connection to thinking. Based on this argument, the next section presents the current situation of considering language in computing education. The three functions of language presented here serve as a framework to categorize the research that has already been conducted. Nevertheless, this thesis focuses on the epistemic function, that is, language "as a tool for thinking" [69]. This does not mean, however, that the other functions play a less important role or are excluded altogether.

2.2.2. Current Situation in Computing Education

Consideration of natural language (NL) has been part of CER since the early 1980s, e.g. [101, 102, 126]. It has played only a minor role, but is currently increasing. We have identified six perspectives on how researchers have included or addressed language in their research on computing education. These are categorized into earlier but persistent trends and more recent trends. The earlier trends are: (1) the *source perspective* with NL as a source of misconceptions, (2) the *predictor perspective* with NL as a predictor of success, (3) the *metaphor perspective* with NL both as source for scaffolding and errors, and (4) the *perspective of commonsense computing* through NL as longer trends. In contrast, more recent trends are: (5) the *linguist perspective*, such as second language acquisition instruments and non-native English speakers (NNES), (6) the *comparative perspective*, which examines how programming language (PL) and NL differ both in general and in neural execution.
Not each of these perspectives is equally relevant to our overarching research question. The comparative perspective and the NNES strand are outside the scope of our overarching research question. Therefore, we will devote less space to presenting research on these perspectives and more attention to the other perspectives. Despite the following categorization by perspective, some publications contain multiple perspectives. For example, Qian and Lehman [155] combined the NNES strand with the predictor perspective when they studied Chinese students' programming skills. However, the publications are assigned to one main perspective to provide a clear and comprehensive overview of the current situation in CER.

Source perspective

First, we present the results according to the source perspective. As early as the 1980s, natural language was listed as a possible cause of misconceptions [39] and continues to be listed today [154]. This includes recent textbooks for educators in computing, e.g., [176]. In programming languages, keywords have fixed, specific meanings that do not necessarily correspond to their multiple everyday use in natural language. Here, the transfer from natural language to programming can cause misconceptions. Pane and Myers have described this problem as a "human interpreter problem" [141]. Below we list references for documented incorrect transfers from old to new. In 1983, Bayman and Mayer [10] described an incorrect transfer of the BASIC code `INPUT A`, namely adding the letter A to computer memory instead of taking a number as input and storing it in variable A. With Pascal as PL, novices think that the condition of a conditional

loop with the keyword `while` must have a true Boolean value throughout the loop, not just once per iteration [20]. Here the keyword `while` was used in the sense of "throughout". Similarly, Pea [145] described that the condition in the condition block with the keyword `if` continuously waits for the condition to have a true Boolean value; even outside the usual control flow. This behavior is identical to that of the word "if" in natural language. In 1986, Du Boulay [39] compared the temporal sequence of the term "and" in NL as opposed to the Boolean operator `AND`. More recently, Miller [123, 124, 125] investigated another area where the transfer from NL to PL fails, namely reference-point errors. Miller identified metonymy, as used in NL, as a possible cause. Metonymy means that structurally related elements are taken instead of the intended ones. For example, novices take an attribute as a reference for a whole object. Guo et al. [59] collected 16,791 learners' written explanations of errors they encountered while programming in Python for three years. Among these, many misconceptions were due to incorrect mapping between NL and Python as a programming language. These include omitting quotes for strings, capitalizing the beginning of lines of code, confusing singular and plural forms of nouns in collections, referring to variables by their type instead of their name, placing function calls as verbs, abbreviated concatenation conditions, and iterations that read like English.
In summary, the incorrect transfer of NL to PL is indeed one of several sources of error and misconceptions. However, we believe that avoiding NL is not the right solution, because students do bring their NL with them. Instead, students can be explicitly taught about the incorrect transfer so that they build their negative knowledge. Negative knowledge is based on the assumption that "to know what is wrong helps in understanding what is right" [137] and has already found its way into educational research.

Predictor perspective

Second, the focus is shifted to the predictor perspective. Here, the research base on language as a predictive factor is thin and has yielded mixed results. Hellas et al. [68] conducted a systematic literature review on predictive factors that included 357 reviewed papers from 2010 to mid-2018. They also used language as a predictive factor category and considered 11 of the 357 papers. According to the authors, the exact assignment of papers to factor categories can no longer be traced. For both native and non-native English speakers (NNES), the results are mixed, with the latter having additional barriers [12]. For students from Indiana, USA, both English unit and verbal scores on the scholastic aptitude test correlated significantly with their grade in the introductory programming course [102]. Byrne and Lyons [24] examined the English and foreign

language skills of Irish students. These did not correlate significantly with exam scores in their introductory programming course. For NNES, English proficiency had the greatest impact in explaining differences in programming among Chinese middle school students [155]. South African students' English scores did not significantly correlate with their programming performance in an introductory course [9]. Ameri et al. [6] included language in the form of scores in English and reading on American College Testing (ACT) in their prediction framework without listing its particular effects. Prat et al. [152] compared second-language aptitude of learners with other predictors for learning Python in terms of learning rate, declarative knowledge, and programming accuracy. This aptitude explains 17% of the outcome variables.
We would like to introduce a new aspect for future research that could be a possible reason for the mixed results. When considering the influence of language, we should distinguish between language in the prosaic sense, e.g., English literature essays, and language in STEM subjects with its ideal characteristics such as precise, structured, logical, and operational.

Metaphor perspective

Next, we turn to the perspective of *metaphors*. Similar to the the usage in CER, we adopt here Indurkhya's [84] formulations for target and source: the thing that the metaphor describes is the target, and the concept used in the metaphor is the source. For example, a cooking recipe is a metaphor for a subroutine in programming. In this case, the subroutine is the target and the cooking recipe is the source [164]. In CER, the potential usefulness of metaphors has been assessed in mixed ways. This goes back, among other things, to Dijkstra [35] recommending in an opinion piece that other educators not use metaphors. Guzdial [60] has recently argued against this. In turn, the misuse of metaphors may explain students' misconceptions. For example, Kohn [94] attributes the misconception that a variable can store multiple values simultaneously to the box metaphor. Holland et al. [80] advise against using a static collection of data, such as the data from a CD, as a metaphor for an object because the dynamic behavior is not evident in it. One limitation of metaphors is that students attempt to extend a metaphor beyond the correct connection between two domains. For example, the previous example between the box and the concept of variables can be seen as an inappropriate extension. However, teachers tend to be less aware of this limitation. In an interview study of metaphors, only one out of ten teachers stated this limitation [164]. On the other hand, figurative paraphrases and metaphors are among the tools of pedagogical content knowledge for teachers [164, 191]. In addition to listing 20 different metaphors used by teachers, Sanford et al. [164] provide also categorization of metaphors. More recently, research has shown interest in new forms of metaphors, such as oral metaphors [78] and embodied metaphors [114]. The research also examined the effects of using different metaphors for the same concept, in this case variables, [74]. Despite ongoing research on metaphors, the area of student understanding of metaphors, previously described by Sanford et al. [164], remains a gap in CER.

Perspective of commonsense computing

Fourth, as the last longer trend, we present research results on commonsense computing formulated in NL. We have used the term “commonsense computing” here from a line of research by Simon, Sanders, and McCartney, e.g. [28, 173, 186], who coined the term. The term describes the thought structures used in programming that students who have not received formal training in programming bring with them. As for sorting, 57% of novices who had no experience with PL actually had commonsense computing skills, as measured by correctness [28, 173]. The sorting task was to arrange a set of numbers in ascending, sorted order and describe this process in English. Here, the process description corresponds to writing a program using natural language. Miller [126] and Onorato and Schvaneveldt [136] asked participants to formulate a similar process description, but with the task of formulating instructions for combining employee lists [126] and telephone directories [136]. Miller found that these descriptions include standard concepts of programming languages, such as testing of attributes, albeit control actions as iteration, sequencing, termination, and full conditionals are rare used. Onorato and Schvaneveldt [136] found that non-programmers and novices follow a straight path in solving this task, while experts consider all contingencies and describe appropriate cases. This straightforward path was executed step-by-step by non-programmers rather than using repetitive structures used by novices and experts. Galotti and Ganong [53] described that non-programmers were indeed able to use control statements according to the control flow when writing instructions. In this case, it was about how to play a certain card game and how to create a list of certain employees. Pane et al. [142] examined descriptions of the game PacMan by non-programmers, both children and adults. In contrast to the results of Galotti and Ganong [53], participants used descriptions that were more consistent with event-based programming than with imperative programming style. They also preferred to process operations on multiple objects using sets and subsets rather than using loops. Based on these results, they developed HANDS, a language that works the way non-programmers expect [140, 132]. However, other research addressing the question of whether natural languages such as English are suitable as full-fledged

programming languages is not considered here. Despite their long history, beginning with the first paper in 1966 [162], they are not the subject of this thesis.
In summary, students have commonsense computing skills, although to varying degrees. There is a lack of large-scale studies in this area, recent studies, and studies with non-native English speakers who formulate commonsense computing skills in their language. It also remains an open question how the transition from commonsense computing to formal programming skills works.

Linguistic perspectives

Next, we focus on recent trends in the perspectives addressed. In particular, research that can be called linguistic perspectives is gaining momentum. This includes several strands, such as how the acquisition of PL and NL resemble each other, instructional materials based on second language acquisition, a focus on terms and classroom language, and non-native English speakers (NNES). Most of the following research was published more recently, after the mid-2010s. With the exception of Ledgard [101], who published an approach to switching syntax from PL to NL as early as 1980. This approach resulted in fewer errors and was preferred by participants compared to the traditional syntax.
Next, we describe the various strands. First, we focus on how the acquisition of PL and NL are similar. Fedorenko et al. [47] hypothesized that there are parallel processes between comprehension of language and code on the one hand and generating language and code on the other. Both processes were described as consisting of six phases that are identical for both language and code. For example, the words of language correspond to the "keywords/identifiers/function names" [47] of code. For comprehension, the process begins with the sequence of symbols and ends with the meaning of the language or the goal of the program code. For generation, again, the process begins with the meaning or goal and ends with the finished text or code. Portnoff argued that the similarities being even deeper and hypothesized that an "introductory computer programming course is first and foremost a language course" [150]. In consequence, these programming courses should abandon explicit rule-based syntax instruction, which is outdated for learning languages. Instead, implicit learning techniques borrowed from principles of second language acquisition should be included in the courses. These three principles listed by Portnoff [150] are: (1) focus on specific features but use them variably to emphasize differences in meaning, (2) transform highly condensed syntactic features in their unrolled form (like all repetitions of a loop), and (3) ongoing exposure of the optimal solution to build and memorize archetypes of solutions.

These considerations lead us to the second strand, which emphasizes how these similarities can be put into concrete practice through the use of second language acquisition (SLA) strategies. As specific tools, researchers have developed gradual languages such as Hedy [54, 72] or AryaBota [158]. Both languages support gradual development from natural language syntax to programming syntax rooted in SLA. Task formats were also adopted from SLA, such as perfect memorization [151]. Here Parry adapted [143] several formats, including color coded words, fading cloze exercises[1] for writing code, and mapping exercises between definitions with program statements. Furthermore, in a quasi-experimental pre-post study, Parry [143] showed that additional language-focused exercises had a positive impact on student learning.
Another SLA strategy investigated by researchers Hermans and Swidan was reading aloud [73, 180]. This strategy improved recall of the correct syntax of programming concepts, but not comprehension in terms of tracing and understanding the concepts [180]. Moreover, Izu [86] presented their ongoing research project which puts Portnoff's [150] ideas of SLA into practice. However, their findings are not published yet.

Another strand addresses terms and classroom language. Here, Diethelm and Goschler [33] proposed four open questions on terms and language in spoken form in conjunction with a call for research in this area. These questions address "the terms (1.), their usage (2.), recommendations for terms (3.), and recommendations for their usage (4.)" [33, p. 23]. So far, there is only sparse research which has taken up this call. There are two studies for the perspective of teachers in the field of language [14, 166]. For the students' perspective, Hermans et al. [73] conducted an exploratory study how 20 high school students aged 11 to 13 years used terms when reading aloud. In addition to their different focus on K-12, they also focused more on phonological issues like the pronunciation and promoting reading lines of code aloud in the classroom. In contrast to the two other studies, they did not relate to the prior mentioned research call. The opposite perspective on terms was explored by Stefik and Gellenbeck [177], who examined the syntax preferred by programmers and novices for certain terms and concepts. For example, the preferred syntax for the concept of the keyword `return` was "return" for programmers and "provide" for novices. They also examined how these two groups rated the intuitiveness of the syntax of nine different PL.
When not focusing exclusively on programming, there are other publications on terms. Holmboe [81] examined novice terms in data modeling and Gold-Veerkamp et al. [56] presented a research approach to terms in software engineering. However, to the best of the author's knowledge, the latter research approach was not followed up. From

[1]In a cloze exercise, parts of the text or, in this case, the code are masked or presented only as gaps that students must fill in.

a broader perspective, both the global terminology within the CS community [172] and the terminology of non-computing lecturers outside the CS community [11] were examined.

In contrast to the previous strands, the area of NNES is outside the focus of this thesis as outlined in Section 1. Therefore, we only refer to a recent review in this area, namely [103]. However, it is important to note that NNES face additional barriers [12] and therefore knowledge of terms is particularly important to them [3].
In summary, the linguists' perspective encompasses a variety of strands. They are all attracting increasing interest, especially in recent years. We believe that the peak of interest has not even been reached yet. In particular, new teaching tools and methods offer promising opportunities to improve the learning of programming. To conclude this perspective, we refer to a chapter from a textbook for educators that summarizes language-based research and its orientation to classroom practice [34].

Comparative perspective

The sixth and final perspective presented are the research findings from the comparative perspective. In general, programming languages are distinguished from natural languages by a higher typing speed [42] and by more repetitiveness [25]. However, results according to the neural execution are mixed. Endres et al. [44] used functional near-infrared spectroscopy (fNIRS) in their study. Here, code comprehension and reading NL are mentally distinct areas for novices just started programming [44]. Several studies used functional magnetic resonance imaging (fMRI) as a method belonging to cognitive neuroscience [51, 85, 97, 170]. For code comprehension, research has both confirmed [170] and rejected [51] the activation of the same brain regions as in language processing. Here, neural dissimilarity decreased the higher the programming skills [51]. When writing code, computer science students used very different brain regions for writing code and prose [97]. Ivanova et al. [85] compared reading code with reading corresponding descriptions in natural language. The corresponding language description actually matched the language system, but reading the code did not.
In summary, most of these studies were conducted with small sample sizes of up to 30 participants. It remains to be seen what the relationship between NL and PL will be in future large-scale studies.

Conclusion

In conclusion, the following picture emerges of the current situation of research on language in the field of CER. Consideration of language has always been part of CER and continues today. The perspective from which language can be viewed is very diverse, as we have shown by looking at the different perspectives. At the same time, not all perspectives relate to the general relationship between language and thought mentioned earlier. Nevertheless, they contribute to the study of this relationship relating to computer science and, in particular, to programming. In the context of our overarching research question, the research strands of the linguistic perspective are of particular interest. Indeed, these relate to current contexts, such as modern programming languages. Moreover, they share the same focus and goal, namely practical implications for teaching. However, most of the linguists' research is not yet related to postsecondary education, but to the K-12 level. This strand of research is also still in its infancy and raises many unanswered questions, as Diethelm and Goschler [33] show. This thesis and its overarching research question address these gaps.

2.3. Research Areas relevant to Teaching-Learning Situations

The overarching research question is about making explicit the relationship between natural language and programming in teaching-learning situations, see Section 1. Next, we turn to areas of research relevant to teaching-learning situations in learning to program. The research presented here comes specifically from CER. This is the introduction to the teaching-learning situations under investigation, which are presented in the section below. As the first research area, we focus on students' understanding of the programming language, which is most evident in reading and writing code, see Section 2.3.1. Second, in section 2.3.2, we provide an overview of the measurement tools used to assess students' understanding of the programming language. Third, we focus on how instructors use teaching methods to improve students' understanding of the programming language, see Section 2.3.3. Finally, in Section 2.3.4 we focus on all those who support students in their learning process of programming. All four sections provide a brief overview of research in the related areas. Based on these general overviews, subordinate research questions on specific teaching-learning situations are derived in Section 2.4. These specific teaching-learning situations, in turn, are each located in the areas outlined here.

2.3.1. Students' Understanding

Students' understanding of programming language is reflected primarily in basic activities with code expressed in programming language: reading, tracing, and writing code. In these activities, students often make mistakes. Therefore, research on both reading, tracing, and writing code and misconceptions sometimes root in the same data analysis. In consequence, this section first presents an overview of research approaches followed by the research on reading and writing code. Finally, we focus on the examination of the errors, (mis)conceptions, and mental models associated with the concepts used in reading and writing code.

The research approaches used vary widely. On the one hand, there are large-scale data analyzes with nearly 100 million compilations of written code from more than 900 thousand users [22]. On the other hand, studies with interviews, e.g. [27], or the use of exam data, e.g. [187], have smaller sample sizes.

Much of the research has focused on reading, tracing, and writing code and the relationships between them. This is evident in many publications, including an ITiCSE working group[2] report that focuses specifically on code reading and tracing [106]. Several studies reported a correlation between reading code and writing code [109, 131, 167, 187]. In addition, Lister et al. [107] proposed a hierarchy of these skills. In contrast, Xie et al. [192] defined distinct stages between reading and writing code. As an example of assessments, we refer to the rubric assessing students' responses to the "Explain in plain English" tasks [26]. These tasks involve both reading and tracing code with the additional requirement of abstracting the purpose of the code. Moreover, code completion is an emerging trend to complement code writing tasks, especially since these tasks correlate well with code writing tasks [38, 174]. Code completion involves filling in blanks [174] or the Parsons Puzzle[3], which Du et al. have summarized in a review [38].

As already mentioned, there were from the beginning, e.g. [39], and there is extensive research at CER to this day on what errors and misconceptions students have. For a recent overview, we refer to Qian and Lehman's review [154]. They categorize the

[2]According to ITiCSE 2023, this is an "intense collaboration between five to ten researchers from around the world to produce a high-value report on a topic of interest in computing education." [135]

[3]In a Parsons puzzle, students must choose from a subset of predefined code fragments and put them in the correct order to create a solution for a given program.

difficulties into syntactic knowledge, conceptual knowledge, which is closely related to mental models, and strategic knowledge. Mental models, in turn, are similar but have a broader meaning, namely a particular person's understanding of a phenomenon. In 2023, Heinonen et al. [67] described in their review that interest in research on mental models is declining despite their widely recognized importance. Most research on mental models pre-dates the year 2000 and is therefore based on outdated programming languages. In addition, the development of mental models is underdeveloped, leaving a gap for current research.
Overall, research on student understanding is well researched, but not all perspectives have received equal attention over the decades. Moreover, because of modern programming languages and the different contexts in which students find themselves, there is always a need to investigate their understanding.

2.3.2. Assessment of Students' Understanding

We provide a brief overview of assessment research in chronological order, depending on when assessment occurs in the introductory programming course. Thus, the chronological order consists of before or at the beginning of the course, during the course, and at the end of the course. Finally, we address the general question of the valid basis for assessment.
Both self-assessment instruments and surveys were used to assess prior knowledge of programming before or at the beginning of the course. Self-assessment instruments were created for basics [41] and for advanced topics [48]. Surveys have asked, for example, about programming languages [61, 91, 179], lines of longest program [91], type of experience [79, 91, 179], fluency [179], and usefulness of experience [179].
Assessments during the course play a smaller role than the final course performance assessment [112] and some tools can be used for both scenarios. The Foundational CS1 (FCS1) Assessment [184] is cited as an example and is particularly noteworthy because of its validation and standardization. However, in-course assessment has a growing research in the area of automated assessment [139].
Typically, exams with smaller tasks or lab exams with a large programming task are the standard method of assessing final course performance [112]. Developing and grading CS1 exams with traditional code writing questions is considered difficult [197]. This is particularly noticeable because a large portion of the exams consist of questions about writing code rather than reading code or short-answer tasks about programming concepts [146].

Finally, we focus on the valid basis and standardization of the assessments used. Unfortunately, they play a minor role in CER. Margulieux [115] reviewed the standardization of measurements used in CER in papers and listed 17 computing-specific standardized instruments developed or used in the 197 papers reviewed. However, they conclude that most papers did not use these standardized instruments, despite the existence of an appropriate instrument for many of the constructs measured. In contrast, Taylor et al. [182] list frequently used informal assessments, such as the rainfall problem, e.g., [49]. As described in Section 2.1.2, the whole area of assessment is a gap, as only 6% of papers fall into this category [112]. In addition, the lack of valid foundations and standardization described above calls for a greater focus of *research* on assessment. Luxton-Reilly et al. [112] also list feedback on assessment as a subcategory in their review, which is similarly underdeveloped as the other areas of assessment such as design and analysis. Overall, all parts of assessment need a greater research focus at CER.

2.3.3. Instructors' Teaching Methods

Instructors use a variety of teaching methods, but no silver bullet has yet been found. In a recent review, Kanika et al. [89] identified five common research areas for tools and teaching methods for introductory programming. Before listing them, we should mention that the earlier review work by Vihavainen et al. [188] reached similar conclusions. The first area is visual programming, where programming languages can be manipulated in a graphical way. The development of visual programming languages went from flow-charts to block-based ones, where Scratch and Alice belong to the most widely used examples [89, 134]. For using these programming languages specialized integrated development environments (IDEs) are required. Visual programming has several advantages, such as good accessibility and ease of use due to low knowledge overload [134]. The second area identified by Kanika et al. [89] is game-based learning, also called gamification [195]. This involves teaching methods and tools that use elements from games [195], game development, or gaming through programming [89]. The effectiveness of this approach has been demonstrated in a meta-analysis, particularly for student motivation and academic achievement [195].
Pair and collaborative programming forms the third area where students work in pairs or small groups when writing programs. Here, students write programs in pairs or small groups. Hawlitschek et al. [65] summarize the research on pair programming in a recent review. The empirically based results mostly show that pair programming

leads to higher programming performance than programming alone [65].
Fourth, robot programming is a long lasting and ongoing approach [89]. Here, education-specific robots are controlled by the programs that students have written. Its benefits include increased student interest and motivation, and broadening participation [8].
The last area identified by Kanika et al. [89] is assessment systems. Here, software tools check and grade students' programs, sometimes supported by feedback in natural language. As described earlier, this area is currently in growth; see the work of Paiva et al. [139] for an overview.
In addition, we would like to mention the growing trend of flipped classroom [4]. Here, hands-on experiments are the most common in-class activity and watching videos is the most common out-of-class activity, occurring in nearly 60% of the classes studied [70]. A meta-analysis found that this approach significantly improved student performance in programming courses [5].
Overall, there are a variety of teaching methods and tools that instructors can draw upon, and new ones are being developed all the time.

2.3.4. Support of Students by Teaching Staff

In this thesis, we have a broad understanding of teaching staff that includes all supporters such as instructors, lecturers, student teaching assistants (TAs), and teachers. This section focus on the teaching staff themselves with their competencies and perceptions and not the teaching techniques and the support they provide.
In contrast to the usual descriptions, teaching assistants are also included as teaching staff because their integration in CS1 is described as a best practice [149]. Moreover, they have a significant positive impact on student learning [46, 148]. However, research specifically addressing TAs is only a small part of CER.
High-quality student support requires that all the teaching staff possess a variety of competencies, including content knowledge, pedagogical content knowledge, and pedagogical knowledge. We refer to Hubbard's review [82] for research on pedagogical content knowledge specific to CS. Professional development (PD) addresses the formation of these necessary competencies. Research on the PD of teachers and teaching assistants is sufficiently developed for reviews, although research on teachers [133] is richer than that on teaching assistants [127]. However, research on teachers is outside the scope of this thesis as it focuses on teaching-learning situations in postsecondary education. For teaching assistants, the PD typically includes a workshop format with role-playing techniques [127]. In contrast, to the best of the author's knowledge, there

is no research on the PD of instructors and lecturers. However, not all instructors and lecturers responsible for CS1 are also part of the computing education community and therefore have appropriate competencies for providing support.
Similarly, the perceptions, reflections, and beliefs of instructors and faculty regarding the support they provide also remain a gap. Next, some exceptions are presented. Kinnunen et al. [92] specifically emphasized that their goal was to find out what instructors perceive as the factors that influence student success, not what those factors might be. Patitsas et al. [144] asked CS instructors about their beliefs of bimodal grade distributions and whether they identify given grade distributions as bimodal. McCartney et al. [119] combined the two concepts and examined educators' beliefs about the geek gene. Even more interestingly, this gap in perceptions and reflections is even rarely recognized as such. One exception is in the area of live coding, where Selvaraj et al. [165] specifically pointed out this gap.
Overall, the field of research on teaching staff support of students is underdeveloped when focusing on faculty themselves. Consequently, there are more unanswered questions overall than answered questions.

2.4. Conclusion and Derived Subordinate Research Questions

As already described, this thesis cannot examine the insights of making the relationship between programming language and natural language in general, but only in specific situations. Thus, we will look at exemplary teaching-learning situations according to the overarching research question. As a reminder, we pose the overarching research question from the perspective of the different actors in introductory programming: students, instructors, and teaching assistants. The perspectives of all these actors are addressed in the following specific teaching-learning situations. Moreover, these teaching-learning situations are both located in the research on language in CER and in the previously outline research areas which are relevant to teaching-learning situations.

In selecting exemplary teaching-learning situations, it is important to consider the context in which this work was written. During the phases of research design and data collection, the constraints of everyday university life were particularly drastic due to the Corona pandemic. For example, the small group discussions in CS1 did not take place on site. Therefore, they could not be observed. The thesis determined that only teaching-learning situations that take place in online courses would be selected. The selected teaching-learning situations are presented below, followed by a location in previous research.

(S1) Students describe the code as they read it. This can be done more or less formally; describing only to oneself in thought also falls under this situation.

(S2) Students solve code writing tasks.

(S3) Students watch instructional videos that scaffold the student learning process.

(S4) Instructors assess students' prior programming knowledge.

(S5) Teaching staff answers students' queries about the code in writing.

These teaching-learning situations are located in the previously outlined research areas as follows. The teaching-learning situations S1 and S2 belong to the area of students' understanding of programming languages, see Section 2.3.1. Moreover, they are a genuine part of learning programming and have probably always been present in all introductory programming courses. The use of instructional videos in situation S3 is one of the many possibilities of instructors' teaching methods described in Section 2.3.3. Situation S4 corresponds directly to the chronological first assessment of student understanding mentioned in Section 2.3.2. Last but not least, the formulation of written explanations, as described in situation S5, is a concrete way in which teaching staff support students. Therefore, situation S5 is located in the appropriate area described in Section 2.3.4. Finally, we describe the transition between the presented literature, the selected exemplary teaching-learning situations, and the following studies. The methods used in the conducted studies to answer the subsequent subordinate research questions are presented in the following section 3.

As the first study, the Basis Study approaches the overarching research question exploratively by applying Situation S1 to the gap on terms. Here, Diethelm and Goschler's research call [33], which explicitly points out this gap, is the decisive reason to explore the terms first. This exploration is a necessary preliminary step for the other subsequent studies. Only then can the subsequent studies approach the answer to the overarching research question more directly. Students' understanding also includes their misconceptions and conceptions, see Section 2.3.1. Therefore, a second research question was formulated regarding the relationship between students' terms and misconceptions and conceptions. Thus, the research questions of the Basis Study are:

(RQ1) What terms do novices use to describe code snippets in Java?

(RQ2) To what extent and in what way do the terms indicate programming language misconceptions or conceptions?

The second study, Application Study I, primarily addresses situation S3 as a specific teaching-learning situation. The overarching research question asks about emerging insights when the relationship between programming language and natural language is made explicit, see Section 1. Application Study I examines the effects of the didactic basis of the instructional videos in an experiment. One of the two didactic bases used is the language-sensitive teaching (LST) approach, where the relationship between programming language and natural language is made explicit. The observed effects are students' performance in writing code and their self-efficacy. The former, students' writing code, is itself one of the specific situations, namely situation S2. The latter, self-efficacy, is used as an additional effect because it is related to several constructs [178] and is a "legitimate outcome in itself" [196]. From this, the following research question was derived:

(RQ) How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers?

As a third study, the Application Study II directly addresses situation S5 by assessing the quality of teaching staff answers to students' queries about code. To assess quality, this study develops and evaluates a tool. Therefore, the basis of Application Study II is research goals instead of research questions. A rubric is used as the framework for the tool because it allows for assessment at multiple levels. The evaluation focuses on both internal and external evaluation, the latter including expert opinion on perceived support. For the latter, we focus on teaching assistants as part of the teaching staff and on pedagogical content knowledge competencies as familiar areas for experts. Therefore, the two research goals are:

(RG1) Development of a rubric that assess the quality of written answers to student queries about code.

(RG2) Evaluation of the rubric by (1) assessing the quality of answers and (2) obtaining expert opinion on how it can support structured assessment of TAs' PCK competencies.

Finally, we address situation S4, combining the assessment of students' prior programming knowledge with our focus on natural language. Here, the lack of prior experience does not mean that these novices do not have the thought structures necessary for programming. The latter was previously described as commonsense computing, which we identified as a longer language-related trend in computing education, see Section 2.2.2. As pointed out in Section 2.3.2, valid foundations and standardization of assessment play a minor role in CER. Therefore, this study combines both assessment with natural language-only tasks and the goal of a valid instrument. The resulting research question is:

(RQ) How successful is a commonsense computing test based on natural language as a programming aptitude test?

3. Methods

This thesis explores teaching-learning processes and the actors involved in them. In a broader sense, it is situated in empirical social research in the sense of general education research. Therefore, an approach from this line of research has also been chosen as the general research approach. This is that of mixed methods. In Section 3.1, the mixed methods approach is first explained in general terms and then described how it was specifically applied in this thesis. Then, the individual methods that were used in this thesis are presented in Section 3.2. The joint presentation is useful at this point because the methods have been used in several studies conducted. Finally, Section 3.3 is dedicated to the participants and the general context.

3.1. Mixed Methods as General Research Approach

This section first defines what is meant by the term *mixed methods*. Then, the reasons for choosing this research approach are discussed, followed by its applications in CER. Finally, the general research design of this thesis is presented and how the different studies build on and complement each other.

There are many definitions of what is meant by mixed methods [30]. A frequently cited definition comes from 2007 by Johnson et al. [88], who defined the term as follows:

> *"Mixed methods research is the type of research in which a researcher or team of researchers combines elements of qualitative and quantitative research approaches (e.g., use of qualitative and quantitative viewpoints, data collection, analysis, inference techniques) for the broad purposes of breadth and depth of understanding and corroboration." [88, p. 123]*

They derived their definition of mixed methods by composing 19 prior definitions of the term. The advantage of their definition is that by way of constructing through composing diverse perspectives were integrated. Which in turn yields building a broad understanding of the term mixed method. Moreover, the definition by Johnson et al. [88] does not focus on a single study and is therefore suitable for this thesis.

In this thesis, the mixed methods approach is used because the thesis aims to take advantage of this approach, i.e., both breadth and depth of understanding, as stated in the previous definition. The following additional reasons support the use of the mixed methods approach to answer the overarching research question: First, the explicit linking of programming language and natural language as a teaching and research approach has not yet occurred in CER, so there are no pre-existing theories on this topic to build upon. Consequently, a purely explanatory, quantitative approach to testing an existing theory is not appropriate here. Therefore, the first step in answering the overarching research questions is an exploratory, qualitative approach. However, this first step cannot be the last. Especially in education research, the practical application of research findings is very welcome. Therefore, the hypotheses and approaches derived from the first step should also be tested in explanatory, quantitative follow-up studies. This does not mean that these types of studies are the only follow-up studies. Teaching-learning situations involve multiple actors, the content itself, and the interactions between them. Therefore, the study of teaching-learning situations is a complex problem. The mixed methods approach "provides the most complete analysis of complex problems" [88, p. 23]. This work specifically aims to include multiple actors of teaching-learning situations. Consequently, the mixed methods approach is particularly well suited to this specific complex problem formulated in the overarching research question.

The mixed methods approach offers not only a comprehensive analysis, but also other advantages listed by Creswell and Clark [30] and summarized below. A mixed methods research approach provides every opportunity for the selection of research instruments. Thus, the research instrument can be freely chosen depending on whether it fits the context or not. In addition, the mixed methods approach provides the opportunity to compensate for the weaknesses of each approach by combining them. For example, qualitative research is limited to rather small sample sizes, which sometimes has a negative impact on generalization. Quantitative research, on the other hand, is less reflective of participants' perspectives and concepts. In addition to these two advantages, which are on the practical side, there are other advantages that go beyond the sum of their parts: There are research questions that cannot be answered by one approach alone, either quantitatively or qualitatively. These include the overarching research question posed in this thesis. With one approach alone, it is not possible to capture and examine different exemplary teaching-learning situations and the perspectives of the different actors in them. Moreover, the mixed-methods approach offers new insights that can only be gained by combining the individual findings. In other words, the answer to the research question consists of more than the sum of its parts. However, not all benefits are relevant in this case, such as helping researchers develop a broader range of skills. This is a welcome side benefit, but not directly relevant to answering the research question. In summary, providing a comprehensive analysis for questions like our overarching research question that cannot be answered by a single approach alone is the key benefit.

Before focusing on the specific methods used in this work, we will turn to how mixed methods were used in CER in general. Mixed methods are used to varying degrees in CER. Randolph et al. [157] examined a sample of 352 papers published between 2000 and 2005. They classified only 15 of the 144 studies with human participants as having used mixed methods. This represents approximately 10.4%. Following this period, Sheard et al. [168] focused on papers with reported research in the years 2005 to 2008. They used a broad definition of mixed methods, for examples, papers that focused on qualitative analysis and reported on the frequency of responses. In this way, they classified 42% of the papers as mixed. Later, Marguliuex et al. identified 64 of the total 197 papers reviewed between 2013 and 2017 as mixed methods [115]. This corresponds to approximately 32.5%. In contrast, Heckman et al. [66] categorized 351 empirical papers, all published in CER between 2014 and 2015. Here, only five of the 351 papers were categorized as papers using the mixed methods approach. Accordingly, the mixed methods approach is definitely used in CER, although it is not the most common approach. As far as the author is aware, there is no current review of research approaches used in CER studies.

Next, the specific research design of the mixed methods approach is presented. Our specific research approach is situated in the pragmatist worldview [128, 181]. Here the intention arises from the relevance of the practice. Thus, the consequences that research has for practice are the purposeful motivation. This worldview seems to us particularly appropriate for the focus on teaching-learning situations. Teaching-learning situations themselves occur in everyday practice. Therefore, research on teaching-learning situations is also relevant because of its implications for everyday practice.

The mixed methods approach used here is a sequential, exploratory design. What this means is explained below and is illustrated in Fig. 1.1 on page 6. As a reminder, this illustration has already been briefly presented in Sect. 1. Here, Fig. 1.1 is described in more detail. In general, a sequential, exploratory design involves three phases. In the first phase, qualitative data are collected to explore the general research context. The results of the first phase are used to develop new hypotheses or new instruments. The second, subsequent phase builds on the first. This phase involves parallel studies, that is, parallel development of study design and data collection. In the field of mixed methods, the term convergent studies is also often used to refer to parallel studies, e.g., [30]. Conducting parallel studies in the sense of overlapping data collection has the advantage of allowing efficient use of time. The third phase aims to provide an overall answer to the overarching research question. Here, the results of all studies are compared and synthesized. The comparison follows the principle of discussing the areas of compliance and deviation. In other words, the results are compared for similarities and differences according to the answer to the overarching research question. Phase three ends with an integrated interpretation as a synthesis based on the previous comparison. This sequence of the three phases is represented by the term sequential in sequential, exploratory design.

In the following, the four studies conducted are assigned to the phases of the research design used. The first phase consists of the Basis Study. The Basis Study is a qualitative-dominant, exploratory study which examines the terms novices used when describing code. The second phase here consists of three studies, namely Application Studies I to III. Application Study II collects qualitative data, while Application Studies I and III collect quantitative data. Application Studies I and III again differ in the way they use quantitative data. Application Study I conducts a randomized control trial, i.e., an experiment, that examines the effects of the didactic basis of instructional videos. In contrast, Application Study III quantitatively evaluates the NLCT as a developed instrument using an item response theory modeling analysis. In turn, the Application Study II also evaluates a developed instrument, namely a rubric. However, this evaluation is qualitative. The following third phase combines the findings of the previous phases. All four studies conducted deal with different teaching-learning situations and focus on different actors, as already described in Section 2.4. Therefore, the combined results give a comprehensive insight and provide a sound answer to the overarching research question.

3.2. Mixed Methods Used

In this section, three different methods are presented, namely qualitative content analysis (3.2.1), item response theory (3.2.2), and statistical analysis (3.2.3). All three methods were used in more than one of the total four studies. Therefore, this joint presentation at this point in the thesis avoids the repetitive presentation of the same methods and serves to improve readability in the following chapter. Thus, the following chapter 4 can focus on the actual study-specific methodology and especially on the findings themselves.

3.2.1. Qualitative Content Analysis

In this section, the method of *qualitative content analysis*, *QCA* for short, is presented. For this purpose, after an initial definition, the classification and differentiation from other methods will be discussed. Then, the general principles of this method are presented in more detail. Finally, the specific approach of QCA relevant to this thesis is presented.

In this thesis, the QCA as carried out by Mayring [118] is used. Other versions are discussed later when differentiating from other methods. For definition, qualitative content analysis is a text analysis technique characterized by its strong focus on systematicity and implemented through a strictly rule-based approach. In addition, QCA focuses on specific research questions rather than open-ended exploration, and it is economical because it can handle larger textual material.

Qualitative content analysis is in itself a method that belongs to the mixed methods. This is explicitly illustrated by the following quote, in which Mayring describes which parts of QCA belong to the qualitative and which to the quantitative methods: *"This is what qualitative content analysis does, combining qualitative and quantitative steps of text analysis [...]. The assignment of categories to text passages represents a qualitative interpretative (but rule guided) step; the analysis of frequencies of such assignments represent a quantitative step."* [118, p. 14]. This method thus contains the principles of the pragmatic mixed method described above. In our view, however, the qualitative element of the approach is decisive. In a quantitative content analysis, the text is not interpreted and no categories are abstracted, but the use of words is counted. In this way, no deeper meaning emerges. Our view on the importance of the qualitative aspect is in line with Mayring, who does not consider the mere counting of aspects as in quantitative content analysis to be sufficient [118]. However, we intend to explore this deeper meaning in the studies of this thesis, where the qualitative approach predominates. In conclusion, the QCA method is well suited for use in these studies.

In addition to Mayring's version, there is also the QCA version according to Kuckartz, which was first presented in 2012 [98] and has been further developed until today [99]. Similar to Mayring, Kuckartz's approach is also characterized by the importance of systematics. In contrast, Kuckartz's process of QCA does not follow a sequential process like Mayring's with fixed steps. Instead, the process has open phases that are passed through in a cycle. Accordingly, the character of QCA according to Kuckartz is much more open and inductive, but less focused on a specific research question. Accordingly, this QCA approach is somewhat closer to hermeneutic approaches to textual analysis. Hermeneutic methods are characterized by a deeper immersion in the texts, where are trade-off that this also involves a large investment of time. Therefore, these approaches are less suitable for large amounts of text.

Before focusing on principles, a brief note on the use of QCA in CER. Mayring's version of QCA has been used in several different research activities. These include, for example, analyzing curricula to build a model of pedagogical content knowledge [83], summarizing teachers' open-ended responses in a survey [193], or using this method to conduct a systematic literature review [58]. This gives a first impression of the range of possibilities for using QCA.

Next, the general principles of QCA are presented and what strengths and weaknesses result from them. Mayring summarizes the principles by calling QCA a "strongly rule based oriented step-wise procedure with a clear theoretical background [118, p. 71]. Moreover, the emphasis is on categories that can be either deductively assigned to the text or inductively derived from the text. These categories are considered together with their respective references to textual excerpts. The identification and referencing of categories in the text is referred to in the literature as coding. However, coding is not applied to the entire text, but to predefined segments. By tracking references, quantitative analysis of the distributions of category assignments is possible to further examine the text. For a rigorous methodology, pilot testing is absolutely necessary to ensure quality criteria. Another quality criterion is the consistency of coders both between different coders and between the same coder at an earlier and later time.
Mayring presents strengths and weaknesses together [118, p. 195f.], because they are the two sides of the same coin. Every strength is also a weakness, depending on the context, the research intent, and the scientific orientation of the researcher as to how the research should be conducted. Strengths include rigorous scientific standards guided by those of quantitative research and intersubjectivity. These two become weaknesses if one applies a purely qualitative interpretation with a subjective understanding of the text. According to this understanding, the strength of increasing objectivity through inter-coder agreement also becomes a weakness. The ability to analyze and reduce extensive textual material is a strength; unless the focus is on the deeper meaning of studies with little textual material. Likewise, reducing complexity through categorization as a strength is a weakness when the deeper meaning of the text is the focus of the analysis.

The step-by-step approach with fixed rules after the pilot test has been conducted is a weakness because it can limit understanding and prevents new aspects from being found in the textual materials. However, this very approach makes it comprehensible to other researchers and allows replication of the results.

Finally, the focus shifts more to the specific approach and steps of QCA. Previously, we introduced the principle of coding with the matching of text segments to categories. Here follows a more detailed description of what the segments are. Mayring called these segments "content-analytical units" [118, p. 64], which emphasizes that the analysis is based on and defined by these units. The context-analytical units include three distinct units: (1) The "coding unit" defines the smallest textual component to be coded, i.e., assigned to a category. This may be a part of a word, a word, an entire text document, or other (sub)parts. The coding unit thus represents the sensitivity of the analysis. (2) The "context unit", in contrast, defines the largest text component to be coded. (3) The "recording unit" represents which parts of the text can be coded with a given set of categories. The latter is important because multiple research questions, and thus different categories, can be applied to the same texts.
The variety of analytical possibilities that arise from corresponding definitions of content-analytical units is evident in the forms that QCA can take. All of the forms distinguished by Mayring are based, to varying degrees, on three basic goals to be achieved by QCA analysis: summarizing, explaining, and structuring by categorizing. As a reminder, category definition is one of the central principles of QCA, as mentioned earlier. Two forms are particularly appropriate for category definition, namely inductive category formation and deductive category assignment. In the former form, categories are developed from the material itself. In the latter, theories are used to construct categories whose manifestations are then identified in the material.

Here, QCA is used in both the Basis Study and the Application Study II. The related research questions are exploratory or descriptive in nature. Thus, they fit an inductive logic and inductive category formation is particularly appropriate for answering the research questions. Mayring formulates eight steps in how inductive category formation is carried out [118, p. 81–85]. Briefly, the focus of the steps are as follows: (1) research question, (2) definition of categories and context-analytical units, (3) coding, (4) pilot test with revision, (5) final coding, (6) formation of main categories, (7) agreement between coders and within coders, (8) quantitative results. If revision of categories or context-analytical units is needed, go back to step (2) and repeat all these steps again. For the goal of QCA as categorization, it is important that the main categories in step (6) emerge as an abstraction of the categories found. Therefore, step (6) and its quantification in step (8) are particularly important. Finally, step (7) is explained in more detail because of its importance of intersubjectivity as a strength of QCA. Agreement between different coders is also called inter-coder agreement, while agreement between the same coder but at an earlier or later time is called intra-coder agreement. For inductive category formation Mayring recommends a qualitative check and for deductive category formation the consideration of a statistical coefficient of agreement [118, p. 181].

3.2.2. Item Response Theory

Item response theory (IRT) has a long history in the science of measurement in various disciplines. CER is not exempt from this. Representative of many, only a few papers are listed here, namely [19, 192, 193]. IRT is a statistical modeling approach for estimating examinees' abilities based on their responses to test items [138]. Here, in the studies conducted, an examinee corresponds to a student participant. The responses to the test items are their data collected according to the study-specific tasks.
The general assumption of IRT is that there are concepts that are not directly observable, such as "intelligence", but the theory has no problem in assuming them as known. Such concepts are referred to as "underlying constructs" or "latent traits" [138]. In the context of this work, the constructs we focus on are, for example, self-efficacy and commonsense computing. IRT solves the problem of non-direct observability by deriving a measure based on responses to various test items. To solve these test items, examinees need competencies in the underlying construct. As the first defining feature of IRT, the modeling approach results in each IRT model containing the probability of each possible response for each test item. The data collected in the studies provide the data from which these probabilistic measures can be derived. In addition, the modeled abilities of the participants and the parameters of the test items, such as difficulty, are calculated separately. This separation between parameters is described as the "ultimate defining characteristic" of IRT [1, p. 8]. Because of this separation, IRT allows for a more detailed analysis of how good the accuracy of the items is. In addition, the difficulty of the test items and the modeled abilities of the participants can be compared on the same scale.

Based on these properties, IRT solves the following problems: First, as in classical testing, assign sub-points to test items and thus collect raw data in the form of ordinal data. Neither the distances between the sub-points nor the difficulty of the items are the same. Therefore, it is not possible to sum the raw data to obtain a test score. IRT provides a solution because the mathematical models allow us to convert raw data into metric scaled data. In this way, both the examinee and the difficulty of the item can be described on a metric scale. This allows the use of a variety of statistical tools, while tools for ordinal data are limited. Second, the modeled abilities of the participants are based on the responses to the test items. The test items can be added, deleted or replaced. However, comparability between participants is not lost [18]. This is in contrast to classical measurements, where participants' abilities are expressed by numerical counts. These classical measurements are based on classical test theory (CTT). In CTT, an examinee's observed score is a combination of the unobserved true score and some measurement error. However, this assumption is unfalsifiable [110]. In contrast, the assumptions of IRT are falsifiable. What these assumptions are is described below.

IRT is based on two prerequisites. First, all test items measure the same underlying construct, which is referred to as *unidimensionality*. For example, in Application Studies I and III, self-efficacy and commonsense computing, respectively, were used as the underlying construct. There are also multidimensional IRT models [18]. However, the focus on unidimensionality is more common and fits better with the studies conducted in this work. Therefore, the focus remains on unidimensionality. There are several ways to ensure unidimensionality. Here, we used principal factor analysis of standardized residuals (PCA) as a simpler method and confirmatory factor analysis (CFA) as a more sophisticated method. For the former, we refer to the textbook by Boone and Staver [21] and for the latter to the textbook by Hair et al. [62]. Briefly, PCA examines the patterns in the data and finds the component that explains most of the variance in the data. There are no set limits for interpreting PCA results, but commonly used rules of thumb based on the eigenvalue of the first component.

In contrast, CFA brings several established model fitting statistics as a more sophisticated method. Unlike PCA, these fit statistics have established bounds for interpretation rather than just rules of thumb. CFA starts from a previously specified model, in this case a unidimensional model, and tests whether the collected data fit this previously specified model. Performing CFA requires knowledge of both the approach and the associated software, as well as further steps to interpret the values. This makes ensuring unidimensionality more sophisticated, but also requires more effort.

Second, IRT requires *local independence* of individual items [43]. This means that the items of the test are statistically independent of each other when controlling for the same underlying construct. To meet this requirement, the items should not build on each other. Yen's [194] Q3 statistic is commonly used to check for local independence. However, there is no established rule of thumb, but a variety of arbitrary rules.

For adaptation to different situations, IRT provides different models. IRT distinguishes between models based on the number of their item parameters. Models with one parameter (1PL) calculate only the difficulty of the items. Models with two parameters (2PL) additionally compute item discrimination, i.e., a measure of the differential ability of an item. Ideally, high discrimination parameters are desirable to detect subtle differences in examinee ability. There are also models that take into account the guessing probability of the items, i.e., how likely it is that an examinee will guess the correct solution. In addition, IRT has several models based on how the items are scored. The simplest scoring is for dichotomous items, which distinguish between incorrect and correct. Examples of dichotomous items are multiple-choice items with a single open response. In contrast, polytomous items are scored on an ordinal scale. Examples of polytomous items are ratings of items on the Likert scale or items whose responses are rewarded with sub-points. For the combination of 1PL model and polytomous items, there is the Rating Scale Model [7] and the Partial Credit Model [117]. In the case of the 2PL models, these polytomous models generalize to the following models: the Generalized Partial Credit Model [130] and the Graded Response Model [161]. The decision of which model to choose can be based on both item fit analysis and information criteria. For the latter, the Akaike information criteria (AIC) and Bayesian information criteria (BIC) are most commonly used, as evidenced by their use in popular textbooks, e.g., [138].

IRT provides a variety of ways to analyze the fit of items and the test in general. These include: (1) Infit and Outfit statistics describe how well the item measures for examinees those person parameters that are close to or away from the item's difficulty level. (2) For polytomous items, thresholds should be ordered between sub-points. That is, higher ability should lead to higher scores. If this is not the case, then the item has a misfit. (3) Characteristics curves, where the examinee's ability is mapped to the probability of solving the item correctly (for dichtomous items) or achieving specific sub-points (for polytomous items). Simply said, all partial scores should have the highest probability for some range of examinee's ability. (4) The person item map, in which the calculated values for item difficulty and examinee ability are combined. Here one can examine whether the full range of the examinee's ability is covered and which parts of the ability are under- or over-represented in the test items. Full coverage and good representation is necessary for accurate measurement.

3.2.3. Statistical Analysis

This section on statistical analysis lists the methods used in the studies conducted. In this sense, it serves as preparation for the presentation of the method and findings in the immediately following chapter. A total of four methods of analysis are listed: (1) correction for multiple testing, (2) examination of group differences, (3) quantitative description of agreement between raters, and (4) correlation between variables.

First, the correction for testing multiple hypotheses is described. Multiple testing requires the adjustment of the significance level α. Otherwise, the probabilities of a type I error increase enormously. A type I error is also be called a false positive, since one falsely rejects the null hypothesis. The *Bonferroni correction* is a commonly used method to reduce the increase of type I errors. It was introduced by Dunn [40] but named after Bonferroni. Compared to other correction methods, it is the simplest and at the same time the most conservative method [87]. This correction works as follows: Suppose there are m hypotheses to be tested at the significance level α for the same data set. Then the adjusted significance level α^* is calculated by $\alpha^* = \alpha/m$. For example, if you set α=0.05, the desired global significance level of 0.05 is maintained. In this thesis, several hypotheses were tested in Application Study I and therefore the Bonferroni correction was applied.

Second, the statistical analysis for the comparisons of the groups is presented. In general, there are a variety of tests for all combinations of the scale level of the data. Thus, the selection of the appropriate tests depends on the scale level of the data. In this thesis, both metric and ordinal data were collected in Application Study I. Therefore, only the methods for comparing groups with respect to these two scale levels are presented below. For metric scaled data, the calculation of the mean is possible. Therefore, the two-sample t-test is the method of choice for comparing groups. However, for ordinally scaled data, only the calculation of the median is possible. Therefore, the groups are compared using the Mann-Whitney U test. Next, both tests are separately described along with their requirements.

Both tests have their own requirements, which are listed below. For a two-sample t-test, these are the following, listed by Herzog et al. [76]: (i) The measurements of the groups being compared must be independent; (ii) The data must have a normal distribution; (iii) The data must not contain outliers; (iv) The variances in each group should be (approximately) equal, i.e., homoscedasticity. The first requirement, (i), can be easily ensured by the study design, e.g., randomized groups in an experiment, such as Application Study I is. For (ii), a check with the Kolmogorov-Smirnov test or the Shapiro-Wilk test is possible. The latter is preferred here because of its greater statistical power [159]. For (iii), the absence of outliers is qualitatively tested using boxplots; For (iv), Levene's test is appropriate because this test actually tests for homoscedasticity [55].

The Mann-Whitney-U test compares only the entire distributions - i.e. location and shape, and not only the medians of the two groups. If the distributions between both groups are the same and have only been shifted along the x-axis, the Mann-Whitney U test actually compares the medians of both distributions [36, 63]. Accordingly, a prerequisite for comparing the differences in the medians using the Mann-Whitney U test is that the distributions are the same. The Kolmogorov-Smirnov test is used for verification.

The final aspect to consider when comparing group differences is effect sizes. Research is now moving away from simply reporting p-values. Instead, effect sizes are of additional importance. Research in computing education is not exempt from this general trend, although few effect sizes have been reported so far [163]. Effect sizes are a way to better distinguish between statistically significant and significant. The former describes that the outcome has a low probability of being random, while the latter describes that the outcome actually makes a difference. The choice of effect sizes, like the choice of tests for group differences, is largely determined by the available scale levels.

As a reminder, group comparisons were conducted in Application Study I of this thesis. In this study, the independent variable was nominal, while the two dependent variables were a metric and an ordinal variable. For the effect from a nominal variable to a metric variable, Freeman's θ [52] was the effect size of choice. The effect from a nominal variable to an ordinal variable was measured with Cohen's d [29]. For better estimation of effect sizes, the 95% confidence interval was always also reported.

The third method presented is the quantitative study of inter-rater agreement (IRA). A variety of different tests were used in the past and are still used today. These tests differed depending on the number of raters (two or more) and the scale level of the data being compared. However, because of the different tests, the results of IRA from different studies are difficult to compare. In addition, some of these tests are susceptible to missing data. For example, Cohen's κ itself cannot handle missing data, but requires a variant [156]; the same is true for Kendall's τ [50]. One way out is to use Krippendorff's α [96]. This metric for evaluating IRA is to be used regardless of the number of testers, scale level, and missing data. In this way, overall comparability between studies is greatly improved or made possible in the first place. For the evaluation, Krippendorff recommended relying only on variables with $\alpha \geq 0.800$, and variables with $0.667 \leq \alpha \leq 0.800$ should be used only for tentative conclusions [96, p. 272]. In this thesis, one of the three research questions of the Application Study III explicitly asks about the performance of the developed test in relation to IRA. Consequently, Krippendorff's α was examined there with n=15 raters.

Finally, the focus shifts to correlation coefficients. If the variables under study are metric, then two different coefficients come into question. These are the Bravais-Pearson correlation coefficient [77] - also known as Pearson's r - and the Spearman's rank order coefficient - also known as Spearman's ρ. If the variables are normally distributed, we can use the Bravais-Pearson correlation coefficient; otherwise, Spearman's rank order coefficient is appropriate. In the case of perfect linear correlation, the absolute value of r is equal to 1, while a value of 0 means that there is no linear relationship between the variables under study. Similarly, an absolute value of ρ of 1 means a perfect monotonic increasing relationship, 0 means no monotonic relationship, and -1 means a perfect decreasing monotonic relationship. Scatterplots are appropriate prior to statistical calculations to examine the relationship between these variables. These scatterplots allow the study of outliers or patterns of interest. In this thesis, an analysis of correlation of coefficients was used to investigate the performance of the developed test as a predictive factor in the Application Study III.

3.3. Participants and Their Context

First, the Table 3.1 provides an overview of the participants in the studies and the associated semester. With the exception of the CER experts, all participants were students, as the student teaching assistants were also students themselves. All students were enrolled at the Technical University of Darmstadt in Germany at that time. The CS1 students form the main group of participants. Therefore, the context of these participants is presented first.

The following description of CS1 courses applies to all courses in which CS1 student were recruited as participants. Learning to program in Java is the central theme of the 14-week CS1 course. The instructor was assisted by 30 student teaching assistants (TAs). The author supported the course with TA training and an exam preparation course. Topics include, among others, the basics of object orientation, static and dynamic types, error handling, and generics. Students received points for 14 individual homework assignments, with half of the total points required for admission to the exam. There was an optional programming group project prior to the exam. The exam was written and proctored online. The only exception to this course design was in the Fall 2020/21

Table 3.1.: Overview of the participants in the studies conducted

Study	**Participants**	**Semester**	**Sample size**
Basis Study	CS1 students	Fall 2020/21	n=123

Application Study I	CS1 students	Fall 2021/22
		and 2022/23
			n=133
			n=428

Application Study II	Student teaching assistants
	and CER experts
		Fall 2022/23	n=30
			n=13

Application Study III	CS1 students	Fall 2022/23	n=681

semester. Due to the Corona pandemic, the course in that fall semester consisted of 12 weeks.

Experience has shown that in the CS1 courses in question, not all students associated with the course actively participate in the course. Therefore, the number of students in the corresponding administrative course, i.e., the Moodle course, was not used. Rather, we were interested in the active population. The active population of the course was calculated based on the number of students submitting homework assignments at the time of the corresponding study. Thus, the study response rates reported were the proportion of participants in the active population in the associated course. These varied between 16.9% for the first cohort of Application Study I and 63.3% for Application Study III.

We now turn to the participants in the Application Study II. All student teaching assistants had previously participated in a two-day TA training workshop. This workshop was conducted by the author and was specifically tailored for TAs supporting the CS1 course. The content of the workshop included (mis)conceptions of typical CS1 topics such as variables, loops, and objects. Therefore, a minimum level of competence in TA's supporting skills could be assumed. In contrast, CER experts as other participants could be assumed to have a high level of competence in computing education. The experts were invited individually. Most of them had previous experience as authors of publications on pedagogical content knowledge in computing education and/or with training TAs. In contrast to the CS1 students and TAs, the experts had a broader background. The experts were from four countries, three in Europe and one in North America. The CS1 students and TAs instead were all from the same university.

Next, the giving of informed consent and related aspects are described. Participation was optional. At the beginning of the study, participants were given a description of the subject, procedure, duration, and benefits. Participants indicated whether we could use their data. The study complies with the ACM Publications Policy on Research Involving Human Participants and Subjects.

Demographic data varied from study to study. In the fall 2020/21 and 2021/22 semesters, we did not collect precise demographic data because it was not permitted by the local ethics committee. However, descriptions of participant demographics are proposed in CER [66]. To overcome these differences, the following approach was taken: In the fall semester of 2022/23, we added an additional optional survey to the Application Studies I and III that asked for demographic data. In contrast, we did not ask for demographic data in the Application Study II due to small sample sizes of both student teaching assistants and experts. Under these circumstances, approval was not required according to ethics committee guidelines. The demographic survey of the Application Studies I and III had to be separated from study participation. In both cases, more students reported their demographics than participated in the studies. Because of the separation, the reported distributions may differ from those of the actual studies. Both demographics showed a fairly similar distribution. The median age for both studies was 20 years, between 22% and 28% identified as female, and about 1% as diverse. A migration background was affirmed by between 35% and 37% of the students, and between 50% and 52% had CS as a school subject.

Similar to collecting demographics, compensation varied between studies. Participants in the fall 2020/21 and 2021/22 semesters each had a chance to receive one of 20 vouchers as an incentive. Each voucher contained a small amount of money. This compensation practice resulted in lower response rates than expected. Therefore, the practice was changed in subsequent semesters. The participants of the fall semester 2022/23 received a small number of bonus points for the final course exam. Here the points were awarded for completeness and not for the correctness of the response. Apart from the CS1 students, the student teaching assistants were credited with compensation in the amount of one working hour.

Overall, the context of the participants may not be representative of other situations. This is due to the specific CS1 course with the associated students as the main group of participants. For these reasons, only preliminary results can be used for generalization to other universities, institutions, and participants.

4. Study-Specific Findings

This chapter contains both study-specific findings (Sections 4.1 to 4.3) and a general summary (Section 4.5) of the findings. The preliminary work for the study-specific findings was done in the previous chapters on literature and methodology. There, as a conclusion from the previous literature (Section 2.4), the subordinate research questions were formulated. Following this path, both cross-study and study-specific methods (Section 3) were introduced based on the research questions. This chapter now follows this path. For ease of reading, the research questions are reiterated in each of the following study-specific sections. The general summary is the segue to answering the overarching research question, which is the first focus of the next chapter.

The following sections list the research questions or goals and summarize the methodology and findings. Further discussion is summarized for all studies together in the following section, see Section 5.

4.1. Basis Study: Terms Novices Use

As a reminder, the basis study aims to explore the relationship between programming language and natural language through the focus on novices' terms. Both the language and the terms used by novices are areas with more unanswered questions than answered ones [33]. The studied research questions were:

(RQ1) What terms do novices use to describe code snippets in Java?

(RQ2) To what extent and in what way do the terms indicate programming language misconceptions or conceptions?

By way of introduction, the focus of the study is on answering RQ1, while RQ2 is answered more shallowed.

The data collected consisted of over 1800 free text responses from n=123 CS1 students from the fall 2020/21 semester. Here, participants were asked to explain what the associated code snippet does. In total, descriptions for fourteen different code snippets were analyzed. The code snippets covered several fundamental content areas of introductory programming, namely variables, input and output, control structures such as conditional statements and loops, functions, and object-oriented programming basics such as objects and classes [183]. They had a length of one to three lines. In some cases, the lines of code were preceded by an additional comment line to introduce the data type of the occurring reference variables. For example, the code line `a = b;` was preceded by the comment `// a and b are of type int`.

For the analysis of RQ1, qualitative content analysis (QCA), described above, was used to answer RQ1, and for RQ2, qualitative analysis of individual responses was used. Briefly, QCA was applied such that responses were first divided by category-for example, the left-hand side of the assignment-and then converted to the associated term. In addition, frequency tables, total number of unique terms, bar graphs of term distributions, and n-grams were used for further analysis.

For RQ1, the terms typically used by novices and the diversity among the terms are examined. The top ten terms include names, technical terms, and natural language with mostly compound terms. For names, the novices used “a” for variables and “Armin” for strings in their descriptions, as they were also used in the code snippets. Aside from the names, most of the terms come from technical language rather than natural language. This means that, by and large, participants are able to use the appropriate technical language in the appropriate places. The technical language, in turn, differs from the keywords and syntax of the programming language. For example, the term “loop” from the technical language corresponds to the keywords `for` and `while` in Java. However, among the ten most frequent terms, only one keyword appears, `int`. The terms typically used include four common verbs, namely “assign,” “execute,” “output,” and “create.” Each code snippet contains exactly one of these verbs among the ten most frequent terms. However, the grouping of the verb terms does not match the grouping that results from the content areas as the traditional grouping. Moreover, only the terms for three content areas, namely “variable,” “output,” and “object,” appear among the first ten terms. The content areas also differ in their similarity of terms. The similarity of terms between code snippets of the same content area is low for variables and loops and high for input/output and objects. As last aspect for the terms typically used, the

findings on n-grams are presented. The 3-grams and 5-grams partially share terms and the order of terms. The same is true for 3-grams and the top ten terms. All terms of 3-grams, except for articles such as “a” and “the”, are part of the top ten terms.
In terms of term diversity, the term distributions are considered. The distribution of code snippets consists of the term frequency of each term of the associated code snippet. The distributions as a whole are characterized by a high degree of homogeneity. They resemble an exponential distribution that has a high peak at the beginning and then approaches zero. Thus, most terms were used at most by a handful of novices. However, the term distributions differ with respect to the total number of terms, which ranges from 40 to 168. The conclusion from examining the distributions is that the top ten terms provide good coverage of all terms. Thus, they provide a sufficient answer to RQ1.

In RQ2, the focus is on exemplary terms and code snippets. First, the examples include terms for variable swapping and the combination of referencing and aliasing. For this, the specific code description and the included terms are analyzed. For example, the term “Sarah” stands for the entire object, which corresponds to an “identity/attribute confusion” [80]. As another example, terms such as “same value” in the context of swapping variables directly indicate misconceptions. While other terms like “swap” are context-dependent and do not by themselves indicate whether the novice has misconception.
Second, terms novices use to describe conditional statements are considered. The corresponding code snippet consists of a conditional statement block introduced via `if` and a subsequent statement block without condition. On the one hand, 34 of the 123 participants used terms for the subsequent statement block grouped as *counterpart* as a misconception. On the other hand, there were three different correct conceptions. First, terms that conveyed that novices considered the subsequent statement block as *separate entity*. Second, terms that clarified that the subsequent statement block is executed *later in time*. Finally, novices use terms that describe *skipping* the conditional statement block.

In summary, there is a homogeneity of terms. However, at the individual level, there is a wide range both in the terms themselves and in the number of terms used. Moreover, the analysis of terms can be a possible heuristic for the discovery of programming language misconceptions and conceptions. But they do not necessarily lead to this goal. Therefore, practitioners - such as teachers and instructors - can use the task of describing code snippets as a diagnostic tool. However, they must be aware that the context used and the interaction of the terms used are equally important. The terms provide clues to students’ thinking, but they are not clear indicators. The most important contribution of this study are the listings of the terms used and their categorization, as described previously. For a comprehensive listings of the terms, the reader is referred to Section 7. The reason that this is the contribution is the following: To the best of the authors' knowledge, this study is the first answer to the open questions of Diethelm and Goschler [33] concerning terms and their use by novices in their natural language in postsecondary education.

4.2. Application Study I: Context of Instruction Videos

Application Study I examined the effects of the didactic basis of the instructional videos. For this experiment, one of the two didactic bases used was the language-sensitive teaching approach, in which the relationship between programming language and natural language was made explicit. As reminder, effects on both writing code and self-efficacy were examined. The former is an apparent criterion in introductory programming an apparent effect, whereas the latter was used as additional but relevant criterion. From this, the following research question (RQ) was derived with its four hypotheses, H1 through H4. The abbreviation LST stands for *language sensitive teaching*. It was used in the publication to describe the approach in which the relationship between programming language and natural language is explicitly thought of together. The abbreviation WE in turn stands for *worked examples*, a common approach in CER [129]. The hypotheses were based on considering theories of learning development [189] and on transferring the researched effectiveness from mathematics didactics [153] to our context. The detailed results can be found in the publication listed in chapter 8:

(RQ) How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers?

H1 In general, i.e. for all novices regardless of their basic skills, the didactic basis has no effect on the CW performance.

H2 Instructional videos based on LST lead to better CW performances than those based on worked examples for novices with medium basic skills.

H3 Novices watching an LST instructional video have higher self-efficacy scores than when watching a video with worked examples.

H4 The effect stated in H3 is especially true for novices with medium basic skills.

For the research design, the experiment included four phases. First, basic skill was measured as a control variable through a self-developed code completion task. Second, participants were randomly assigned to one of three groups: Video based on LST, Video based on WE, or no video as the control group. Third and fourth, self-efficacy and code writing performance were measured. Self-efficacy was measured using a modified version of an already established test for measuring self-efficacy, namely the MSLQ [147]. For the subsequent code writing task, participants worked on a template in their regular IDE.

In terms of methodology, both for the measurement of the basic skill and the self-efficacy an item response theory approach was used. For the group differences, statistical analysis with Bonferroni correction were used to compare the groups. There, each test was selected according to whether the median of the code-writing performance or the mean of the self-efficacy scores were compared. The background of these tests, their requirements, their analysis and effect sizes were described previously, see Section 3.2.3. Participants were from two cohorts of CS1 students, with n=133 for the first cohort and n=428 for the second cohort.

Next, the answers to the hypotheses on code writing, i.e., H1 and H2, are presented. Hypothesis H1 was confirmed in both cohorts. In the statistical tests, this was characterized by the fact that the differences between the medians were not significant. As a reminder, because of the Bonferroni correction, the corresponding p values must be less than .0083 at a significance level of $\alpha = .05$. The p-values considered were far above this, with values of .188 and .322 for the comparison of the two didactic bases. The result is that didactic base has no effect on code writing performance among all novices. In contrast to H1, H2 could only be confirmed for cohort 1, while this could not be replicated for cohort 2. Thus, in response to the research question, it cannot be confirmed that the didactic basis affects code writing performance at all. Below, possible reasons are briefly listed why the results for H2 could not be replicated in the second cohort.

In Phase I, the code completion task served to measure students' basic skills by identifying those students who were in their zone of proximal development (ZPD) according to Vygotsky. It turned out that determining their ZPD required more fine-tuning. In addition, the two cohorts differed in their percentage of compilation errors, 4% (cohort 1) versus 21% (cohort 2). This affected the medians of the distributions, as compilation errors resulted in the lowest score of 1 out of 5 for code writing performance. Finally, the measurement of basic novice skills is based on Sindre's [174] findings on the correlation between completing and writing code. However, the results of this study do not support this correlation. Therefore, the determination of novices with medium basic skills may not be accurate and affects the testing of hypothesis H2.

As the results for H3 and H4 show, none of the hypotheses regarding the effects on students' self-efficacy could be confirmed. In the statistical tests, this was indicated by the fact that the differences between the means were not significant. The p-values, which ranged from .412 to .685, were well above the cutoff for a significant difference between the two didactic bases.

Possible reasons for this lie in Phase I, which was designed to determine the basic skills of novices. Self-efficacy and performance influence each other [105]. Performance in Phase I is not exempt and therefore influences participants' self-efficacy. In addition, participants spend a lot of time on the task in Phase I, which also negatively affects students' self-efficacy [57]. Thus, due to the strong effect in Phase I, the videos had no effect on self-efficacy.

In summary, the didactic bases chosen - language-sensitive teaching and worked examples - have no impact on novice code writing performance and self-efficacy.[1] The basis of language-sensitive teaching included the explicit connection of programming language and natural language mentioned in the overarching research question.

4.3. Application Study II: Context of Teaching Staff

This study presents a rubric as a developed tool and its practical contribution to the setting of computing education. Therefore, it is a tool paper which uses research goals instead of research questions. The research goals are:

RG1 Development of a rubric that assess the quality of written answers to student queries about code.

RG2 Evaluation of the rubric by (1) assessing the quality of answers and (2) obtaining expert opinion on how it can support structured assessment of TAs' PCK competencies.

[1]In CER, the publication of null and negative results in the call for papers of the relevant conferences is explicitly welcomed.

For RG1, the development started with theoretical framework. Here, the theoretical framework was derived by applying the prior theoretical models of GPK and PCK to the context of the rubric. As context, written answers to student queries related to programming task were in the focus. These queries were presented in vignettes, which are short, self-contained scenes depicting a realistic, pedagogical situation. The rubric's assessment could only be based on the written answers. Therefore, the inner processes were not part of the rubric, for example inner thoughts, reflections, and consciousness of the written answer's author. Then, the categories of the rubric were composed. Leading for the selection of the rubric's categories were the model of König et al. [100] and the model of Shulman [169]. From the former, the concept of structure was used to develop the categories of coherence and meta-level explanations. From the latter, the categories of illustrations, analogies, and examples were adopted albeit with different names. Additionally, two categories which share a language focus as well as the category addressing completeness were integrated.
Finally, all derived categories were formulated with competency descriptions on three different levels. Benefit of level descriptions are that they allow qualitative feedback on how to improve and what that might look like. This idea is based on the Hattie's [64] model of feedback. If the category's aspect is missing the written answer, Level 1 is assess as the lowest level. Misleading or implicit uses belong to Level 2. If all uses of the category's aspect are adequate, they represent Level 3 as the highest level.

As result, the rubric consisted of two basic categories (B) and six additional categories (A), each with three levels (L). In contrast to the basic categories, the additional categories are not necessary in the sense of an absolute minimum. The general structure of the category was that the following sentence is ended by each category separately. The particular sentence was: *The author demonstrates competence in formulating a written answer of adequate quality by [. . .]*. The two basic categories listed this by: formulating a coherent answer (B-I) and formulating a complete answer (B-II). The six additional categories listed this by: integrating meta-level explanations (A-I), integrating multiple representations (A-II), including concrete examples (A-III), using metaphors (A-IV), incorporating the student's language (A-V), and linking the language to the programming language (A-VI). For illustration purposes, one of the categories of the rubric is shown with all levels.

> **A-III** *[...] including* ***concrete examples****, i.e., an example whose terms are assigned concrete values, such as 5 for a variable. In contrast, an example is not concrete if the example uses only terms at an abstract level without associating those terms with concrete values.*

L1 The answer does not contain concrete examples.
L2 The answer contains at least one specific example. Even so, it is unrelated to the general concept before, after, or parallel to it.
L3 The answer contains at least one concrete example that is linked to the general concept. The example could be realized before, after or in parallel with the general concept.

This clearly shows how each category is constructed. First, the category is numbered and named. This corresponds here to the number A- III and the bold "concrete examples" as the name of the category. Each category name is followed by a description of what is meant by the associated category. Then all three level descriptions are listed, starting with L1 as an abbreviation for Level 1 as the lowest category and so on.

Finally, for RG1, the context of the use of the rubric is that of professional development. Here we include anyone who supports students in postsecondary education, including instructors, teachers, and teaching assistants. The purpose of the rubric is to base the feedback that supporters receive on the assessment of the rubric. This will provide a more solid foundation for feedback to supporters. This context does not preclude other uses, although they have not been the primary context.

The response to RG2 included both an internal and external evaluation. For the internal evaluation, 85 written answers from n=30 student teaching assistants (TAs) were analyzed using the rubric. All written answers were based on three vignettes on variable swapping, conditional statements, and loops. Each vignette contained a student's query or problem based on a code snippet or programming task.
For the result, the percentages of levels for each rubric category were examined. The percentage of Level 1 ratings in the basic categories was low, 13% and 16%, respectively. However, the percentages are too high for TAs who have completed a two-day training course. The percentages are also too high for TAs to be able to adequately support CS1 students. There were also four additional categories in which more than half of the answers were Level 1. In contrast, about one-third of the answers include examples and/or address the student's language at the highest level.

For the external evaluation, n=13 experts were interviewed one-on-one. Each interview included a written answer to be assessed. During the interviews, the experts were asked to rate the PCK of the TA as an author using the think-aloud method. Subsequently, the experts re-evaluated the written answer using the rubric. Follow-up questions included (1) general experiences with the rubric, (2) how the rubric supported

them, and (3) possible areas of application. Interviews were analyzed using deductive-inductive qualitative content analysis. Predefined themes were concrete positive and negative aspects as well as areas of application. Inter-coder agreement was examined qualitatively using n=2 raters.
As result, the most frequently mentioned positive aspects in the interviews were the analytical breakdown of the construct PCK and the support provided by the rubric for assessments. The most frequently mentioned negative aspects were the need to improve layout and features, and that PCK categories were missing. However, of the total 27 categories, 22 were mentioned by only one or two experts. Thus, most of them were not apparent or relevant to most experts. For the areas of application, experts addressed four dimensions: *activities* for which the rubric is used (e.g., reflection, planning), *context* in which the rubric is used (e.g., lesson observations), *addressees* for whom the rubric is used, and *discipline* in which the rubric is used, such as mathematics.

Overall, the development of the rubric has worked well. However, despite our efforts and multiple iterations, it is difficult to find clear and sound wording. Therefore, we had to fine-tune the descriptions in our rubric after the expert interviews. In addition, the rubric could not cover all aspects of quality. However, we believe that the rubric is a valuable basic tool that can be supplemented by additional categories. It was also positive that the expert interviews showed that the context of the rubric is even broader than assumed. This includes, for example, support for planning and reflection.

4.4. Application Study III: Context of Programming Aptitude Tests

The goal of this study is to develop and analyze a programming aptitude test that measures this construct using natural language tasks. We call this test *Natural Language Computing Test (NLCT)*. The NLCT is examined in terms of its accuracy, measurement precision, and ability to predict student success. Therefore, the following research question is fleshed out by three subordinate questions:

(RQ) How successful is a commonsense computing test based on natural language as a programming aptitude test?

How does the NLCT perform in terms of …

(RQ1) …*evaluation based on inter-rater agreement?*

(RQ2) ...*evaluation based on item response theory?*

(RQ3) ...*predictive factor for student success in CS1?*

First, the NLCT is outlined with its characteristics as a test, its development, and its items. The NLCT aims to measure the extent to which novice programmers are proficient in commonsense computing expressed in natural language. In this case, commonsense computing consists of following, formulating, and abstracting complex procedures, logical reasoning, and programming simulation through natural language. Thus, it requires reasoning in *procedural* programming expressed in natural language. The target group of the test are novices in CS1 courses based on a procedural programming language before or during the first week of the course. The NLCT is a performance test consisting of six items. The items are single-choice items with two possible answers, short-answer items, ordering items, and matching items. The estimated completion time is 40 minutes.
The NLCT was developed iteratively in four pilot phases and began with nine items. The items and coding manual were tested both qualitatively and quantitatively. The qualitative evaluations consisted of think-aloud interviews with students with no programming experience and expert review. The quantitative evaluations consisted of both students with no programming experience and experienced CS students who participated in solving the NLCT items. Due to their simplicity, three items were eliminated.
The six items cover all aspects of commonsense computing as previously defined. Two items, L2 and L6, use as context symbols such as circles and squares with dots in them. L2 requires to follow a complex procedure that modifies an array of six symbols. L6 implements a natural language variant of a Parson's puzzle with seven lines of code and two distractors. Items L3 and L5 use a different context, that of a robot in a small two-dimensional graphical world inspired by Karel the robot [16]. In L3, participants must follow a complex procedure of correctly placing six coins on the 4x4 square according to the given procedure. L5, in turn, is a specific code completion task, namely a task titled "Skeleton Code" [106] or "Fill in blanks" [174]. Item L1 deals with logical reasoning and Boolean expressions. Here we used the sandwich task [71, 186]. In this task, a good sandwich must follow three logical rules and students must decide and explain whether certain sandwiches are good or not. Finally, students must trace a code using a loop with multiple integer variables and abstract the purpose in item L4. Thus, L4 is the natural language version of the "Explain in plain English" [108] task.

Second, the focus shifts to the research questions. In general, the study included 684 participants for RQ1 and RQ2, but only 681 participants for RQ3. The difference came from the fact that three students did not provide data on homework for student success.

All three research questions use different methods. Therefore, both the study-specific methods and the associated findings are presented together, but separately for each research question.

For RQ1, the Krippendorff coefficient α was used to examine inter-rater agreement (IRA) because of its previously described advantages over other coefficients, see Section 3.2.3. For accurate assessment, IRA was used for each item as well as for the entire test. Due to the expected large sample size, a subset of 50 participants was scored. The author was accompanied by 14 student teaching assistants (TAs) as raters.
As result, only one of the items, namely L1, did not meet benchmark requirements of being at least 0.800 [96]. Examination of the TAs' scoring led to the fact, that four of them scored more severe than the coding manual stated. After elimination of these raters, also item L1 meet the requirements. Thus, the NLCT provides sufficient IRA results. Moreover, the good IRA results led to good reliability of the test, as the scoring is (almost) independent of raters.

For RQ2, the NLCT was evaluated based on item response theory (IRT). Due to the nature of IRT, the presentation of results follows the sequence of methodological IRT steps performed. The first step was to ensure the two requirements of the IRT approach: Unidimensionality of the construct to be measured and local independence. For unidimensionality, confirmatory factor analysis (CFA) was performed with the associated fit tests and factor loadings analysis. All four fit tests showed good model fit for an unidimensional model, ensuring unidimensionality. The factor loadings were also fine, with the exception of item L3. Participants had difficulty using the questionnaire software to answer the task posed in this item. Therefore, item L3 was dropped from the test to ensure high measurement accuracy. For local independence, Yen's Q3 scores were also all fine.
In a second step, all items were analyzed to determine whether they exhibited any misfit and required adjustments. Here, minor adjustments were made to the coding manual. For example, for items L2 and L4, two categories of points were combined. For item L6, the assignment of errors to points received was adjusted for this item because item L6 was relatively easy. After these adjustments, all items provided a good fit. Good fit here means that the better a student's ability, the more points they receive for the items. Furthermore, all (partial) points that can be achieved in the task should also be represented accordingly in the data. The third step was to select the General Partial Credit Model (GPCM) as a suitable model. The parameters of this model allowed quantifying both the difficulty and discrimination of each item. In a final step, the student scores on the NLCT resulting from the IRT analysis were linked to the difficulty

of the items. The result is that the NLCT is a little skewed towards easier items, with no undesirable floor effect but a ceiling effect. In general, the NLCT measures accurately.

For RQ3, correlation analysis was used to measure the value of NLCT as a predictive factor for answering RQ3. Here, student success in CS1 was defined as the percentage on homework assignments. The NLCT value was the test score calculated by IRT analysis. Since all variables were metric, Pearson's r was the means of choice to measure correlation. In addition, the scatterplot between the two variables and R^2 as the explained variance were examined. The Pearson's r correlation was 0.340 with [0.272, 0.405] as 95% confidence interval. The explained variance R^2 was 0.116. These values are lower than those reported in similar tests, which had R^2 values ranging from 15.57% [175] to 25% [185] for final exam scores. Thus, the suitability of the NLCT as a predictive factor was limited.

Overall, the NLCT performs well in terms of inter-rater agreement and measurement accuracy based on IRT analysis. However, it performs poorly on its ability to predict learning success in CS1. The NLCT would improve if there were more items for higher ability students. We can conclude that the NLCT does indeed accurately measure its construct, commonsense computing, but still has room for improvement.

4.5. Summary

Four studies were conducted in this work, divided into the Basis Study on one side and the Application Studies I to III on the other. As a reminder, the Basis Study approached the overarching research question in an exploratory manner by examining the terms novices use. The Application Studies in turn, approached the overarching research question by examining the relationship in specific teaching-learning situations.
The Basis Study revealed homogeneity of terms, although at the individual level there is a wide range in both the terms themselves and the number of terms used. Its main contribution was the listing of the terms used and their categorization. In addition, the analysis of the terms showed that although they provide clues to students' thinking, they are not clear indicators.
In Application Study I, a randomized controlled trial was conducted to investigate the effects in the teaching-learning situation when novices watch instructional videos. The result is that the selected didactic bases - language-sensitive teaching and worked examples - have no effect on novice code writing performance and self-efficacy. The basis of language-sensitive teaching included the explicit connection between programming

language and natural language mentioned in the overarching research question.
Application Studies II and III both addressed the topic of assessment to answer the overarching research question. In Application Study II, a rubric was developed for the quality of teaching staff's responses to student code questions. In doing so, some categories of the rubric explicitly related to the language focus of this work. The development of the rubric worked well, although the evaluation revealed that fine-tuning was needed and not all conceivable aspects of quality were covered. Nonetheless, the evaluation showed that the rubric is a valuable tool for educating teaching staff.
In Application Study III, the Natural Language Computing Test (NLCT) was developed and validated. The NLCT accurately measures commonsense computing and does not require knowledge of programming languages since all tasks are based on natural language only. Accuracy was examined using inter-rater agreement and item response theory analysis. However, the NLCT performs poorly in terms of its ability to predict learning success in CS1.
As a reminder, the publication of null and negative results in the call for papers of the relevant conferences is explicitly welcomed in CER. Thus, even the results of Application Study I and the poor performance of the NLCT are relevant findings.

5. General Discussion

Based on the study findings presented in the previous chapter, this discussion section interprets them in general, abstract terms. First, in Section 5.1, the main findings are summarized to answer the overarching research question. Section 5.2 then examines the significance of the findings and the strengths of the thesis in general. Significance also includes how the thesis contributes to the existing body of knowledge. Limitations and weaknesses that threaten the validity of the thesis and studies are acknowledged and discussed in Section 5.3. Finally, implications for teaching and research are presented as possible practical applications.

5.1. Answer to the Overarching Research Question

As a reminder, the overarching research question is *"What new insights emerge from exemplary teaching-learning situations in introductory programming in postsecondary education when the relationship between programming language and natural language is made explicit?"*

The insights from the four studies conducted differ with regard to this research question, but show a consistent overall picture. The differences are the different directions in which the insights point. To this end, the insights of the four studies are first presented. Then it is described how the insights do not contradict each other.

In the Basis Study, the relationship between programming language and natural language was made explicit by diagnosing the terms used by students. *The resulting insight is that the way the relationship is made explicit through terms is fruitful for conducting computing-specific research. Another insight is that the exploration of terms is itself a fruitful area.* Moreover, terms provide heuristics for identifying correct conceptions and misconceptions. Looking ahead, explicit thematization of terms is likely to be an effective support for practitioners in their teaching.

In Application Study I, the relationship under consideration is made explicit by connecting the relevant phrases of the task description and the associated code concepts and syntax. In addition, the programming language, the natural language, and an associated visualization were combined as multiple but parallel representations of the same programming concept. This connection is presented as a heuristic called language-sensitive teaching, or LST. *The resulting insight is that LST is no different from worked examples when it comes to being an effective didactic basis for an instructional video.* Both didactic bases did not have different effects on self-efficacy. In terms of code writing performance, the positive effects of LST found in the first cohort were not replicated in the second cohort.

In Application Study II, the relationship under consideration is made explicit by linking the relevant phrases in the task description and the associated code concepts and syntax. This linking is therefore similar to that used in Application Study I. The language used by the students was also considered in order to focus on language in Application Study II. The context were two categories of a rubric to assess the quality of teaching staff' responses to students' queries about code. Expert evaluation of the rubric revealed that these two language-focused categories are a relevant part of teaching staff's competencies. In addition, these language-focused categories can be assessed well on a rubric at multiple levels. *This leads to the conclusion that these versions, in which the relationship between programming language and natural language is made explicit, are fruitful for research and practice related to teaching staff.*

In Application Study III, the considered relationship is made explicit by transferring typical programming tasks into equivalents that use only natural language. The results show that tasks based on natural language are suitable for accurate measurement of commonsense computing, even if the suitability of the developed test as a predictive success factor was low. *The resulting insight is that the connection between programming language and natural language offers new possibilities and ways of thinking about assessments and task types. Thus, the connection is an asset for the development of assessments and task types in general in CER.*

These findings are then compared with results from related work. The effectiveness of multiple but parallel representations of previous work in mathematics education [153] could not be confirmed in this work. However, the relevance of the terms to research and teaching in postsecondary education is as salient as in K-12 [73]. This relevance is consistent with the theoretical considerations of relevance outlined by Diethelm and Goschler [33] in their research call on terms in CER.

The original perspective was that making the relationship between programming language and natural language could be the new perspective in research and teaching in computing education. However, this approach was not as effective as probably assumed. This was evident in the non-effectiveness of this approach as a didactic basis for instructional videos in Application Study I and the non-effectiveness as a suitable predictive factor in Application Study III. This does not mean, however, that this approach has been shown to be ineffective in general, as the findings and insights presented earlier indicate. Rather, the general insight is that it is how the relationship under consideration is made explicit that matters in arriving at a fruitful and effective approach. Here, the perspective of terms was particularly useful. Thus, this new approach is a new fruitful heuristic rather than a panacea.

5.2. Contributions and Strengths of the Thesis

First, the categories of contributions that the thesis has made are presented. This is followed by the advances that the thesis has made, both scientific in its field, namely CER, and outside. Then, a specific focus on the relevance of the contributions presented is added. Finally, the strengths of the studies conducted and the work in general are presented.

The categorization system of Draper and Maguire [37] is used for categorizing the contributions. They developed this system specifically for CER, so it is particularly well suited for this work. In general, in this work, contributions are divided into three categories. These are: (1) theoretical reasoning and predictions, (2) planned observation, and (3) tools. Draper and Maguire define the first category as follows: “To take a theory and then work out what it would predict in some new particular situation, and hence prepare the way for new empirical work on that situation, is a distinct research step. As such it is a contribution to knowledge.” [37, p. 13]. Here this process was carried out in the following way: The theoretical relationship between language and reasoning was applied in the context of learning programming languages. It was then predicted that making this relationship explicit to learners would be an effective approach to successful teaching-learning situations. Both this idea and its implementation together form the new approach presented in this thesis. Thus, the idea of this new approach is a stand-alone contribution in the form of theoretical considerations and predictions.

The second category of contributions is that of planned observations where the new approach has been put into practice. This includes two empirical studies, namely the case study on terms in the Basis Study and the experiment on instructional videos in the Application Study I. These two studies build on each other. First, the Basis Study includes data collection and initial findings on the implementation of the new approach in the form of a planned observation. However, no specific theory on terms was tested in this study. Second, building on these initial findings, a theory about the effectiveness of the new approach was developed. The resulting hypotheses were tested in the planned observation through an experiment.

The third and last category also deals with the implementation of the new approach, through the development and evaluation of new tools. The tools are the Natural Language Computing Test (NLCT) and the rubric for examining the quality of answers to students' questions about code. The NLCT is a formal programming aptitude test that directly measures commonsense computing without being based on a programming language. An item response theory approach was used for formal evaluation to ensure validity and reliability. The rubric, in turn, is less formal. Thus, it is more of a diagnostic tool than a formal assessment tool. Therefore, its evaluation is based on internal use with written explanations by student teaching assistants and external interviews with experts in the field. Nonetheless, both tools are contributions.

Next, the focus is on the scientific progress that the dissertation as a whole has made in its field, namely the field of computing education research (CER). The dissertation fills two knowledge gaps that have already been identified as such in its field. These include the research call on the terms used, which was contributed by Diethelm and Goschler [33]. In this call, they formulated four open questions about terms and their use by both students and teachers. To the best of the author's knowledge, the Basis Study is the first *empiric* study to address the two questions addressing students in postsecondary education. In fact, this is the case even though the research call was published back in 2015. The only other study of which we are aware is the exploratory study by Hermans et al. [73] with 20 high school students aged 11 to 13 years. However, they focused more on phonological issues and promoting reading code lines aloud in the classroom. Therefore, their results are not really applicable to teaching-learning situations in postsecondary education. In contrast to the lack of empirical research, practice-based recommendations for instruction were formulated to consider terms and language, e.g., [34]. The second knowledge gap that was addressed was a methodological one, namely testing a theory by conducting a randomized controlled trial, i.e., an experiment. In introductory programming research, conducting experiments is rare [171]. Thus, even recent reviews have formulated calls for conducting more formal experiments [120]. Thus, conducting an experiment is filling a knowledge gap in itself.

The other two studies included in this dissertation, i.e., the Application Studies II and III, also contribute to scientific progress in the field of CER. Unlike the previous ones, they did not respond to explicitly formulated calls for research. However, the tools presented in the studies, the NLCT and the rubric, are relevant to CER. The NLCT is the first programming aptitude test that directly measures commonsense computing without relying on programming languages. Therefore, it specifically measures the skills of novices who possess the skills but have not received formal training in programming languages prior to their introductory programming course. The rubric, in turn, is the first tool to both describe and assess the quality of the responses written by those who teach at various skill levels. These descriptions and assessments are relevant to the professional development of those how teach including student teaching assistants and others. Consequently, those how teach receive sound and differentiated feedback to provide quality support to students.

Following on from this, we will now take a broader perspective. Here we will present the advances that the thesis has made outside the realm of CER. First, we shift the focus from research to teaching introductory programming. The tasks of the questionnaires used in the studies, as well as the tools developed, can be used directly in teaching. In particular, the rubric is suitable for direct use because it does not require further assessment in the form of software. In addition, the specific material presented in the Basis Study can be used directly in programming classes. In this case, the specific material includes the presented code snippets, terms, and descriptions written by the students. All of this can be used to both diagnose the language used by students in class and to initiate a meta-discourse, as recommended [34].

Even more broadly, consider related research areas such as STEM education, which includes science, technology, engineering, and mathematics. The advance for STEM education in general is that a focus on language can be a fruitful approach. So far, this is only an established and long-standing approach in mathematics education. Erath et al. [45] has summarized the publications on language in mathematics education from the last forty years. In other subjects, this approach is increasing, as exemplified by the growing number of publications on learning “Chemish” [116], the language of chemistry, and a language focus in physics education [95].

For STEM education, several methods of this work can be applied to it. One method is the base study methodology for capturing terms and concepts in larger studies as interviews. This method consisted of a qualitative content analysis of short descriptions of atomic concepts. These atomic concepts can be different depending on the STEM topic, e.g. a chemical equation or a part of a circuit diagram as an atomic concept. Therefore, this developed method is an important methodological contribution to STEM education in general. This method is very practical as the investigation terms of atomic concepts (a qualitative approach) can also be used with larger sample sizes. It overcomes the limitations of similar qualitative approaches such as interviews with their small sample sizes. Another transferable method is the development and validation of aptitude tests containing only natural language based tasks. Here, the NLCT addressed and described them for testing programming aptitude. The general idea of the method is to focus on skills, here commonsense computing, rather than technical language vocabulary. This provides another way to assess prior knowledge, which is a common area across all STEM education subjects.

Broadly speaking, we outline the progress that the thesis means for the rest of the world. Vygotsky laid the foundation between language and thinking [190], in fact for *all* human activities. In this broadest sense, this work empirically shows that the same is true for introductory programming. Therefore, the results of this work support this theoretical foundation with empirical evidence.

Next, we focus explicitly on significance of findings. In general, language on the one hand is related to thinking, communication, and belonging on the other [69]. Language and terms as part of language are thus key factors in thinking, communication, and belonging. Learning, just like daily life in general, involves all these three domains. For learning to program, this is even more true, because programming languages are as alive in the brain as natural languages. Therefore, the significance of this work is based on the relevance of language and on the fact that this work brings this relevance to research practice.

Finally, the strengths of the thesis are presented. The strengths are (1) the inclusion of large data sets and (2) the use of rigorous methods. Both of which are not self-evident in CER, which was also described by Heckman et al. [66] in their review on empiricism in CER literature. Regarding (1), the Basis Study examined more than 1,800 free-text responses in depth. Application Study I comparing instructional videos used two cohorts of n=126 and n=367 participants. The Application Study III evaluating NLCT included an even larger data set with n=684 participants. In (2), the rigorous methods used differed. In Application Study I, a randomized experiment was conducted and the results were examined the following semester with a second cohort. This was particularly beneficial because one of the hypotheses could not be confirmed for the second cohort. The Application Study III used an item response theory approach to ensure that the assessment instrument measured the construct of commonsense computing accurately. Inter-rater agreement was considered in the Application Studies II and III. The NLCT coding manual was even tested with n=15 raters. In summary, both aspects contribute to the soundness of the findings.

5.3. Limitations and Threats to Validity

There are both limitations and various threats that warrant discussion. On the one hand, the limitations and threats listed in this section are those that are interwoven with the thesis as a whole. On the other hand, they represent those that appear in several of the studies conducted. A note in advance: The limitations listed below are unavoidable for scientific work in this field. The limitations do not go beyond what is expected of such a piece of scientific work. Even the most ambitious scientific work will therefore not go beyond these limits. However, it is good research practice in this field – and required by virtually all high-ranking conferences and journals – to be aware of the limitations and explicitly point them out. With this in mind, the limitations are listed below.

To begin with the limitations: Only a small number of common teaching-learning situations were studied. This is mainly due to the limited resources, especially time, available for the preparation of this thesis. The list of common teaching-learning situations is huge. It includes such diverse situations as giving lectures, working in small groups, one-on-one consultations with teaching assistants, planning lectures, holding office hours, and so on. Based on these exemplary situations, it is clear that this work must necessarily be limited to some of them. Moreover, the selection of the teaching-learning situations studied was influenced by the Corona pandemic and its impact on campus life in Germany. For this reason, some of the situations did not take place or took place only online. As a consequence of the situations studied and the methods used, the focus was on situations and research methods that can be carried out purely online.

For the second limitation, a quote from Application Study II is provided to illustrate the point: It is "only a snapshot at a point in time and may not be representative". This limitation applies to all studies conducted. In fact, the data for all studies were collected only at one point in time. Whether it was students' programming aptitude at the beginning of the semester, as measured by the newly developed Natural Language Computing Test. Or whether it was the more or less appropriate use of terms and technical language that students used when describing code snippets, see the results of the base study. Consideration of these snapshots evolved through the overarching research question. Accordingly, the focus was on various teaching-learning situations as areas of application. In contrast to this broad view, a survey and analysis of individual courses and developments of the relevant actors, i.e., primarily students, would be necessary. The extent to which this longitudinal aspect could be promising will be taken up in the next section in the implications for research.

The other limitations relate to the studies rather than to the thesis as a whole. The base study is a case study and therefore generalization to other CS1 students simply cannot be guaranteed. Incidentally, generalization was not one of the goals of the study. Rather, two opposing goals were pursued: An initial investigation of the relationship stated in the overarching research question and an initial filling of the research gaps on the terms used by students [33]. Therefore, the lack of generalization in this study is a limitation, but not a threat, as it was never intended.

Let us now turn to the internal threats: For all the studies conducted, and thus for the entire work, the motivation of the participants is a relevant factor. All participants were CS1 students excluded from Application Study II, which included student teaching assistants. Motivation is always a confounding factor in voluntary participation in studies. This is particularly relevant given that participation time was no less than 20 minutes for all studies and less laborious tasks such as multiple-choice tasks were not used. It may have been particularly pronounced in the studies that were conducted. This is also because participants were rewarded with incentives based on their completeness rather than correctness. This may have lowered the motivation to make a special effort. On the other hand, this was necessary to reward all participants fairly and equally for their participation. And to do so regardless of their current level of performance.

The second internal threat that appeared in all studies was the maturation effect. This effect refers to the fact that participants' measured abilities naturally improve over time. Here, the improvement is likely a result of learning in the course in general. Consequently, participants who later participated in the studies might have improved their skills. Thus, their maturation might have led to a higher rate of appropriate responses in the studies. To compensate for the maturation effect, the participation period for later data collection was shortened. For the base study, as the first study, data collection lasted nine weeks. However, for the second cohort of Application Study I, the participation period was shortened to two weeks. Further shortening, such as conducting the study during a lecture, was not possible. Even though this would have limited the participation period to a single appointment on a single day. This is due to the fact that attendance is not compulsory in the CS1 courses studied. Therefore, a not insignificant number of students do not attend the lecture, but work through the content independently and take advantage of other support opportunities in the course. Therefore, such considerations of overcoming the maturation effect were not possible for contextual reasons.

For the external threats, the singular foci is most relevant in several ways. The singular foci include the university, the course, the programming language, and the natural language. In the studies, CS1 courses at a single university, namely the Technical University of Darmstadt, were considered. In addition, only one CS1 course was considered in most cases. The only exception to this is Application Study I, which included two cohorts from two CS1 courses in consecutive fall semesters. In terms of languages, German was the only natural language considered and Java was the only programming language considered. As a consequence of these singular foci, only preliminary generalizations can be drawn for other universities, participants, and learning contexts.

The other external threat that appeared in the studies conducted was the lack of inclusion of demographic characteristics. The source of this threat was that the local ethics committee did not allow us to collect demographic data. Therefore, accurate demographic data were not collected in the first two data collections. These included the base study and the first cohort of the Application Study I. In the later data collection, a way was found to meet the requirements of the ethics committee and the goal of describing and categorizing participants. This was implemented in the form of a separate, optional demographic survey. However, in both the second cohort of Application Study I and in Application Study III, more students provided their demographic data than participated in the associated studies. Therefore, again, only tentative conclusions can be drawn about the demographic distribution of participants. As a result, there could be a potential bias in all studies regarding certain demographic characteristics of the participants. This could include language bias in particular. Here, students whose first language is not German might have more difficulties or a higher cognitive load when participating in the studies. As a reminder, all descriptions of the study procedure and the tasks themselves were formulated in German. This language bias could have occurred as approximately one-third of the CS1 students had a migration background. A migration background is often associated with the fact that German is not their first language.

In general, the limitations and external threats described are characterized by a focus on a particular context. This focus is inherent in the application of a new approach – such as the language-based approach used here. Therefore, this work cannot address them. Rather, some of them are also necessary choices – such as focusing on a single programming language to make answering the research question manageable. Not all limitations and threats could be addressed with mitigating measures, as the studies represent a single snapshot in time. Therefore, transferring the approach to other contexts as a mitigating measure could be part of further research. In contrast, the maturation effect as part of the internal threats could be attenuated in the later studies by shortening the participation time. Motivation as a confounding factor could not be compensated. Overall, most of the limitations and threats described can be addressed and overcome by future research, with the exception of the internal threat to motivation directly related to voluntary participation.

5.4. Implications for Teaching and Research

The implications for teaching and research are divided into implications that are directly related to the research and the results of the studies carried out, and research that develops the ideas further. The further development of research is described in detail for the most fruitful sub-area, the terms. Starting with the implications that arise directly from the studies, three implications are presented, all of which apply to both teaching and research.

The first direct implication is that of *direct use*. Here we mean both the tools developed, namely the NLCT and the rubric, and the questionnaire tasks in the Basis Study. All of these can be used directly in teaching and research. Beginning with the Basis Study, we have recommended several options for instruction for direct use in the associated paper, see Sect. 7. These include two instructions: (1) presenting examples of terms and code descriptions in class and then discussing them with students. As (2), teachers can compare their own descriptions of code snippets with those presented in the study. Teachers can then reflect on what terms they are asking students to use, what terms students are actually using, and whether these two uses correspond to the learning objectives set by the teachers. They can also reflect on whether they themselves are unintentionally using inadequate terms by using shorthands. For example, “the persons a and b” is a shorthand for the more appropriate term “the variables a and b of the type Person”.

Next, we describe the direct use of the rubric presented in Application Study II. One advantage of the rubric is that direct use is quite natural. This is based on the fact that the rubric is listed in its entirety in the associated paper and no additional software is needed for analysis. As mentioned earlier, the rubric aims to assess the quality of the answers to students' queries about code. In accordance with this objective, the direct use of the rubric is to assess the quality of the answers previously mentioned. In addition, the rubric can be directly used to provide feedback, structure reflection, plan and implement different learning activities, according to the experts interviewed. Thus, the rubric offers several areas of application for use in the classroom as well.

In contrast, direct use of the NLCT is possible but requires more effort. Because the NLCT is a formal assessment, a more precise measurement was required. This was provided by incorporating an item response theory based analysis. However, this involves effort and therefore direct application requires more effort. Nevertheless, we believe that the tasks themselves are also fruitful for direct use. In this case, direct use is relevant for both researchers and instructors because it is important to learn about students' programming aptitude.

The direct use is not limited to the assessment part of the developed tools, but is also intended as a self-assessment for the participants. In the free-text field for critique and comments, several participants of the Basis Study have addressed the use in terms of self-assessment for undergraduate assignments. Here, self-assessment is about knowing where you are right now, seeing what you can already do, and seeing what you can not yet do and identifying those gaps. This three-part concept of self-assessment also applies to the tools developed, namely the NLCT and the rubric. For the NLCT, students were given a reference solution after the participation phase to check themselves. For the rubric, it is even easier because the rubric itself contains descriptions of each of the three competency levels.

The second direct implication is that *diagnosis is important*. This conclusion has emerged primarily from the results of Application Study I. There, in the first of four phases of the experiment, the zone of next development of students was determined. This concept is based on Vygotsky's [189] "zone of proximal development" (ZPD), in which the learner cannot solve the task independently, but only with support. In our study, it was found that determining students' ZPD requires more fine-tuning. For research, this means that a focus of accurate assessment of students' competencies is needed. Here, ZPD assessment is particularly fruitful, as these concepts describe the next developmental step in learning.

Teaching, in turn, can also build on the importance of diagnosis. Adaptive learning seems promising here. Adaptive learning means that the selection of learning material is continuously adapted to the needs of the learner. However, adaptive learning does not have to mean that all learners have the same learning goal. Rather, it is about the learning path continuously adapting to learners as they work through the learning

material. Learners are often not a homogeneous group, but have individual learning levels and different previous experiences. Therefore, adaptive learning appears fruitful in overcoming the difficulties of a one-size-fits-all approach. However, adaptive learning requires appropriate and accurate diagnosis, which again underscores the claim made at the beginning that diagnosis is important.
The importance of diagnosis is not ignored in the other studies either. In the Base Study, the entire structure of the study of the terms used by the novices is a diagnosis. This diagnosis focused on the language used by the novices and, to some extent, on what misconceptions and correct conceptions they convey with the terms they use. In the Application Studies II and III, the connections are even more obvious, as these studies developed and assessed assessment instruments to diagnose various skills.

As third direct implication, *transfer to other languages* is recommended. In this work, Java was treated as the only programming language and German as the only natural language. How this poses an external threat was discussed in the previous section. Transfer to other languages, both programming languages and natural languages, is particularly recommended for the terms questionnaire in the Basis Study and for the NLCT tasks in Application Study III. In the following, special emphasis is placed on the exploration of terms. This is based on the following considerations. The starting point for the exploration of terms in the baseline study was the question of the implications of making the relationship between programming language and natural language explicit. However, this subfield has developed into a fruitful area of research in its own right. This is consistent with the fact that parts of the general research on terms had already been posed as open questions by Diethelm and Goschler [33]. In contrast, the language approach as a didactic basis for an instructional video was no better or worse than the basis of the worked examples. In this area, according to the results of the related study, namely Application Study I, less fruitful research directions are to be expected.

In the last part, the possible future work on terms is presented. It is presented from three perspectives: that of the *participants* in a teaching-learning situation, i.e., those who use terms and those who respond to the terms used, that of the *context*, and that of the *topic and content*. For the participant perspective, future work can empirically capture the terms used by teachers and instructors. In addition, these terms can be compared to those used by students. One can also consider the perspective of teachers and instructors on the terms used by students. This includes how they perceive (or not) the terms used in accordance with their own understanding of appropriateness.
In terms of context, research can examine the use of terms in environments other than the artificial ones of an online questionnaire used in the base study. Examples of other

environments include classroom observations similar to those of Becker er al. [14], but with a focus more specific to CS terms.

From the perspective of topic and content, variations and the inclusion of other themes are welcome in the exploration of terms. For example, Tew et al. [183] have included other common concepts that are fundamental to CS1, such as nested loops, selection statements with the alternative, recursion, and inheritance, to name a few. In addition, topics already covered in the base study can be explored in more depth. Examples of this are a questionnaire on terms for functions only. In this case, all code snippets to be described have the same focus.

Overall, research can vary and combine all of the opened perspectives. This shows how broad and diverse the research on terms can be. Thus, a well-founded picture of the use of terms can be formed.

6. Conclusion

The overarching research question was, *"What new insights emerge from exemplary teaching-learning situations in introductory programming in postsecondary education when the relationship between programming language and natural language is made explicit?"*

The most important result is that the consideration of language and the explicit connection between programming language and natural language represents a new perspective. This new perspective applies to all actors: researchers, teaching staff such as instructors, and students. Among other things, it offers new approaches and perhaps new solutions to old problems. The extent to which the explicit representation of the relationship is a relevant factor in the exploration and implementation of teaching-learning situations depends on the particular situation.

Summarizing the studies conducted, the results first show that terms describing a programming language are relevant in their own right. In this context, a term comprises a word or expression with a precise meaning and is part of the language. The relevance of terms also applies to diagnosis and building student conceptions, but to a lesser extent. Moreover, making this relationship explicit also affected the training of teaching staff in the context of teaching experiences. To provide more qualified and informed feedback, a rubric for the quality of responses to student questions about code was developed and evaluated. In doing so, the language perspective was integrated as a fundamental component of the rubric categories. Thinking programming language and natural language together also led to the following new approach: Measuring programming aptitude using programming tasks based only on natural language. And this both in the task and in the answer. The resulting Natural Language Computing Test, or NLCT, has proven to be highly accurate, valid, and reliable. However, as a basis for instructional videos to support students, the implementation of the language focus across multiple representations was unconvincing. The topic studied was the difference between dynamic and static types in Java.

The contribution of this thesis is both of a putting forth up a new meta-perspective to think about researching and teaching in computing education. This was achieved by identifying and bringing together various fragments of research on language in introductory programming as aspects with a common denominator. At the same time, it was shown that natural language and programming language and, more broadly, thinking have always had a relationship and have always been present in research. In practical terms, this work also showed how the use of terms (a qualitative approach) can nevertheless be used with larger sample sizes. This method, developed as part of the Basis Study, is an important methodological contribution to STEM education in general. This method overcomes the limitations of similar qualitative approaches such as interviews with their small sample sizes.

In addition, this work has listed various possible applications of the new perspective. In this way, the previous findings, which belong to basic research, were directly related to practice – or rather, subjected to a practical test. In this way, the limits of this new perspective were also identified and explored. The result of Application Study I is an example of this. In this case, the actual skill level of the students, their zone of proximal development, was probably more decisive than the basis of the instructional video.

As an additional contribution, this work fills several gaps in CER. The field of terms has more open questions than answers [33]. To the best of the author's knowledge, the Basis Study is the first empiric study to address the questions addressing students in postsecondary education. This is even though the research call was published in 2015. Moreover, conducting experiments is a rarity [171], and even recent reviews call for conducting more formal experiments [120]. In Application Study I, this gap was filled by conducting a randomized controlled trial, i.e., an experiment. The final area is assessments, on which research needs to focus more [112]. Both Application Study II and Study III developed new assessment tools to fill this gap. In the former, a rubric was developed for the quality of responses of teaching staff, while in the latter, a programming aptitude test was developed based solely on natural language skills.

In summary, the following applies to the new language approach presented as well as to others: There are situations in which it greatly enriches research and teaching. However, it is not a magic bullet or a set screw mechanism that always works. Rather, it is one of several possible perspectives from which to research and teach. Or metaphorically speaking: It is another component of the subject didactic toolbox of those who teach and research.

Bibliography

[1] Handbook of item response theory, volume one : models, 2016.

[2] ACM. Acm computing classification system. https://dl.acm.org/ccs, 2012.

[3] Alaofi, S., and Russell, S. A validated computer terminology test for predicting non-native english-speaking cs1 students' academic performance. In *Australasian Computing Education Conference* (New York, NY, USA, 2022), ACE '22, ACM, p. 133–142.

[4] Alhazbi, S., and Halabi, O. Flipping introductory programming class: Potentials, challenges, and research gaps. In *Proceedings of the 10th International Conference on Education Technology and Computers* (New York, NY, USA, 2018), ICETC '18, ACM, p. 27–32.

[5] Almassri, M., and Zaharudin, R. Effectiveness of flipped classroom pedagogy in programming education: A meta-analysis. *International Journal of Instruction 16*, 2 (2023), 267–290.

[6] Ameri, S., Fard, M. J., Chinnam, R. B., and Reddy, C. K. Survival analysis based framework for early prediction of student dropouts. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (New York, NY, USA, 2016), CIKM '16, ACM, p. 903–912.

[7] Andersen, E. B. The rating scale model. In *Handbook of Modern Item Response Theory*, W. J. Linden and R. K. Hambleton, Eds. Springer, New York, NY, 1997, pp. 67–84.

[8] Anwar, S., Bascou, N. A., Menekse, M., and Kardgar, A. A systematic review of studies on educational robotics. *Journal of Pre-College Engineering Education Research (J-PEER) 9*, 2 (2019), 2.

[9] Barlow-Jones, G., and van der Westhuizen, D. Problem solving as a predictor of programming performance. In *ICT Education* (Cham, 2017), J. Liebenberg and S. Gruner, Eds., Springer International Publishing, pp. 209–216.

[10] Bayman, P., and Mayer, R. E. A diagnosis of beginning programmers' misconceptions of basic programming statements. *Commun. ACM 26*, 9 (Sept. 1983), 677–679.

[11] Becker, B. The roles and challenges of computing terminology in non-computing disciplines. In *Proceedings of the 2021 Conference on United Kingdom & Ireland Computing Education Research* (New York, NY, USA, 2021), UKICER '21, ACM.

[12] Becker, B. A. Parlez-vous java? bonjour la monde != hello world: Barriers to programming language acquisition for non-native english speakers. In *30th Workshop of the Psychology of Programming Interest Group - PPIG '19* (2019).

[13] Becker, B. A. What does saying that 'programming is hard' really say, and about whom? *Commun. ACM 64*, 8 (jul 2021), 27–29.

[14] Becker, B. A., Gallagher, D., Denny, P., Prather, J., Gostomski, C., Norris, K., and Powell, G. From the horse's mouth: The words we use to teach diverse student groups across three continents. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2022), SIGCSE 2022, ACM, p. 71–77.

[15] Becker, B. A., and Quille, K. 50 years of cs1 at sigcse: A review of the evolution of introductory programming education research. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2019), SIGCSE '19, ACM, p. 338–344.

[16] Becker, B. W. Teaching cs1 with karel the robot in java. *SIGCSE Bull. 33*, 1 (feb 2001), 50–54.

[17] Berglund, A., and Lister, R. Introductory programming and the didactic triangle. In *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103* (AUS, 2010), ACE '10, Australian Computer Society, Inc., p. 35–44.

[18] Bock, R. D. *Item Response Theory*. Wiley, Hoboken, 2021.

[19] Bockmon, R., and Bourke, C. Validation of the placement skill inventory: A cs0/cs1 placement exam. In *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2023), SIGCSE 2023, ACM, p. 39–45.

[20] Bonar, J., and Soloway, E. Preprogramming knowledge: A major source of misconceptions in novice programmers. *Human–Computer Interaction 1*, 2 (1985), 133–161.

[21] Boone, W. J., and Staver, J. R. *Advances in Rasch Analyses in the Human Sciences*. Moremedia. Springer, Cham, Switzerland, 2020.

[22] Brown, N. C. C., and Altadmri, A. Novice java programming mistakes: Large-scale data vs. educator beliefs. *ACM Trans. Comput. Educ. 17*, 2 (may 2017).

[23] Bruner, J. S. *Toward a Theory of Instruction*. Belknap Press Series. Belknap Press of Harvard University, Cambridge, Massachusetts, 1967.

[24] Byrne, P., and Lyons, G. The effect of student attributes on success in programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2001), ITiCSE '01, ACM, p. 49–52.

[25] Casalnuovo, C., Sagae, K., and Devanbu, P. Studying the difference between natural and programming language corpora. *Empirical Software Engineering 24* (2019), 1823–1868.

[26] Chen, B., Azad, S., Haldar, R., West, M., and Zilles, C. A validated scoring rubric for explain-in-plain-english questions. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2020), SIGCSE '20, ACM, p. 563–569.

[27] Chen, C.-L., Cheng, S.-Y., and Lin, J. M.-C. A study of misconceptions and missing conceptions of novice java programmers. In *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)* (2012), The Steering Committee of The World Congress in Computer Science, Computer ..., p. 1.

[28] Chen, T.-Y., Lewandowski, G., McCartney, R., Sanders, K., and Simon, B. Commonsense computing: Using student sorting abilities to improve instruction. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2007), SIGCSE '07, ACM, p. 276–280.

[29] Cohen, J. *Statistical Power Analysis for the Behavioral Sciences*, second edition ed. Lawrence Erlbaum Associates, Hillsdale, 1988.

[30] Creswell, J. W., and Clark, V. L. P. *Designing and Conducting Mixed Methods Research*, 3 ed. SAGE Publications, 2017.

[31] Denny, P., Becker, B. A., Craig, M., Wilson, G., and Banaszkiewicz, P. Research this! questions that computing educators most want computing education researchers to answer. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (New York, NY, USA, 2019), ICER '19, ACM, p. 259–267.

[32] Denny, P., Manoharan, S., Speidel, U., Russello, G., and Chang, A. On the fairness of multiple-variant multiple-choice examinations. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2019), SIGCSE '19, ACM, p. 462–468.

[33] Diethelm, I., and Goschler, J. Questions on spoken language and terminology for teaching computer science. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2015), ITiCSE '15, ACM, p. 21–26.

[34] Diethelm, I., Goschler, J., Arnken, T., and Sentance, S. Language and computing. In *Computer Science Education. Perspectives on Teaching and Learning in School*, S. Sentance, E. Barendsen, N. R. Howard, and C. Schulte, Eds., 2 ed. Bloomsbury Publishing, London, 2023, pp. 167–182.

[35] Dijkstra, E. W., et al. On the cruelty of really teaching computing science. *Communications of the ACM 32*, 12 (1989), 1398–1404.

[36] Divine, G. W., Norton, H. J., Barón, A. E., and Juarez-Colunga, E. The wilcoxon–mann–whitney procedure fails as a test of medians. *The American Statistician 72*, 3 (2018), 278–286.

[37] Draper, S., and Maguire, J. The different types of contributions to knowledge (in cer): All needed, but not all recognised. *ACM Trans. Comput. Educ. 23*, 1 (jan 2023).

[38] Du, Y., Luxton-Reilly, A., and Denny, P. A review of research on parsons problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (New York, NY, USA, 2020), ACE'20, ACM, p. 195–202.

[39] Du Boulay, B. Some difficulties of learning to program. *Journal of Educational Computing Research 2*, 1 (1986), 57–73.

[40] Dunn, O. J. Multiple comparisons among means. *Journal of the American statistical association 56*, 293 (1961), 52–64.

[41] Duran, R. S., Rybicki, J.-M., Hellas, A., and Suoranta, S. Towards a common instrument for measuring prior programming knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2019), ITiCSE '19, ACM, p. 443–449.

[42] Edwards, J., Leinonen, J., Birthare, C., Zavgorodniaia, A., and Hellas, A. Programming versus natural language: On the effect of context on typing in cs1. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (New York, NY, USA, 2020), ICER '20, ACM, p. 204–215.

[43] Edwards, M. C., Houts, C. R., and Cai, L. A diagnostic procedure to detect departures from local independence in item response theory models. *Psychological Methods 23*, 1 (2018), 138–149.

[44] Endres, M., Fansher, M., Shah, P., and Weimer, W. To read or to rotate? comparing the effects of technical reading training and spatial skills training on novice programming ability. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (New York, NY, USA, 2021), ESEC/FSE 2021, ACM, p. 754–766.

[45] Erath, K., Ingram, J., Moschkovich, J., and Prediger, S. Designing and enacting instruction that enhances language for mathematics learning: a review of the state of development and research. *ZDM Mathematics Education 53* (2021), 245–262.

[46] Erdei, R., Springer, J. A., and Whittinghill, D. M. An impact comparison of two instructional scaffolding strategies employed in our programming laboratories: Employment of a supplemental teaching assistant versus employment of the pair programming methodology. In *2017 IEEE Frontiers in Education Conference (FIE)* (Oct 2017), pp. 1–6.

[47] Fedorenko, E., Ivanova, A., Dhamala, R., and Bers, M. U. The language of programming: A cognitive perspective. *Trends in Cognitive Sciences 23*, 7 (2019), 525–528.

[48] Feigenspan, J., Kästner, C., Liebig, J., Apel, S., and Hanenberg, S. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)* (June 2012), pp. 73–82.

[49] Fisler, K. The recurring rainfall problem. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (New York, NY, USA, 2014), ICER '14, ACM, p. 35–42.

[50] Flight, R. M., Bhatt, P. S., and Moseley, H. N. Information-content-informed kendall-tau correlation: Utilizing missing values. *bioRxiv* (2022).

[51] Floyd, B., Santander, T., and Weimer, W. Decoding the representation of code in the brain: An fmri study of code review and expertise. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)* (2017), pp. 175–186.

[52] Freeman, L. C. *Elementary applied statistics: for students in behavioral science*. New York: Wiley, 1965.

[53] Galotti, K. M., and Ganong, W. F. What non-programmers know about programming: Natural language procedure specification. *International Journal of Man-Machine Studies 22*, 1 (1985), 1–10.

[54] Gilsing, M., Pelay, J., and Hermans, F. Design, implementation and evaluation of the hedy programming language. *Journal of Computer Languages 73* (2022), 101158.

[55] Glass, G. V. Testing homogeneity of variances. *American Educational Research Journal 3*, 3 (1966), 187–190.

[56] Gold-Veerkamp, C., Abke, J., and Diethelm, I. A research approach to analyse and foster discipline-specific language competency in software engineering education. In *2016 IEEE Global Engineering Education Conference (EDUCON)* (2016), pp. 652–659.

[57] Gorson, J., and O'Rourke, E. Why do cs1 students think they're bad at programming? investigating self-efficacy and self-assessments at three universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (New York, NY, USA, 2020), ICER '20, ACM, p. 170–181.

[58] Große-bölting, G., Gerstenberger, D., Gildehaus, L., Mühling, A., and Schulte, C. Identity in higher computer education research: A systematic literature review. *ACM Trans. Comput. Educ. 23*, 3 (sep 2023).

[59] Guo, P. J., Markel, J. M., and Zhang, X. Learnersourcing at scale to overcome expert blind spots for introductory programming: A three-year deployment study on the python tutor website. In *Proceedings of the Seventh ACM Conference on Learning @ Scale* (New York, NY, USA, 2020), L@S '20, ACM, p. 301–304.

[60] Guzdial, M., and Adams, J. C. Disputing dijkstra, and birthdays in base 2. *Commun. ACM 64*, 3 (feb 2021), 12–13.

[61] Hagan, D., and Markham, S. Does it help to have some programming experience before beginning a computing degree program? In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2000), ITiCSE '00, ACM, p. 25–28.

[62] Hair, J. F. J., Black, W. C., and Babin, Barry J. Anderson, R. E. *Multivariate data analysis*, 8 ed. Cengage, Boston, 2019.

[63] Hart, A. Mann-whitney test is not just a test of medians: differences in spread can be important. *BMJ 323*, 7309 (aug 2001), 391–393.

[64] Hattie, J., and Timperley, H. The power of feedback. *Review of educational research 77*, 1 (2007), 81–112.

[65] Hawlitschek, A., Berndt, S., and Schulz, S. Empirical research on pair programming in higher education: a literature review. *Computer Science Education 33*, 3 (2023), 400–428.

[66] Heckman, S., Carver, J. C., Sherriff, M., and Al-zubidy, A. A systematic literature review of empiricism and norms of reporting in computing education research literature. *ACM Trans. Comput. Educ. 22*, 1 (oct 2021).

[67] Heinonen, A., Lehtelä, B., Hellas, A., and Fagerholm, F. Synthesizing research on programmers' mental models of programs, tasks and concepts — a systematic literature review. *Information and Software Technology 164* (2023), 107300.

[68] Hellas, A., Ihantola, P., Petersen, A., Ajanovski, V. V., Gutica, M., Hynninen, T., Knutas, A., Leinonen, J., Messom, C., and Liao, S. N. Predicting academic performance: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2018), ITiCSE 2018 Companion, ACM, p. 175–199.

[69] Heller, V., and Morek, M. Academic discourse as situated practice: An introduction. *Linguistics and Education 31* (09 2015), 174–186.

[70] Hendrik, H., and Hamzah, A. Flipped classroom in programming course: A systematic literature review. *International Journal of Emerging Technologies in Learning (iJET) 16*, 2 (2021), 220–236.

[71] Herman, G. L., Kaczmarczyk, L., Loui, M. C., and Zilles, C. Proof by incomplete enumeration and other logical misconceptions. In *Proceedings of the Fourth International Workshop on Computing Education Research* (New York, NY, USA, 2008), ICER '08, ACM, p. 59–70.

[72] Hermans, F. Hedy: A gradual language for programming education. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (New York, NY, USA, 2020), ICER '20, ACM, p. 259–270.

[73] Hermans, F., Swidan, A., and Aivaloglou, E. Code phonology: An exploration into the vocalization of code. In *Proceedings of the 26th Conference on Program Comprehension* (New York, NY, USA, 2018), ICPC '18, ACM, p. 308–311.

[74] Hermans, F., Swidan, A., Aivaloglou, E., and Smit, M. Thinking out of the box: Comparing metaphors for variables in programming education. In *Proceedings of the 13th Workshop in Primary and Secondary Computing Education* (New York, NY, USA, 2018), WiPSCE '18, ACM.

[75] Hertz, M. What do ”cs1” and ”cs2” mean? investigating differences in the early courses. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2010), SIGCSE '10, ACM, p. 199–203.

[76] Herzog, M. H., Francis, G., and Clarke, A. *Understanding Statistics and Experimental Design*. Springer Cham, 2019.

[77] Heumann, C., Schomaker, M., and Shalabh. *Introduction to Statistics and Data Analysis : With Exercises, Solutions and Applications in R*. Springer, Cham, 2016.

[78] Hidalgo-Céspedes, J., Marín-Raventós, G., Lara-Villagrán, V., and Villalobos-Fernández, L. Effects of oral metaphors and allegories on programmingproblem solving. *Computer Applications in Engineering Education 26*, 4 (2018), 852–871.

[79] Holden, E., and Weeden, E. The impact of prior experience in an information technology programming course sequence. In *Proceedings of the 4th Conference on Information Technology Curriculum* (New York, NY, USA, 2003), CITC4 ’03, ACM, p. 41–46.

[80] Holland, S., Griffiths, R., and Woodman, M. Avoiding object misconceptions. In *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 1997), SIGCSE ’97, ACM, p. 131–134.

[81] Holmboe, C. Conceptualization and labelling as cognitive challenges for students of data modelling. *Computer Science Education 15*, 2 (2005), 143–161.

[82] Hubbard, A. Pedagogical content knowledge in computing education: a review of the research literature. *Computer Science Education 28*, 2 (2018), 117–135.

[83] Hubwieser, P., Magenheim, J., Mühling, A., and Ruf, A. Towards a conceptualization of pedagogical content knowledge for computer science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research* (New York, NY, USA, 2013), ICER ’13, ACM, p. 1–8.

[84] Indurkhya, B. Metaphor and cognition.

[85] Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O’Reilly, U.-M., Bers, M. U., and Fedorenko, E. Comprehension of computer code relies primarily on domain-general executive brain regions. *eLife 9* (dec 2020).

[86] Izu, C. Looking at cs1 through three colour-tinted glasses. In *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2021), SIGCSE ’21, ACM, p. 1291.

[87] Jafari, M., and Ansari-Pour, N. Why, when and how to adjust your p values? *Cell Journal (Yakhteh) 20*, 4 (2019), 604.

[88] Johnson, R. B., Onwuegbuzie, A. J., and Turner, L. A. Toward a definition of mixed methods research. *Journal of Mixed Methods Research 1*, 2 (2007), 112–133.

[89] Kanika, Chakraverty, S., and Chakraborty, P. Tools and techniques for teaching computer programming: A review. *Journal of Educational Technology Systems 49*, 2 (2020), 170–198.

[90] Kempert, S., Schalk, L., and Saalbach, H. Übersichtsartikel: Sprache als werkzeug des lernens: Ein überblick zu den kommunikativen und kognitiven funktionen der sprache und deren bedeutung für den fachlichen wissenserwerb. *Psychologie in Erziehung und Unterricht 66*, 3 (2019), 176–195.

[91] Kinnunen, P., Marttila-Kontio, M., and Pesonen, E. Getting to know computer science freshmen. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2013), Koli Calling '13, ACM, p. 59–66.

[92] Kinnunen, P., McCartney, R., Murphy, L., and Thomas, L. Through the eyes of instructors: A phenomenographic investigation of student success. In *Proceedings of the Third International Workshop on Computing Education Research* (New York, NY, USA, 2007), ICER '07, ACM, p. 61–72.

[93] Kinnunen, P., Meisalo, V., and Malmi, L. Have we missed something? identifying missing types of research in computing education. In *Proceedings of the Sixth International Workshop on Computing Education Research* (New York, NY, USA, 2010), ICER '10, ACM, p. 13–22.

[94] Kohn, T. Variable evaluation: An exploration of novice programmers' understanding and common misconceptions. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2017), SIGCSE '17, ACM, p. 345–350.

[95] Krabbe, H., Rincke, K., and Aleksov, R. Language in physics instruction. In *Physics Education*. Springer, Cham, 2022, pp. 361–382.

[96] Krippendorff, K. *The Reliability of Generating Data*. Chapman and Hall/CRC, 2022.

[97] Krueger, R., Huang, Y., Liu, X., Santander, T., Weimer, W., and Leach, K. Neurological divide: An fmri study of prose and code writing. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering* (New York, NY, USA, 2020), ICSE '20, ACM, p. 678–690.

[98] Kuckartz, U. *Qualitative Inhaltsanalyse : Methoden, Praxis, Computerunterstützung*. Beltz Juventa, Weinheim, 2012.

[99] Kuckartz, U., and Rädiker, S. *Qualitative Content Analysis : Methods, Practice and Software*, 2 ed. Sage, 2023.

[100] König, J., Blömeke, S., Paine, L., Schmidt, W. H., and Hsieh, F.-J. General pedagogical knowledge of future middle school teachers: On the complex ecology of teacher education in the united states, germany, and taiwan. *Journal of Teacher Education 62*, 2 (2011), 188–201.

[101] Ledgard, H., Whiteside, J. A., Singer, A., and Seymour, W. The natural language of interactive systems. *Commun. ACM 23*, 10 (oct 1980), 556–563.

[102] Leeper, R. R., and Silver, J. L. Predicting success in a first programming course. In *Proceedings of the Thirteenth SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 1982), SIGCSE '82, ACM, p. 147–150.

[103] Lei, Y., and Allen, M. English language learners in computer science education: A scoping review. In *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1* (New York, NY, USA, 2022), SIGCSE 2022, ACM, p. 57–63.

[104] Lemke, J. L. *Talking science: Language, learning, and values*. Ablex Publishing Corporation, Norwood, New Jersey, 1990.

[105] Lishinski, A., Yadav, A., Good, J., and Enbody, R. Learning to program: Gender differences and interactive effects of students' motivation, goals, and self-efficacy on performance. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (New York, NY, USA, 2016), ICER '16, ACM, p. 211–220.

[106] Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., et al. A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin 36*, 4 (2004), 119–150.

[107] Lister, R., Fidge, C., and Teague, D. Further evidence of a relationship between explaining, tracing and writing skills in introductory programming. In *Proceedings of the 14th Annual ACM SIGCSE Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2009), ITiCSE '09, ACM, p. 161–165.

[108] Lister, R., Simon, B., Thompson, E., Whalley, J. L., and Prasad, C. Not seeing the forest for the trees: novice programmers and the solo taxonomy. *ACM SIGCSE Bulletin 38*, 3 (2006), 118–122.

[109] Lopez, M., Whalley, J., Robbins, P., and Lister, R. Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth International Workshop on Computing Education Research* (New York, NY, USA, 2008), ICER '08, ACM, p. 101–112.

[110] Lord, F. M. *Applications of item response theory to practical testing problems*. Routledge, Hillsdale, NJ, 1980.

[111] Luxton-Reilly, A., and Petersen, A. The compound nature of novice programming assessments. In *Proceedings of the Nineteenth Australasian Computing Education Conference* (New York, NY, USA, 2017), ACE '17, ACM, p. 26–35.

[112] Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., and Szabo, C. Introductory programming: A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2018), ITiCSE 2018 Companion, ACM, p. 55–106.

[113] Malmi, L., Sheard, J., Kinnunen, P., Simon, and Sinclair, J. Computing education theories: What are they and how are they used? In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (New York, NY, USA, 2019), ICER '19, ACM, p. 187–197.

[114] Manches, A., McKenna, P. E., Rajendran, G., and Robertson, J. Identifying embodied metaphors for computing education. *Computers in Human Behavior 105* (2020), 105859.

[115] Margulieux, L., Ketenci, T. A., and Decker, A. Review of measurements used in computing education research and suggestions for increasing standardization. *Computer Science Education 29*, 1 (2019), 49–78.

[116] Markic, S., and Childs, P. E. Language and the teaching and learning of chemistry. *Chemistry Education Research and Practice 17*, 3 (2016), 434–438.

[117] Masters, G. N., and Wright, B. D. The partial credit model. In *Handbook of Modern Item Response Theory*, W. J. Linden and R. K. Hambleton, Eds. Springer, New York, NY, 1997, p. 101–121.

[118] Mayring, P. *Qualitative Content Analysis: A Step-by-step Guide*. Sage, 2022.

[119] McCartney, R., Boustedt, J., Eckerdal, A., Sanders, K., and Zander, C. Folk pedagogy and the geek gene: Geekiness quotient. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (New York, NY, USA, 2017), SIGCSE '17, ACM, p. 405–410.

[120] Medeiros, R. P., Ramalho, G. L., and Falcão, T. P. A systematic literature review on teaching and learning introductory programming in higher education. *IEEE Transactions on Education 62*, 2 (2018), 77–90.

[121] Meyer, J., and Land, R. Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines. In *Improving Student Learning: Theory and Practice - 10 Years on*, C. Rust, Ed. Oxford Brookes University, 2003, pp. 412–424.

[122] Meyer, J. H. Threshold concepts and pedagogic representation. *Education+ Training 58*, 4 (2016), 463–475.

[123] Miller, C. S. Metonymy and reference-point errors in novice programming. *Computer Science Education 24*, 2-3 (2014), 123–152.

[124] Miller, C. S. Human language and its role in reference-point errors. In *27th Workshop of the Psychology of Programming Interest Group* (2016), PPIG.

[125] Miller, C. S., and Settle, A. Learning to get literal: Investigating reference-point difficulties in novice programming. *ACM Trans. Comput. Educ. 19*, 3 (May 2019).

[126] Miller, L. A. Natural language programming: Styles, strategies, and contrasts. *IBM Systems Journal 20*, 2 (1981), 184–215.

[127] Mirza, D., Conrad, P. T., Lloyd, C., Matni, Z., and Gatin, A. Undergraduate teaching assistants in computer science: A systematic literature review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (New York, NY, USA, 2019), ICER '19, ACM, p. 31–40.

[128] Morgan, D. L. *Integrating qualitative and quantitative methods: A pragmatic approach*. Sage publications, 2014.

[129] Muldner, K., Jennings, J., and Chiarelli, V. A review of worked examples in programming activities. *ACM Trans. Comput. Educ.* (sep 2022). Just Accepted.

[130] Muraki, E. A generalized partial credit model. In *Handbook of Modern Item Response Theory*, W. J. Linden and R. K. Hambleton, Eds. Springer, New York, NY, 1997, pp. 153–164.

[131] Murphy, L., Fitzgerald, S., Lister, R., and McCauley, R. Ability to 'explain in plain english' linked to proficiency in computer-based programming. In *Proceedings of the Ninth Annual International Conference on International Computing Education Research* (New York, NY, USA, 2012), ICER '12, ACM, p. 111–118.

[132] Myers, B. A., Pane, J. F., and Ko, A. J. Natural programming languages and environments. *Commun. ACM 47*, 9 (sep 2004), 47–52.

[133] Ni, L., Bausch, G., and Benjamin, R. Computer science teacher professional development and professional learning communities: A review of the research literature. *Computer Science Education 33*, 1 (2023), 29–60.

[134] Noone, M., and Mooney, A. Visual and textual programming languages: a systematic review of the literature. *Journal of Computers in Education 5* (2018), 149–174.

[135] on Innovation, C., and in Computer Science Education (ITiCSE) 2023, T. Call for working groups, 2023.

[136] Onorato, L. A., and Schvaneveldt, R. W. Programmer-nonprogrammer differences in specifying procedures to people and computers. *Journal of Systems and Software 7*, 4 (1987), 357–369.

[137] Oser, F. K., Näpflin, C., Hofer, C., and Aerni, P. *Towards a Theory of Negative Knowledge (NK): Almost-Mistakes as Drivers of Episodic Memory Amplification*. Springer Netherlands, Dordrecht, 2012, pp. 53–70.

[138] Paek, I., and Cole, K. *Using R for item response theory model applications*. Taylor & Francis, Abingdon, Oxon, 2020.

[139] Paiva, J. C., Leal, J. P., and Figueira, A. Automated assessment in computer science education: A state-of-the-art review. *ACM Trans. Comput. Educ. 22*, 3 (jun 2022).

[140] Pane, J. F. *A Programming System for Children that is Designed for Usability:(appendices)*. PhD thesis, Citeseer, 2002.

[141] Pane, J. F., and Myers, B. A. *Usability issues in the design of novice programming systems*. Carnegie-Mellon University. Department of Computer Science, 1996.

[142] Pane, J. F., Myers, B. A., et al. Studying the language and structure in non-programmers' solutions to programming problems. *International Journal of Human-Computer Studies 54*, 2 (2001), 237–264.

[143] Parry, A. Investigating the relationship between programming and natural languages within the primm framework. In *Proceedings of the 15th Workshop on Primary and Secondary Computing Education* (New York, NY, USA, 2020), WiPSCE '20, ACM.

[144] Patitsas, E., Berlin, J., Craig, M., and Easterbrook, S. Evidence that computer science grades are not bimodal. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (New York, NY, USA, 2016), ICER '16, ACM, p. 113–121.

[145] Pea, R. D. Language-independent conceptual "bugs" in novice programming. *Journal of Educational Computing Research 2*, 1 (1986), 25–36.

[146] Petersen, A., Craig, M., and Zingaro, D. Reviewing cs1 exam question content. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), SIGCSE '11, ACM, p. 631–636.

[147] Pintrich, P. R., et al. A manual for the use of the motivated strategies for learning questionnaire (mslq).

[148] Pivkina, I. Peer learning assistants in undergraduate computer science courses. In *2016 IEEE Frontiers in Education Conference (FIE)* (Oct 2016), pp. 1–4.

[149] Porter, L., and Simon, B. Retaining nearly one-third more majors with a trio of instructional best practices in cs1. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2013), SIGCSE '13, ACM, p. 165–170.

[150] Portnoff, S. R. The introductory computer programming course is first and foremost a language course. *ACM Inroads 9*, 2 (apr 2018), 34–52.

[151] Portnoff, S. R. A new pedagogy to address the unacknowledged failure of american secondary cs education. *ACM Inroads 11*, 2 (may 2020), 22–45.

[152] Prat, C. S., Madhyastha, T. M., Mottarella, M. J., and Kuo, C.-H. Relating natural language atitude to individual differences in learning programming languages. *Scientific reports 10*, 1 (2020), 1–10.

[153] Prediger, S., and Wessel, L. Fostering german-language learners' constructions of meanings for fractions—design and effects of a language- and mathematics-integrated intervention. *Mathematics Education Research Journal 25*, 3 (jun 2013), 435–456.

[154] Qian, Y., and Lehman, J. Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Trans. Comput. Educ. 18*, 1 (Oct. 2017).

[155] Qian, Y., and Lehman, J. D. Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning 5*, 2 (2016), 73–83.

[156] Raadt, A. D., Warrens, M. J., Bosker, R. J., and Kiers, H. A. L. Kappa coefficients for missing data. *Educational and Psychological Measurement 79*, 3 (2019), 558–576.

[157] Randolph, J., Julnes, G., Sutinen, E., and Lehman, S. A methodological review of computer science education research. *Journal of Information Technology Education: Research 7*, 1 (2008), 135–162.

[158] Rao, P., Sundaresh, V., Venkatasubramanian, V., Kumar, V., R, S., and Kumar, N. S. Gradual and tolerant programming for novices. In *2022 IEEE Global Engineering Education Conference (EDUCON)* (2022), pp. 1460–1466.

[159] Razali, N. M., Wah, Y. B., et al. Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests. *Journal of statistical modeling and analytics 2*, 1 (2011), 21–33.

[160] Rountree, J., and Rountree, N. Issues regarding threshold concepts in computer science. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95* (AUS, 2009), ACE '09, Australian Computer Society, Inc., p. 139–146.

[161] Samejima, F. Graded response model. In *Handbook of Modern Item Response Theory*, W. J. Linden and R. K. Hambleton, Eds. Springer, New York, NY, 1997, pp. 85–100.

[162] Sammet, J. E. The use of english as a programming language. *Commun. ACM 9*, 3 (mar 1966), 228–230.

[163] Sanders, K., Sheard, J., Becker, B. A., Eckerdal, A., Hamouda, S., and Simon. Inferential statistics in computing education research: A methodological review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research* (New York, NY, USA, 2019), ICER '19, ACM, p. 177–185.

[164] Sanford, J. P., Tietz, A., Farooq, S., Guyer, S., and Shapiro, R. B. Metaphors we teach by. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), SIGCSE '14, ACM, p. 585–590.

[165] Selvaraj, A., Zhang, E., Porter, L., and Soosai Raj, A. G. Live coding: A review of the literature. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (New York, NY, USA, 2021), ITiCSE '21, ACM, p. 164–170.

[166] Sentance, S., and Waite, J. Teachers' perspectives on talk in the programming classroom : Language as a mediator. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (New York, NY, USA, 2021), ICER 2021, ACM, p. 266–280.

[167] Sheard, J., Carbone, A., Lister, R., Simon, B., Thompson, E., and Whalley, J. L. Going solo to assess novice programmers. In *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2008), ITiCSE '08, ACM, p. 209–213.

[168] Sheard, J., Simon, S., Hamilton, M., and Lönnberg, J. Analysis of research into the teaching and learning of programming. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (New York, NY, USA, 2009), ICER '09, ACM, p. 93–104.

[169] Shulman, L. S. Those who understand: Knowledge growth in teaching. *Educational researcher 15*, 2 (1986), 4–14.

[170] Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., and Brechmann, A. Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering* (New York, NY, USA, 2014), ICSE 2014, ACM, p. 378–389.

[171] Simon, Carbone, A., de Raadt, M., Lister, R., Hamilton, M., and Sheard, J. Classifying computing education papers: Process and results. In *Proceedings of the Fourth International Workshop on Computing Education Research* (New York, NY, USA, 2008), ICER '08, ACM, p. 161–172.

[172] Simon, Clear, A., Carter, J., Cross, G., Radenski, A., Tudor, L., and Tõnisson, E. What's in a name? international interpretations of computing education terminology. In *Proceedings of the 2015 ITiCSE on Working Group Reports* (New York, NY, USA, 2015), ITICSE-WGR '15, ACM, p. 173–186.

[173] Simon, B., Chen, T.-Y., Lewandowski, G., McCartney, R., and Sanders, K. Commonsense computing: What students know before we teach (episode 1: Sorting). In *Proceedings of the Second International Workshop on Computing Education Research* (New York, NY, USA, 2006), ICER '06, ACM, p. 29–40.

[174] Sindre, G. Code writing vs code completion puzzles: Analyzing questions in an e-exam. In *2020 IEEE Frontiers in Education Conference (FIE)* (Uppsala, Sweden, 2020), IEEE, pp. 1–9.

[175] Smith, D. H., Hao, Q., Jagodzinski, F., Liu, Y., and Gupta, V. Quantifying the effects of prior knowledge in entry-level programming courses. In *Proceedings of the ACM Conference on Global Computing Education* (New York, NY, USA, 2019), CompEd '19, ACM, p. 30–36.

[176] Sorva, J. Misconceptions and the beginner programmer. In *Computer Science Education. Perspectives on Teaching and Learning in School*, S. Sentance, E. Barendsen, N. R. Howard, and C. Schulte, Eds., 2 ed. Bloomsbury Publishing, London, 2023, pp. 167–182.

[177] Stefik, A., and Gellenbeck, E. Empirical studies on programming language stimuli. *Software Quality Journal 19*, 1 (08 2010), 65–99.

[178] Steinhorst, P., Petersen, A., and Vahrenhold, J. Revisiting self-efficacy in introductory programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (New York, NY, USA, 2020), ICER '20, ACM, p. 158–169.

[179] Strong, G., Higgins, C., Bresnihan, N., and Millwood, R. A survey of the prior programming experience of undergraduate computing and engineering students in ireland. In *Tomorrow's Learning: Involving Everyone. Learning with and about Technologies and Computing* (Cham, 2017), M. W. Arthur Tatnall, Ed., Springer, pp. 473–483.

[180] Swidan, A., and Hermans, F. The effect of reading code aloud on comprehension: An empirical study with school students. In *Proceedings of the ACM Conference on Global Computing Education* (New York, NY, USA, 2019), CompEd '19, ACM, p. 178–184.

[181] Tashakkori, A., and Teddlie, C. Handbook on mixed methods in the behavioral and social sciences, 2010.

[182] Taylor, C., Zingaro, D., Porter, L., Webb, K. C., Lee, C., and Clancy, M. Computer science concept inventories: past and future. *Computer Science Education 24*, 4 (2014), 253–276.

[183] Tew, A. E., and Guzdial, M. Developing a validated assessment of fundamental cs1 concepts. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2010), SIGCSE '10, ACM, p. 97–101.

[184] Tew, A. E., and Guzdial, M. The fcs1: A language independent assessment of cs1 knowledge. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), SIGCSE '11, ACM, p. 111–116.

[185] Tukiainen, M., and Mönkkönen, E. Programming aptitude testing as a prediction of learning to program. In *14th Workshop of the Psychology of Programming Interest Group* (London, UK, 2002), PPIG, pp. 47–57.

[186] VanDeGrift, T., Bouvier, D., Chen, T.-Y., Lewandowski, G., McCartney, R., and Simon, B. Commonsense computing (episode 6): Logic is harder than pie. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), Koli Calling '10, ACM, p. 76–85.

[187] Venables, A., Tan, G., and Lister, R. A closer look at tracing, explaining and code writing skills in the novice programmer. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop* (New York, NY, USA, 2009), ICER '09, ACM, p. 117–128.

[188] Vihavainen, A., Airaksinen, J., and Watson, C. A systematic review of approaches for teaching introductory programming and their influence on success. In *Proceedings of the Tenth Annual Conference on International Computing Education Research* (New York, NY, USA, 2014), ICER '14, ACM, p. 19–26.

[189] Vygotsky, L. S. *Mind in society: Development of higher psychological processes*. Harvard university press, 1978.

[190] Wertsch, J. V. Dialogue and dialogism in a socio-cultural approach to mind. In *The dynamics of dialogue*, K. Marková, Ivana; Foppa, Ed. Harvester Wheatsheaf, New York, London, 1990, pp. 62–82.

[191] Woollard, J. The implications of the pedagogic metaphor for teacher education in computing. *Technology, Pedagogy and Education 14*, 2 (2005), 189–204.

[192] Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., and Ko, A. J. A theory of instruction for introductory programming skills. *Computer Science Education 29*, 2-3 (jan 2019), 205–253.

[193] Yadav, A., and Berges, M. Computer science pedagogical content knowledge: Characterizing teacher performance. *ACM Trans. Comput. Educ. 19*, 3 (may 2019).

[194] Yen, W. M. Effects of local item dependence on the fit and equating performance of the three-parameter logistic model. *Applied Psychological Measurement 8*, 2 (1984), 125–145.

[195] Zhan, Z., He, L., Tong, Y., Liang, X., Guo, S., and Lan, X. The effectiveness of gamification in programming education: Evidence from a meta-analysis. *Computers and Education: Artificial Intelligence 3* (2022), 100096.

[196] Zingaro, D. Peer instruction contributes to self-efficacy in cs1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2014), SIGCSE '14, ACM, p. 373–378.

[197] Zingaro, D., Petersen, A., and Craig, M. Stepping up to integrative questions on cs1 exams. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2012), SIGCSE '12, ACM, p. 253–258.

Part II.

Publications

7. Basis Study: Terms Which Novice Programmers Use to Describe Code Snippets in Java

Bibliographic Information

This contribution is based on the following publication, the version below being the accepted version.

Svana Esche and Karsten Weihe. 2023. Case Study on the Terms Novice Programmers Use to Describe Code Snippets in Java. In *IEEE Transactions on Education*, vol. 66, no. 6, pp. 642-653, 2023, `https://doi.org/10.1109/TE.2023.3290259`. [1]

The contribution of the author of this thesis is summarized as follows.

"As corresponding author, Svana Esche has contributed in all phases, including developing the research questions, conceptual research design, planning research activities, data collection, data analysis and interpretation, and writing the manuscript. Karsten Weihe provided valuable feedback as a co-author during the revision part of the writing process of this publication."

[1] In reference to IEEE copyrighted material which is used with permission in this thesis, the IEEE does not endorse any of Technical University of Darmstadt's products or services. Internal or personal use of this material is permitted. If interested in reprinting/republishing IEEE copyrighted material for advertising or promotional purposes or for creating new collective works for resale or redistribution, please go to `http://www.ieee.org/publications_standards/publications/rights/rights_link.html` to learn how to obtain a License from RightsLink. If applicable, University Microfilms and/or ProQuest Library, or the Archives of Canada may supply single copies of the dissertation.

Case Study on the Terms Novice Programmers Use to Describe Code Snippets in Java

Svana Esche, Karsten Weihe

Abstract—*Contribution:* **Most work on languages in computing education currently focuses on non-native speakers. In contrast, to the best of the authors' knowledge, this article is the first response to the call for research on terms that takes into account the terms used by novices in their language.**

Background: **Terms are key factors in communication, thinking, and belonging to a community, but questions about terms and their use by novices are listed as research calls that have not yet been answered for novices. Terms can be used to identify misconceptions about programming languages and conceptions, which is an ongoing trend in computing education research.**

Research Questions: **(RQ1) What terms do novices use to describe code snippets in Java? (RQ2) To what extent and in what way do the terms indicate programming language misconceptions or conceptions?**

Methodology: **An inductive-dominant qualitative content analysis (QCA) was conducted to examine the terms used by novices. The data consisted of more than 1800 free-text responses from 123 undergraduate CS1 students from the fall 2020/21 semester.**

Findings: **In general, novices use technical language appropriately in their terms when describing code but individual responses and terms revealed a wide diversity. The terms reflect both undescribed programming language misconceptions and conceptions, confirming previous research in this area.**

Index Terms—**Computer science, Terminology, Misconceptions, Mental models, Survey, Qualitative, Higher education**

I. Introduction

LANGUAGE is an important factor in academic success. Learning a programming language is no exception. This is all the more true because programming languages are as alive in the brain as natural languages [1], [2]. However, both language and the terms used by novices are areas with more unanswered questions than answered ones [3]. In this article, the term *language* is used when referring to a natural language (such as English). The term *programming language* is used when referring to such. A *term*, in turn, includes a word or expression with a precise meaning and is part of the language.

In general, Vygotsky laid the foundation for the connection between language and thinking [4]. Terms are part of the language through their definition and therefore give insight into the thinking of individuals. Two other perspectives from other disciplines support this connection. These are the threshold concepts already applied in computing education research (CER) and the perspective of linguists. The threshold concept links "to think like a Computer Scientist" [5] with using "appropriate language" [6] for the connection between thinking and language mentioned earlier. Some linguists, however, consider this connection to be one of the three functions of academic language, in which language acts "as a tool of thinking" [7]. The other functions are those of communication and social belonging. These three perspectives reveal the same thing, namely the importance of language. Therefore, terms as a part of a language are a relevant area of research.

This article is about novice programmers and the terms they use. These relate to the first two open questions, which refer to the terms and their usage, regardless of the speaker [3]. In contrast to the domain of non-native English speakers and the terms used by teachers, this article focuses on the terms used by novices in their language.

Building on Vygotsky's idea that language, and thus terms, provide insights into thinking, this article explores how terms reveal programming language (PL) misconceptions and conceptions. In this work, the term conception is used as an antonym to the term misconception. Thus, a PL conception is defined as a statement that (1) refers to a specific code and (2) does not contradict the syntax and/or semantics of the programming language that occur in that code.

PL misconceptions, here in the definition of Chiodini et al. [8], have been the subject of interest since the beginning of CER [9]. Previous work on novices' statements on PL code often consisted of interview studies, which are time-consuming and result in small sample sizes, e.g., [10], [11]. This article aims to fill a gap revealed by novices' statements about code in a larger sample. Based on the previous analysis, the research questions are:

(RQ1) What terms do novices use to describe code snippets in Java?

(RQ2) To what extent and in what way do the terms indicate programming language misconceptions or conceptions?

The first question is primary, while the second question is secondary. Therefore, a more in-depth analysis will take place for RQ1, while the answer to RQ2 will be more shallowed and thus limited to a few examples.

II. Related Work

The terms and their usage are the starting point, both for the related work and for answering the research questions (Sect. II-A). First, a connection to the language-related theoretical background is established (Sect. II-B), as this serves as motivation for the exploration of the terms and their placement

Manuscript received April 27, 2022; revised November 22, 2022 and 27 March. *(Corresponding author: Svana Esche)*

The authors are with the Department of Computer Science, Technical University of Darmstadt, Darmstadt, Germany. (e-mail: svana.esche@tu-darmstadt.de, karsten.weihe@tu-darmstadt.de)

The work of Svana Esche was supported in part by the "Qualitätsoffensive Lehrerbildung", a joint initiative of the Federal Government and the Länder which is funded by the Federal Ministry of Education and Research, under Grant 01JA1814. The authors are responsible for the content of this publication.

in a larger context. This includes, in particular, perspectives related to PL (Sect. II-C), which form the transition to RQ2.

A. Terms and Language

In the area of language and terms, three independent strands of research can be identified. First, Diethelm and Goschler [3] proposed four open questions on terms and language in spoken form in conjunction with a call for research in this area. These questions address "the terms (1.), their usage (2.), recommendations for terms (3.), and recommendations for their usage (4.)" [3, p. 23] So far, this call has only been taken up for the perspective of teachers in the field of language [12], [13]. In this article, RQ1 addresses both the terms and their usage, i.e. (1.) and (2.), for *programming novices*. Second, the focus is on computing terms and their use from a broader perspective: the global terminology within the CS community [14] and the terminology of non-computing lecturers outside the CS community [15]. Finally, there is the area of research that focuses on non-native English speakers (NNES) [16], with an emphasis on language [17] and terms [18].

As this article focuses on the terms programming novices use in *their* language, the domain of NNES is left out from further consideration. In summary, the field of terms in CER is rather sparse and there are few publications. In contrast, there is much more work in science education [19] and mathematics education [20].

B. Language-related Theoretical Background

Several theories from different subjects, disciplines, and decades have emphasized the connection between language, thinking and learning in general. Since the 1930s the psychologist Vygotsky has made clear the connection between language and thinking [4]. The same connection is made by the idea of threshold concepts, which CER borrowed from Meyer and Land [21]. The ability to express oneself "in appropriate language" [6] is part of understanding in a particular domain. Linguists have recognized the interconnectedness of language on the one hand and e.g. thinking on the other hand in a broader sense.

Moreover, linguists have referred to the aforementioned appropriateness as a "ticket and visiting card" [7]. If someone is able to express themselves appropriately in the context of a particular community, then they belong to that community. For the participants of presented study, this is the community of computer scientists. Thus, appropriate language is a ticket into the CS community. Moreover, appropriate language is also a visiting card to belonging to the CS community. Both the perspective of the ticket and that of the visiting card belong to the "socio-symbolic function" [7] of language. In addition to this function, according to linguistic theory, language has two other functions, namely "as a medium of knowledge transmission (communicative function)" and "as a tool for thinking (epistemic function)" [7].

The former function is a growing trend in research on classroom language. The latter function is the one already noted by Vygotsky. While Vygotsky has been frequently cited in CER since the 1980s, threshold concepts are an emerging trend. Although the perspective of the linguists has already found its way into mathematics education, e.g. [22], this is not yet the case in CER. In CER, several aspects were considered, namely the classroom language [12] as part of communication and the ticket and NNES barriers [17]. But no connection – to the best of the authors' knowledge – was made from this generalized linguistic perspective. In summary, language-related theories were less frequently translated into language-specific research in the CER. However, the preceding analysis shows that language-specific research is also important for CER and what connections language has.

C. Programming Language Misconceptions and Conceptions

Since the early days of CER, researchers have studied misconceptions associated with PL code snippets [9], [23]. This research trend has carried over to modern PL and led to numerous publications. Recently, Chiodini et al. [8] have presented a continuously updated online inventory that integrates previous research. The previous research cited there can be categorized by the objects studied: Programs (recent examples [24], [25]), statements from interviews (e.g., [11]), statements from explanations of longer codes [10], [26], and statements based on outdated PL [9], [27]. Chiodini et al. focused on analyzing "students' statements about programming, instead [...] students' programs" [8, p. 383].

Moreover, they stated a definition for PL misconceptions: "A programming language misconception is a statement that can be disproved by reasoning entirely based on the syntax and/or semantics of a programming language." [8, p. 383]. This definition was the basis for the definition of a PL conception as presented in the introduction.

The presented article has the same focus on the statements of students as Chiodini et al. [8]. According to the research questions and the previous considerations, the PL misconceptions are related to the terms. On the one hand, to answer RQ1, it is not appropriate to consider programs, since programs do not reveal concepts. On the other hand, descriptions of programs that are longer than code snippets are not appropriate. This follows from the fact that it would not be clear which terms the novices used belonged to which part of the program. Therefore, the approach of statements referring to descriptions of code snippets seems promising. In contrast, interviews have the disadvantage that larger samples are particularly time-consuming and individual components are less transferable to other programming languages and code snippets. It is difficult to assess how relevant results based on BASIC [9] are to modern PL. Therefore, this article fills this gap by analyzing statements from descriptions of code snippets formulated in Java.

III. Methodology

A. Context of the Study

The sample included 123 students enrolled in the CS1 course during the fall 2020/21 semester at the Technical University of Darmstadt in Germany. The active population consisted of 712 students. This number indicates the number of students who turned in the corresponding weekly homework assignment at the beginning of the study. The extent to which this small sample is a threat for validity is discussed in the threat section (V-D). The participants were mostly first semester high school graduates. About 40% had an immigrant background, which is higher than the average of 20% among all German students [28]. Of the participants, 20% were female. The University Ethics Committee did not permit the collection of precise demographic data. No personal data were collected in the survey. Other ethical issues, such as lack of anonymization or underage participants, were not relevant to the survey. Under these circumstances, no approval was required according to the guidelines of the Ethics Committee.

Learning to program in Java is the central topic of the CS1 course. The CS1 course normally consists of 14 weeks, but due to the pandemic in that fall semester, it consisted of 12 weeks. Topics include, among others, the basics of OO, static and dynamic types, error handling, and generics. The course follows the Object-First approach using a modified version of KarelJ [29]. In the first six weeks, programming is done exclusively in Java. The following two weeks were spent teaching the functional concepts that have found their way into Java. The lessons were accompanied by a first look at the functional language Racket for comparison, but Racket was not explored further. They then worked with Java for another four weeks. In total, students were required to individually submit ten code writing homework assignments. Apart from these assignments, students were encouraged to work together. In addition, 30 teaching assistants supported this course. Afterwards, there was an optional programming group project. The final exam was an online written exam.

After six weeks of the course, students participated in the survey. The participation period was nine weeks: three weeks each before, during, and after the Christmas vacations. The relatively long period was due to the fact that more students participated in the survey, which was voluntary. A trade-off was made between the large amount of data desired and the length of participation, with the former chosen as the more important aspect. Participants had a chance to receive one of 20 vouchers as an incentive. Each voucher contained a small amount of money. Students participated anonymously to protect their privacy. At the beginning of the study, participants received a description of the topic, process, duration, and benefits of the study. Participants could only proceed if they gave informed consent.

B. Questionnaire

The case study consists of a one-time online survey with free-text responses, administrated via the dedicated questionnaire software SoSci Survey [30]. A qualitative analysis of what was written was conducted, see Sect. III-C. The content areas used in the self-evaluation instrument (SEI) [31] are the basis for the code snippets; these are “variables and assignment (var), input and output (io), expressions and arithmetic operators (exp), conditional statements (sel), loops and iteration (loops), data collections (lists), functions and methods (funcs), and classes and objects (objs)”. For each of the above content areas, there are one to three different code snippets in the questionnaire, shown in Table I. Two of these code snippets have their source in other articles, namely the variable swap (C03) [32] and the declaration, initialization and assignment of objects and references (C14) [33].

TABLE I
TABULAR SUMMARY OF ID, CONTENT AREA [31], AND CONTENT FOR EACH CODE SNIPPET IN THE QUESTIONNAIRE.

ID	Content area	Content

C01	var	`// a and b are of type int`
		`a = b;`

C02	var	`// k is of type int`
		`k = k + 1;`

C03	var	`// a, b, c are of type int`
		`c = b;`
		`b = a;`
		`a = c;`

C04	io	`System.out.println(k);`

C05	io	`// k is of type int`
		`System.out.println(k);`
		`System.out.println("k");`

C06	exp	`// t, m are of type int`
		`m = t % 60;`

C07	sel	`// k is of type int`
		`if (k == 5){`
		`  // statementblock A`
		`}`
		`// statementblock B`

C08	loops	`while (i < 100){`
		`   // statement block`
		`}`

C09	loops	`for (int j = 0; j < 100; j = j+2){`
		`  // statement block`
		`}`

C10	loops	`for (String s : strings){`
		`   // statement block`
		`}`

C11	funcs	`// a is of type Person`
		`String s = a.toString();`

C12	funcs	`double meanValue`
		`  (List<Integer> data){`
		`      // statement block`
		`}`

C13	objs	`Person a = new Person("Armin");`

C14	objs	`// a, b are of type Person`
		`a = new Person ("Sarah");`
		`b = new Person ("Kim");`
		`b = a;`

C15	objs	`Vehicle m = new Motorcycle();`

In the questionnaire, each code snippet was presented on a separate page. Each page had the same structure: First appeared the prompt “Explain what the following code snippet does.”, then the code snippet, and finally a text entry field. Only the code snippet for variable swap (C03) and input and output (C05) are exceptions. For C03, the prompt was: “Explain what the purpose of the following code snippet is.” For C05, the prompt was: “Explain how the lines of code differ.” The prompts, the comment lines on the code snippet, and the participants’ responses were in German. Participants could enter responses of any length.

Five TAs who supervised the course conducted a trial run before students could participate. In doing so, they completed the questionnaire in its online form as if they were students

TABLE II
EXEMPLAR CODING PROCESS FOR THE CODE SNIPPET C01.

1. Novice Response	**2. Splitting into Code Categories**			**3. Transforming into Resulting Terms**		
	left-hand side of the assignment	**action in infinitive**	**right-hand side of the assignment**	**left-hand side of the assignment**	**action in infinitive**	**right-hand side of the assignment**
"The variable a is assigned the value of variable b." (Student 9)	*The variable a*	*is assigned*	*the value of variable b*	variable a	assign	variable b
"The integer b is copied into the integer a." (Student 50)	*the integer a*	*is copied into*	*integer b*	integer a	copy into	integer b

themselves. They also had the opportunity to provide comments on each page of the questionnaire. This was done through the pretest mode of the SoSci Survey software. The trial run resulted in a change in the wording of the prompt to provide more clarity. Namely, from "Please write down how you would speak the following code." to the previously described prompt. The code snippets remained unchanged. In addition, the time that the TAs took to complete the trial run allowed for more accurate time estimation.

C. Evaluation of Responses and their Terms

An inductive-dominant qualitative content analysis (QCA) [34] was performed to answer RQ1 with these guidelines [35]. As a reminder, RQ1 asked: *What terms do novices use to describe code snippets in Java?* QCA is a valid, widely used approach to qualitative data analysis. In QCA, the process of coding and categorizing is fundamental. The process went as follows: (1) All terms describing the same part of the code belong to the same category and all terms referring to a verb were stored as infinitives. (2) All codes as a result of QCA have been automatically translated by DeepL. DeepL provides the best quality among machine translations for the English-German language pair, which is even more true for the IT domain [36]. To what extent the results are transferable to the other translation direction, i.e., German-English, used in this study remains open. Nevertheless, this seems to be the better machine translation than others. Manual translation was not suitable due to the large number of free-text responses.

Since the entire coding process is beyond the scope of this article, only the coding process of two responses that refer to the same code snippet is given, see Table II. During the process, first, the code categories were formed inductively from an initial reading of the responses. Second, the response to be coded was split along the categories, see the columns labeled "2. Splitting into Code Categories" in Fig. II. Third, the split responses were reduced to the resulting terms, e.g., "is assign" became "assign", and placed in the appropriate code category. This corresponds to the columns shown in Fig. II labeled "3. Transforming into Resulting Terms". In this example, the responses have been translated a priori for better readability, in contrast to the coding process performed.

Both individual terms that resulted from QCA and term groupings were analyzed for frequency. For individual terms, the frequency of a term from the QCA represented the percentage of participants who used that term. These frequencies were presented in four ways: (1) Frequency tables for the top ten terms for each code snippet; (2) Frequency tables for each content area, considering only the terms that appeared in all of a novice's responses to the code snippets for that content area; (3) Total number of unique terms for each code snippet; and (4) Bar charts showing the distribution for each code snippet. For (2), at least two novices had to use the related term.

For the term groupings, n-grams were considered. N-grams are groups of n terms that occur in the same order. In the previous QCA coding, the order of the terms was not considered. Thus, for n-grams, novice responses were directly translated as before. N-grams were computed for each code snippet individually using a generator [37]. Which n-grams were relevant for the analysis was not clear from the outset.

As a reminder, RQ2 asked: *To what extent and in what way do the terms indicate programming language misconceptions or conceptions?*

The answer to this question was obtained through a qualitative analysis of the individual responses and term usage of novice programmers. There is no previous research in the field of novice programmers' use of terms. Therefore, it was not possible to determine in advance which parts of the data set were relevant to answering RQ2. The depth of analysis is shallow, as this work focuses on answering RQ1 in depth.

IV. RESULTS

As a reminder, RQ1 asked: *"What terms do novices use to describe code snippets in Java?"*. In total, participants formulated over 1800 responses. Students took as much time (Mdn=20:17 min) as the announced duration of 20 to 25 minutes. A typical response consisted of 14 words, with no word limit. The word count differed between the code snippets and the content areas, see Table III. Code snippets C09 and C14 stand out for their high word length and high standard deviation. However, a high word count is not necessarily associated with a high standard deviation, see C07. Not surprisingly, the average word count in the variables and input/output content areas is lower than in other content areas. In the latter areas, there are more elements and relationships between the elements of the code snippet to describe.

For the terms typically used, the ten most frequent terms and term groupings are listed (Sect. IV-A). For term diversity, term distributions are considered and sample responses and their terms are analyzed in detail (Sect. IV-B). In addition, participants' responses yielded undescribed misconceptions and conceptions for the purpose of variable swapping and conditional statements (Sect. IV-C).

TABLE III
DISTRIBUTIONS OF WORD LENGTH OF RESPONSES FOR EACH CODE SNIPPET, EACH CONTENT AREA, AND THE ENTIRE DATASET.

Content Area or ID	Min	Max	Mdn	M	SD
Variables	*1*	*76*	*9*	*11.87*	*8.68*
C01	4	40	9	10.14	4.96
C02	1	36	8	9.10	5.03
C03	1	76	13	16.57	12.19
Input/Output	*2*	*36*	*13*	*13.86*	*6.83*
C04	2	28	9	9.59	4.50
C05	3	36	18	18.12	6.06
Expressions	*1*	*49*	*12*	*12.57*	*5.34*
C06	1	49	12	12.57	5.34
Conditional Statements	*9*	*50*	*20.5*	*21.25*	*7.22*
C07	9	50	20.5	21.25	7.22
Loops	*5*	*166*	*14*	*19.27*	*14.28*
C08	6	53	12.5	16.48	9.87
C09	5	166	23	24.87	19.30
C10	7	39	14	16.91	7.40
Functions	*6*	*57*	*15*	*16.34*	*7.97*
C11	6	57	15	16.34	7.68
Objects	*1*	*122*	*17*	*19.37*	*11.59*
C13	1	42	16	16.83	7.04
C14	2	122	23	24.48	15.45
C15	5	57	15	16.77	8.79
Entire Dataset	1	166	14	16.42	10.82

With the exception of C12, the non-response rate was negligible at 1 to 2 %. C12 was removed from the analysis due to some participants filling out the survey before generics were introduced, thus affecting its non-response bias.

A. Terms Novices Typically Used

For the terms typically used, this paper focuses on the ten most frequent terms and the term groupings in the form of n-grams with the highest frequency. These terms are grouped by code snippets, see Tables IV and V, and by content areas, see Table VI. The n-grams are listed in Table VII. In general, the terms used are not surprising, as indicated by the fact that no unusual or new terms appear in these tables.

The ten most frequent terms include names, technical terms, and natural language with mostly compound terms. Names such as “a” for a variable or “Armin” for a string are the most frequent terms, with the exception of code snippets C09 through C11.

Apart from the names, most of the terms come from technical language and not from natural language. This means that, by and large, the participants are able to use the appropriate technical language in the appropriate places. The technical language used is now clarified using the content area variables and objects. For variables, these are the commonly used terms “variable” and “value” with the associated activities “assign,” “increase,” “increment,” and “swap”. For objects, these are the commonly used terms “object,” “class,” and “type” with the associated activity “assign”,” as well as the used classes from the code snippets like “motorcycle” and “person”.

The technical language, in turn, differs from the keywords and syntax of the programming language. For example, the term “loop” from the technical language in Java has the keywords `for` and `while`. In contrast to the technical language, however, only one keyword appears among all the ten most frequent terms, namely “int” (C01).

The terms typically used include four common verbs, namely “assign,” “execute,” “output,” and “create.” Each code snippet contains exactly one of these verbs among the ten most frequent terms. Thus, the code snippets can be grouped according to the use of the verb: “assign” (C01, C02, C03, C06, C11, C14), “execute” (C07, C08, C09, C10), “output” (C04, C05), “create” (C14, C15). This grouping among verb terms does not agree with the grouping that results from content areas as the traditional grouping.

The terms for the content areas themselves, such as “variable” for the content area variables, come to the fore. Only for some content areas does the associated term appear among the first ten terms. These are (1) “variable” for code snippets C01 and C02 and “variables” for C03; (2) “output” for C04 and C05, (3) “object” for C13 to C15. In contrast, the term “loop” does not appear among the top ten terms in any of the three code snippets in the content area loops, C08 to C10.

Of the four common verbs, the verb “create” is of particular interest. In German, there is a counterpart for the technical term “ instantiate”, namely “instanziieren.” However, in C13, the 3.3% of novices using “instantiate” is insignificant compared to the 65.9% of novices using “create” to describe the creation of the object. Thus, in this case, novices preferred a natural language term. Other terms that originate in natural language are compound terms that combine other terms. Examples are “of,” “to,” “from,” “as,” and “by”.

Next, the terms that appear in all code snippets of a content area are considered, see table VI. The content areas of most interest here are those that consist of more than one code snippet. These are not marked with an asterisk. These content areas differ in several ways: The total number of terms is three for variables, twenty for input/output (with only the top ten listed), four for loops, and nine for objects. The number of terms with a frequency of less than 10% are one for variables, three for input/output, two for loops, and four for objects. The term “variable”, representing the content area itself, appears in only 4.9% of all descriptions of code snippets. A similar picture emerges for loops, where the term “loop” appears only in 7.3%. In contrast, the term “output” appears in 36.6% and the term “object” in 38.2% of all descriptions of code snippets. Thus, the similarity of terms between code snippets is low for variables and loops and high for input/output and objects.

In what follows, the focus is on the analysis of n-grams. Among n-grams, 3-grams and 5-grams have been found to be rich in quality. These are listed in Table VII. Eight of the total fourteen 3-grams are a subset of the corresponding 5-gram, indicated by italics. Thus, 3-grams and 5-grams partially share terms and the order of terms. In addition, 3-grams and the top ten terms share similarities. All terms of 3-grams, except for articles such as “a”, “an”, and “the”, are part of the top ten terms. Some code snippets also have the same 3-gram, namely “the value of” for C01, C04, and C05. The 3-grams of C07 and C09 are not identical but similar, as are the 5-grams of C07 and C10. All of these results suggest a general homogeneity of the terms typically used. However, the frequency of 3-grams and 5-grams differs between code snippets. Code snippet C07 has the highest frequency for n-grams, 71.5% for the 3-gram and 43.9% for the 5-gram. The lowest frequencies are 17.11%

TABLE IV
THE TEN MOST FREQUENT TERMS FOR EACH CODE SNIPPET FROM C01 TO C07, WITH THEIR FREQUENCY (%). TERMS WITH THE SAME FREQUENCY ARE LISTED TOGETHER, SEPARATED BY A SLASH (/).

	C01		C02		C03		C04		C05		C06		C07	
Nr.	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**
1	b	97.6	k	98.4	a	74.0	k	95.1	k	94.3	m	93.5	A	93.5
2	a	96.7	by	79.7	b	68.3	console	61.0	output	72.4	t	83.7	B/k	92.7
3	value	82.9	value	44.7	of	61.0	value	53.7	value	71.5	of	66.7	execute	88.6
4	of	69.1	increase	40.7	c	48.8	output	48.8	of	63.4	assign	50.4	if	82.9
5	assign	66.7	increment	36.6	swap	46.3	of	48.0	first	54.5	modulo	46.3	block	81.3
6	variable	39.0	of	33.3	values	45.5	print	38.2	string	48.8	value	45.5	statement	61.8
7	from/to	9.8	one	28.5	value	43.1	in	35.8	second	47.2	remainder	41.3	then	57.7
8	copy	8.1	variable	26.8	variables	22.8	on	27.6	line	44.7	variable	31.7	value	55.3
9	integer/set/store	7.3	assign	13.0	assign	19.5	line	17.9	letter	40.7	by	27.6	be/equal	39.0
10	int	6.5	add	10.6	to	13.8	to	15.4	variable	23.6	division	22.0	otherwise	27.6

TABLE V
THE TEN MOST FREQUENT TERMS FOR EACH CODE SNIPPET FROM C08 TO C15, WITH THEIR FREQUENCY (%). TERMS WITH THE SAME FREQUENCY ARE LISTED TOGETHER, SEPARATED BY A SLASH (/).

	C08		C09		C10		C11		C13		C14		C15	
Nr.	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**
1	i	91.9	block	80.5	block	68.3	string	89.4	person	92.7	b	95.9	motorcycle	91.9
2	block	87.0	execute	77.2	statement	65.0	a	82.1	Armin	84.6	a	91.1	vehicle	87.0
3	execute	83.7	j	74.0	strings	63.4	of	52.0	object	78.9	person	68.3	object	77.2
4	than	75.6	statement	65.9	each	60.2	to	49.6	a	74.0	Sarah	59.3	m	74.0
5	as	69.9	by	54.5	execute	59.3	toString	47.2	of	70.7	object	54.5	of	65.9
6	less/long	68.3	increase	42.3	for	56.1	method	43.9	create	65.9	assign	52.8	type	56.9
7	its	64.2	times	41.5	string	48.8	object	36.6	new	51.2	of	49.6	create	51.2
8	be	59.3	than	38.2	in	47.2	assign	32.5	class	43.9	Kim/to	45.5	new	43.9
9	statement	56.1	as/be	36.6	s	44.7	person/value	30.9	string	39.0	with	39.0	class	43.9
10	value	30.9	long	31.7	element	24.4	type	28.5	type	37.4	two	38.2	reference	30.1

TABLE VI
THE MOST FREQUENT TERMS FOR EACH CONTENT AREA APPEARING IN EACH RESPONSE TO THE ASSOCIATED CODE SNIPPETS, WITH THEIR FREQUENCY (%). AT LEAST TWO NOVICES HAD TO USE THIS TERM. CONTENT AREAS MARKED WITH AN ASTERISK (*) CONTAIN ONLY ONE CODE SNIPPET. TERMS WITH THE SAME FREQUENCY ARE LISTED TOGETHER, SEPARATED BY A SLASH (/).

	Variables		Input/Output		Expressions*		Conditional Statements*		Loops		Functions*		Objects	
Nr.	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**	**Term**	**%**
1	value	22.8	k	86.2	m	91.1	A	93.5	block	53.7	string/a	80.5	object	38.2
2	of	17.1	output	36.6	t	80.5	B/k	92.7	statement	28.5	s	79.7	of	35.8
3	variable	4.9	of	39.0	of	66.7	execute	88.6	loop	7.3	of	52.0	class	13.8
4			value	36.6	assign	50.4	if	82.9	execute/of	2.4	to	48.8	new	13.0
5			console	22.0	modulo	46.3	block	81.3			toString	47.2	reference/type	11.4
6			in	13.0	value	45.5	statement	61.8			method	43.9	create	9.8
7			variable	11.4	remainder	41.3	then	57.5			object	36.6	variable	4.1
8			line/on	6.5	variable	31.7	value	55.3			assign	32.5	assign	3.3
9			string	4.9	by	27.6	be/equal	39.0			person	30.9	with	1.6
10			store	4.1	division	21.9	otherwise	27.6			value	30.1		

(C11) for the 3-gram and 6.5% (C14, C15) for the 5-gram.

In summary, the terms and term groupings presented in the frequency tables, together with the preceding analysis, provide an answer to RQ1. In this paper, not all terms are listed, but the typical terms with their frequency. More details on the terms used can be found in the following Sect. IV-B.

B. Term Diversity

For term diversity, term distributions and sample responses from novices are considered.

First, the term distributions for each code snippet are shown as bar charts in Fig. 1. Each bar represents a term, and the height of the bar represents the frequency of that term on a percentage scale. The narrower the bars, the more different terms the novices used for this code snippet.

The distributions as a whole are characterized by a high degree of homogeneity. This is evident from the general appearance of the distributions, namely the similarity to an exponential distribution. All bar charts have (1) a peak with a few terms with high frequency and (2) some terms with medium frequency. In addition, (3) for all bar charts, the height of most terms is barely distinguishable from zero. This means that most terms were used by at most a handful of novices. It would also have been conceivable that the bar charts consisted of many more different terms, with no term being mentioned more often than the others. For the bar charts, this would mean that all bars have a similar height and there is no steep flattening.

However, the term distributions differ with respect to the total number of terms, see caption in Fig. 1. The number of

TABLE VII
TABULAR SUMMARY OF THE ID OF THE CODE SNIPPET AND THE MOST FREQUENT N-GRAMS FOR N=3, N=5 WITH THEIR FREQUENCIES (%).

ID	3-gram	%	5-gram	%
C01	the value of	70.7	is assigned *the value of*	19.5
C02	is increased by	26.8	k *is increased by* 1	15.4
C03	a and b	33.3	*a and b* are swapped	15.4
C04	the value of	37.4	*the value of* k is	16.2
C05	the value of	50.4	*the value of* k is	15.4
C06	t modulo 60	48.8	m is assigned the value	17.1
C07	A is executed	71.5	statement block *A is executed*	43.9
C08	as long as	32.5	i is less than 100	23.6
C09	block is executed	21.1	j is less than 100	12.2
C10	the statement block	57.7	*the statement block* is executed	26.0
C11	the method toString	17.1	the return value of the	8.1
C13	a new object	35.8	*a new object* of the	16.2
C14	a and b	35.0	objects of the class Person	6.5
C15	an object of	22.0	object of the class Motorcycle	6.5

terms ranges from 40 (C02) to 168 (C10). The distributions flatten less for the code snippets where the most frequent term has a frequency of 85%. This is true for code snippets C03, C09, and C10. This means that a greater variety of terms is used in these code excerpts.

A possible explanation for this result is that the novices use different approaches to describe the corresponding code snippets. For C03, novices either described the purpose of the three-line swap with using terms similar to "swap values of a and b". Or they described the assignment of the variables separately and used terms like "assign the value of the variable b to the variable c". Both approaches do not share common terms, but all terms are apparent among the ten most frequent used terms. For C09, some novices only described the number of iterations with used terms similar to "the statement block is executed 50 times". Whereas other novices described most or all elements of the loop header, but did not mention the number of iterations. In contrast, the responses for C10 are more varied and no clear-cut approaches were found, as in the previous code snippets.

The conclusion from the distributions is that the ten most frequent terms provide good coverage of all terms. Thus, they provide a sufficient answer to RQ1.

Second, some examples of terms and responses are presented to illustrate the diversity.

1) Example 1 – Conditional Statements: In this example, the term diversity of code snippet C07 is analyzed because the degree of diversity among terms varies. First, the terms for the keyword `if` in the second line of code are considered. In C07, 78.0% of participants used the term "if." The remaining terms are evenly distributed over: "whether," "whether if," "in case," "only if," "whether if yes," "provided," "as long as," and "should," in descending order of frequency. Thus, there is little variety among the nine different terms for the keyword `if`. However, the frequency tables counted single terms. The compound terms containing the term "if" were already included in the frequency of the term "if". Therefore, the reported frequency of 82.9% is higher than the frequency of 78.0% of using only the term "if". If one merges all the previous terms, except "as long as", into the single term "if" as a representative for these terms. Then the frequency of the term "if" increases from 78.0% to 94.3%, making it the most frequently occurring term in C07.

In contrast, there are 46 terms describing the execution of statement block B, which refers to the last line of the code snippet. This higher number, i.e., the comparison of 46 to 9 different terms, indicates a greater diversity among the terms. Among the 46 terms, the five most common terms are: "otherwise" (15.4%), "only" (8.9%), "in any case" (8.1%), "independent" (5.7%), and "always" (6.5%). The low frequencies show that there is no salient term as in the case of the keyword `if`. Participants also chose combinations that yielded a large number of terms, e.g., "otherwise skip directly." In the previous listing, the combined terms were not counted, but listed individually. In the frequency tables, the terms were counted individually, which explains the higher frequency. However, the diversity described here in the description of the following statement block after a conditional statement does not mean that there are no clusters. These clusters are described in Sect. IV-C, which is about RQ2, i.e., how terms reveal PL misconceptions and conceptions.

TABLE VIII
TABLE WITH EXAMPLES OF STUDENT RESPONSES FOR THE PURPOSE OF VARIABLE SWAPPING (C03)

Student	Response for the Purpose of the Variable Swap (C03)
3	It is said that a, b, c have the same value
13	c, b and a are assigned the value b,a and b respectively.
17	The values a and b are exchanged by means of the auxiliary variable c.
18	swap triangle
48	a = a. The variable a is in the end equal to a
49	assignment
90	This code swaps the contents of variables a and b using a temporary variable c.
118	The values of the variables are rotated through/moved up? At the end, a as well as c have the starting value of b. The purpose is not clear to me from this section alone.

2) Example 2 – Variable Swap: Participants' terms and their connection between terms vary widely for the code snippet of the variable swap (C03). As a reminder, participants were asked to describe the *purpose* of this code snippet. Table VIII shows some examples of related student responses. Previous studies have discussed the difficulty of this task for novice programmers, e.g. [32], [38], [39]. This study is no exception in this regard, as the following examples show.

Student 3 incorrectly described the values of the variables. Both student 13 and student 118 explained the three assignments in their excerpts, but in different ways. They did not formulate a purpose for the three assignments. Student 17 understood the purpose of the variable swap. However, the term "values a and b" is not fully correct, since a and b are variables and not values. Student 48 has an assignment PL misconception. They have applied their mathematical experience with the character = to the code. Student 49 has shown only that they understood that the character = stands for the term assignment. Student 90 correctly described the purpose. However, student 118 was aware that there was a purpose for

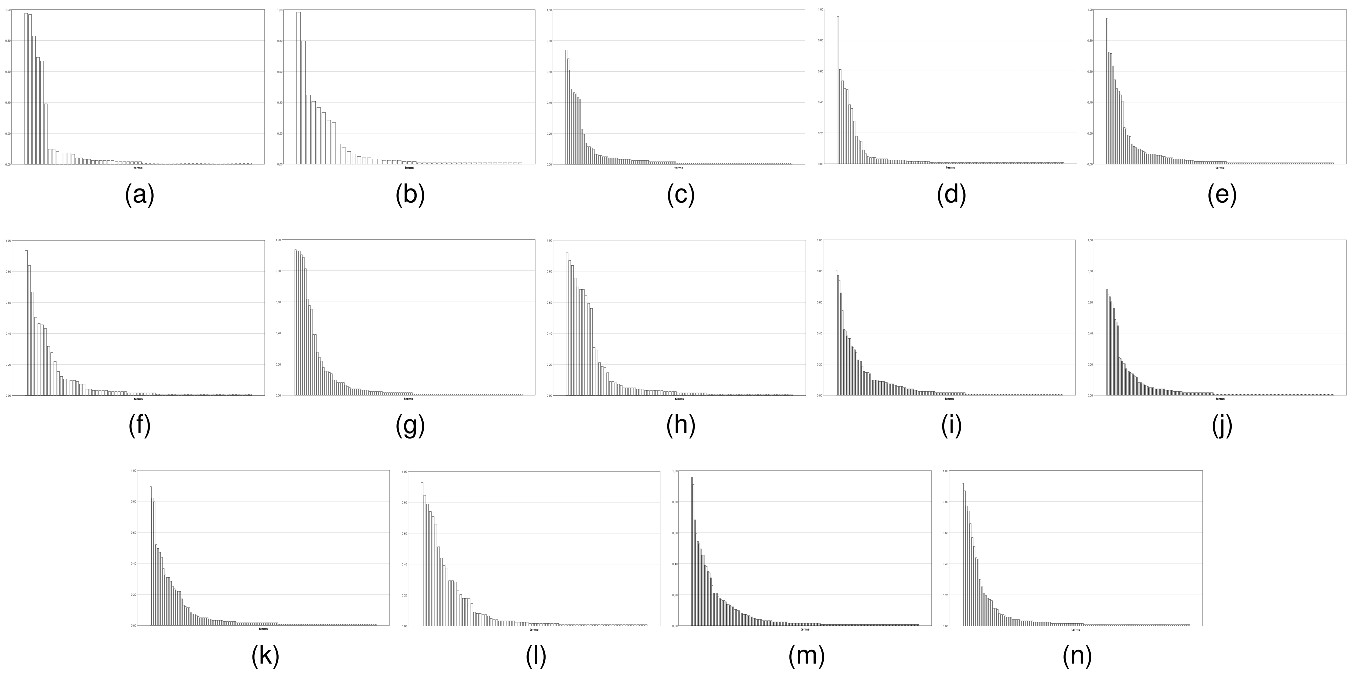


Fig. 1. Distributions of terms for each code snippet, followed by total number of terms. On the x-axis are the terms and on the y-axis their frequency between 0 and 1. (a) C01, 56. (b) C02, 40. (c) C03, 117. (d) C04, 98. (e) C05, 119. (f) C06, 71. (g) C07, 114. (h) C08, 83. (i) C09, 149. (j) C10, 168. (k) C11, 124. (l) C13, 82. (m) C14, 155. (n) C15, 118.

the assignments, but did not describe the value of variable b after the three lines of code.

Students 3 and 18 are not isolated cases, but are representative of two groups within the 28.5% of participants who described a false purpose. The first cluster is *swap*, consisting of 13.0% of participants with the terms "swap triangle," "swap," "swap b and c," "swap a and c." The second cluster is *same value*, consisting of 7.3% of participants with the terms "same value," "same value a," "same value b." Within the description of false purpose, the remaining 8.1% cannot be grouped together nor do any of the terms occur more frequently than two novices. These include, for example, "shift", "cache", or "assign". In addition, Student 13 is representative of both the 28.5% of participants who did not describe the required purpose and the subset of 11.4% of students who explicitly stated the value of all three variables.

In relation to the research questions, the following results emerge: (RQ1) Participants use correct terms (e.g., swap) and incorrect terms (e.g., same value) or terms that describe the process but are not appropriate for abstraction, such as the term "assign" in Student 17's response. (RQ2) Some of the terms directly suggest a PL misconception, such as "same value," but others, such as "swap," are contextual. Therefore, how participants use the term "swap" is critical in determining whether they are conveying a PL misconception or a conception. As with the previous example, the relationship to PL misconception and conceptions is discussed in more detail in Sect. IV-C.

3) Example 3 – References and Aliasing: This last example shows how much the participants' answers vary in length, comprehensiveness, and precise use of technical terms. The related code snippet C14 is about references and aliasing of objects in Java. Table IX lists two student responses for C14, each representing the full student description of this code snippet.

Student 27 summarized the declaration and initialization in lines 1 and 2 of the code in their first sentence. For the initialization, they used the term "are created" for what is done during initialization and "objects of the class Person" for what is created. Student 27 also distinguished between objects and references as separate terms, indicated by the text fragment "to each of which [referring to the objects] a reference [...] points." More precisely, the term should read "reference of *type* Person", not "reference of the class Person" as used by student 27. Student 27 described the assignment in the first part of the second sentence. Then this student explained the consequences of the assignment: both references refer to a single object, the lack of addressability of the other, previously created object. Student 27 did not mention the attributes of the object indicated by the lack of use of related terms.

In contrast, Student 30's response is much shorter because the description covers only the consequences of the third line of code. The declaration and initialization of the references *a* and *b* do not occur. The only terms Student 30 used are the attributes as literal, i.e. "Sarah" and "Kim", and the term "is overwritten". Thus, the string attributes are used to represent the entire object, which is equivalent to an "identity/attribute confusion" [40]. This is an example, how terms are able to indicate PL misconception as stated in RQ2.

The comprehensiveness of the responses of these two students can be deduced by attempting to reconstruct the code backwards from the description. This is largely possible for

TABLE IX
TWO STUDENT EXCERPTS ON DECLARING, INITIALIZING REFERENCES, AND REFERENCING OBJECTS (C14)

Student	Response for the Object Assigment (C14)
27	Two objects of the class Person are created, to each of which a reference of the class Person points. Then b is assigned the value of a, which is why both references now point to the object referenced by a and the other person is no longer addressable.
30	"Sarah" is overwritten with "Kim".

Student 27, but not for Student 30. For conclusion, these two responses show that the answer to RQ1 (What terms do novices use to describe code snippets in Java?) depends on the individual novices. At the same time, it can be said that the analysis of the terms actually points to PL misconceptions and conceptions as a response to RQ2.

C. Terms Revealing PL Misconceptions and PL Conceptions

In the previous examples (see Sect. IV-B), the transition from terms to PL misconceptions and PL conceptions has already been touched upon. Now the answer to RQ2 will be discussed in more detail. Two aspects should be recalled: The definitions of PL misconceptions and PL conceptions are given in Sect. II-C. This work focuses on two exemplary code snippets and therefore the answer to RQ2 will be rather shallow.

Attention now turns to code snippet C07, which belongs to the content area of conditional statements. There the terms can be grouped according to the PL misconception or PL conception to which they belong. These are one PL misconception and three different groups of PL conceptions. The PL misconception is that novices convey the following statement by using terms: “The following statement block is executed only if the Boolean value of the conditional statement is false.” Terms that indicate this PL misconception are “else,” “in the other case,” and “otherwise”. In total, 34 of the 123 participants belong to this group, which considers the following statement block as its *counterpart*. Here, the responses would be appropriate if the code consisted of an if-else structure. However, in C07, there is no else part.

In the first grouping of the PL conceptions, novices consider the following statement block as a *separate entity*. This means that regardless of whether the condition is true or false, the following statement block is executed independently. Terms that indicate this PL conception are “independent,” “always,” and “in any case”. In total, 30 of the 123 participants belong to this group. In the second group, novices describe the following statement block as *later in time*. Thus, the execution of the following statement block takes place after the possible execution of the conditional statement block. Terms that indicate this PL conception are “following,” “after,” and “then”. In total, 11 of the 123 participants belong to this group. In the last group, the PL conception consists of *skipping* the conditional statement block. This is equivalent to the following statement: “If the condition is false, the conditional statement block is skipped.” Terms that indicate this PL conception are “skip,” “ignore,” “if directly,“ and “leave out”. In total, 13 of the 123 participants belong to this group.

The responses belonging to the first group of PL conceptions and the PL misconceptions both occurred in about a quarter of all the participants’ responses. Thus, they are both important elements to consider when dealing with conditional statements, since together they account for half of the participants’ responses. The second and third groups of PL conceptions occur more frequently than sporadic ones, namely in one tenth of all responses, but these PL conceptions play a much smaller role.

V. DISCUSSION

A. Answers towards the Research Questions

The first research question was: *What terms do novices use to describe code snippets in Java?*

The answer to this question consists in the direct sense of the frequency tables of the ten most frequent terms and term groupings (see Sect. IV-A) and the example answers (see Sect. IV-B), since in these the terms used by the novices are represented. The distributions show that focusing on the ten most frequent terms is indeed the answer to RQ1. The frequencies of the terms quickly flatten out and overall resemble an exponential distribution. Thus, the terms that do not appear among the ten most frequent were mostly used by only a few novices. Therefore, the ten most frequent terms are salient for describing the associated code snippet and the programming concepts it contains. Moreover, not all code snippets of one content areas were described using similar terms. In the findings, the use of verb terms that novices typically used in code snippets stood out. The new grouping among the terms “assign,” “execute,” “output,” and “create” did not agree with the grouping that resulted from the content areas. Thus, the answer to the question of which terms novices use can lead one to question the traditional groupings in terms of content areas. To what extent is it useful in class to show similarities between traditionally different code snippets by using the same or similar terms? In general, there is a homogeneity of terms, but at the individual level there is a wide range both in the terms themselves and in the number of terms used.

The second research question was: *To what extent and in what way do the terms indicate programming language misconceptions or conceptions?*

This work has shown superficially, with two examples, that terms have the possibility of revealing novices’ PL misconceptions and conceptions. However, this work does not aim to generalize to other examples, nor is it so simple that the terms always reveal the PL misconceptions and conceptions of the novice. Rather, the analysis of the terms can be as a possible heuriusmus for discovery, but not necessarily leading to the goal. This will be explained with the example of variable swap. In the analysis of the variable swap (C03), it has been shown that there are terms that directly indicate PL misconceptions, such as “same value”, and others that are contextual, such as “swap”. However, the terms used by novices point to PL misconceptions and conceptions. This was also shown by means of the concepts conditional statements and references (see Sect. IV-B, IV-C). In contrast, it is also possible for novices to use certain terms as shortcut. In this case, these novices would not show misconceptions in a follow-up study during the interview. No follow-up study was conducted for this case study. Therefore, the claimed connection between terms and conceptions remains weak. As as results, practitioners – such as teachers, instructors, and teaching assistants – cannot look to terms alone to make appropriate performance diagnoses. The context and interplay of how terms are used are equally important. Nonetheless, by analyzing the terms, previously undescribed PL misconceptions and conceptions can be described (see Sect. IV-C). Moreover, it may be fruitful to consider the novices' terms in one's own teaching. More detailed and further implications for practitioners are discussed in Sect. V-C.

B. *Significance of Findings and Strengths of the Study*

Terms are key factors in communication, both in daily life and in academia. In a CS1 course as part of academic computer science, instructors aim to teach novices how to program. Therefore, the terms of novices are important both for instructors in designing their courses and for researchers in developing evidence-based teaching methods. In summary, the answers to the research questions proposed in this article are relevant to CER.

Moreover, this study is – to the best of the authors' knowledge – the first answer to the open questions of Diethelm and Goschler [3] regarding the terms and their usage of novices in their natural language.

Last, the strengths of this study are presented. The data set is relatively large, with more than 1,800 responses. Moreover, the analysis of the terms was done directly on them and not on more or less related learning artifacts. Therefore, this study provides direct access to the terms and presents some of them directly with examples (see Sect. IV-B).

C. *Recommendations for Teaching in CS1*

Even though this is only a case study, some recommendations for teaching CS1 with Java can already be given.

The authors see a possible application for everyday practice of using the questionnaire with the listed prompt as a *diagnostic tool*. This is based on the connection that language is a "tool of thinking" [7] and that language and thinking are connected [4]. This connection was described in Sect. II-B. Thus, the diagnosis of language provides clues about students' thinking. On the one hand, the questionnaire can serve as a formal assessment, but also as a self-assessment for the students. This practical application was also addressed by participants in the study. Below are three excerpts from the free-text field for criticism and comments from the questionnaire: (a) "The survey was useful so that I can see for myself where I stand right now." (b) "Thank you, it made me feel like I already understood a little bit" (c) "Showed some gaps for me, thanks for that!" From the excerpts, it appears that the self-assessment aspect consists of three parts: (1) knowing where you are right now; (2) seeing what you can already do; and (3) seeing what you cannot do yet and identifying those gaps.

The appropriateness of the terms cannot be determined across the board. Therefore, the authors did not evaluate the terms found in the word clouds for appropriateness. Rather, the terms were examined exemplarily for their use in the respective context. For example, in the last paragraph of Example 2 – Variable Swap, it was shown that some terms directly indicate a PL misconception, while others are context dependent. A follow-up study would be necessary to determine which terms are still considered appropriate by different instructors from different countries and contexts and which are no longer appropriate. However, this exploratory study cannot do this.

What this study can do, however, is present examples of terms and descriptions for code snippets. Teachers can use these to engage in conversation with their students. For example, the responses for the purpose of the variable swap (see Table VIII) could be presented to students along with the code snippet. In groups or in plenary, students should discuss which terms are appropriate and which terms make them appropriate or inappropriate. In this way, students learn not only the correct conceptions, but also how to describe them appropriately. On the other hand, students benefit by building their negative knowledge. Negative knowledge is based on the assumption that "to know what is wrong helps in understanding what is right" [41] and has already found its way into educational research.

In addition, teachers can compare their own descriptions of code snippets in class with the terms and term groupings presented in this work. This could lead to several teaching scenarios: (1) The teacher proactively considers the novice terms and highlights which parts might be appropriate and which do not fit the learning objectives. (2) This could also lead to jointly created best practice description during instructional sessions. (3) The teacher can review their own use of terms in the instructions and determine if they are unintentionally using shorthand terms.

D. *Limitations and Threats to Validity*

There are both limitation of this study and several threats that warrant discussion.

To begin with the *limitations*: This study is a *case study*. It is the nature of case studies that they are less amenable to generalization. This is also the case with this study. Generalization to all Java novices in other CS1 courses is not guaranteed. The results on the terms depend on the context, i.e., the course, the participants, and the code snippets of the questionnaire. If only one of the three parameters is changed, most likely the terms will change as well. Therefore, this study is only a first look at the answers to the open questions of Diethelm and Goschler [3] regarding the terms and their usage.

Second, the *internal threats* are listed. The maturation effect may have occurred: The period of study participation was relatively long, nine weeks. Students could participate before, during, and after winter break. Students who participated later might have improved their skills. Their maturation might have resulted in a higher rate of appropriate responses. The long period of study participation was chosen because otherwise the sample size would have been about half as large, i.e., n=60, for a duration of one week. When weighing more data against the maturation effect, the former was chosen. Possible reasons for the low number of participants are: (1) the duration of study participation was relatively long, 25 to 30 minutes; (2) students focused only on necessities due to the Corona pandemic, so additional activities such as study participation fell short.

In addition, despite efforts to be objective, inductive-dominant qualitative content analysis (QCA) is influenced by the researcher who conducted the QCA. Thus, another researcher could have created different categories for the terms used in the responses. There may also have been translation issues as participants' responses were translated from German to English for presentation in this article. These issues were addressed by using established translation software such as DeepL to reduce individual translation issues, see Sect. III-C.

Last, the *external threats* are listed. No demographic data could be collected. Thus, the sample demographics may not be representative. Good students with prior knowledge of Java would provide a baseline, i.e., what results these students would get when describing code snippets assuming they have no programming problems. In general, it is very likely that students with different levels of Java knowledge participated, even if they took the same CS1 course. This knowledge most likely influenced the students' ability to describe the code snippets in Java. All students were from the same university in Germany and these participants were only a small subset of all students in the CS1 course. Therefore, the results are difficult to generalize to the entire CS1 population in this course. The small subset is mainly due to the fact that participation in the study was voluntary. The research guidelines of the authors' university require that participants in empirical studies explicitly and understandably agree that they are voluntarily participating in our research. In accordance with these guidelines, there is no other option than to offer participation in the study as voluntary.

VI. Conclusion

Language and reasoning are interrelated [4], and language also fulfills other functions like for communication and belonging [7]. A *term* includes a word or expression with a precise meaning and is part of the language. The call for the study of terms proposed by Diethelm and Goschler [3] has so far received little attention, especially in the field of novices and terms in their natural language.

Thus, this article focuses on terms and the connection towards PL misconceptions and conceptions, leading to the following research questions: (RQ1) What terms do novices use to describe code snippets in Java? (RQ2) To what extent and in what way do the terms indicate programming language misconceptions or conceptions?

The results to the research questions are based on more than 1,800 responses from n=123 participants in a German CS1 course. As for RQ1, the results on the terms were presented by frequency tables for the ten most frequent terms, n-grams, distribution plots, and by analysis of the sample responses. On the one hand, the terms used by novices are generally homogeneous, as shown by the fact that the distributions quickly flatten out. On the other hand, the individual responses show great diversity. As for RQ2, the results show that the terms can serve as a possible heuriusmus for discovery of conceptions, but do not necessarily lead to the goal. Moreover, some terms are contextual, so that without context it is not possible to decide whether a conception is present or not. Therefore, there is no causality between terms and conceptions.

In the future, the authors would like to apply the questionnaire to courses outside Germany.

References

[1] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, p. 378–389. [Online]. Available: https://doi.org/10.1145/2568225.2568252

[2] S. R. Portnoff, "The introductory computer programming course is first and foremost a language course," *ACM Inroads*, vol. 9, no. 2, p. 34–52, apr 2018.

[3] I. Diethelm and J. Goschler, "Questions on spoken language and terminology for teaching computer science," in *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '15. New York, NY, USA: ACM, 2015, p. 21–26. [Online]. Available: https://doi.org/10.1145/2729094.2742600

[4] J. V. Wertsch, "Dialogue and dialogism in a socio-cultural approach to mind," in *The dynamics of dialogue*, K. Marková, Ivana; Foppa, Ed. New York, London: Harvester Wheatsheaf, 1990, pp. 62–82.

[5] J. Rountree and N. Rountree, "Issues regarding threshold concepts in computer science," in *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95*. AUS: Australian Computer Society, Inc., 2009, p. 139–146.

[6] J. H. Meyer, "Threshold concepts and pedagogic representation," *Education+ Training*, vol. 58, no. 4, pp. 463–475, 2016.

[7] V. Heller and M. Morek, "Academic discourse as situated practice: An introduction," *Linguistics and Education*, vol. 31, pp. 174–186, 09 2015.

[8] L. Chiodini, I. Moreno Santos, A. Gallidabino, A. Tafliovich, A. L. Santos, and M. Hauswirth, "A curated inventory of programming language misconceptions," in *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*, ser. ITiCSE '21. New York, NY, USA: ACM, 2021, p. 380–386. [Online]. Available: https://doi.org/10.1145/3430665.3456343

[9] P. Bayman and R. E. Mayer, "A diagnosis of beginning programmers' misconceptions of basic programming statements," *Commun. ACM*, vol. 26, no. 9, p. 677–679, Sep. 1983. [Online]. Available: https://doi.org/10.1145/358172.358408

[10] J. Sorva, "Students' understandings of storing objects," in *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88*, ser. Koli Calling '07. AUS: Australian Computer Society, Inc., 2007, p. 127–135. [Online]. Available: https://doi.org/10.5555/2449323.2449337

[11] L. C. Kaczmarczyk, E. R. Petrick, J. P. East, and G. L. Herman, "Identifying student misconceptions of programming," in *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2010, p. 107–111.

[12] S. Sentance and J. Waite, "Teachers' perspectives on talk in the programming classroom : Language as a mediator," in *Proceedings of the 17th ACM Conference on International Computing Education Research*, ser. ICER 2021. New York, NY, USA: ACM, 2021, p. 266–280. [Online]. Available: https://doi.org/10.1145/3446871.3469751

[13] B. A. Becker, D. Gallagher, P. Denny, J. Prather, C. Gostomski, K. Norris, and G. Powell, "From the horse's mouth: The words we use to teach diverse student groups across three continents," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2022. New York, NY, USA: ACM, 2022, p. 71–77. [Online]. Available: https://doi.org/10.1145/3478431.3499392

[14] Simon, A. Clear, J. Carter, G. Cross, A. Radenski, L. Tudor, and E. Tõnisson, "What's in a name? international interpretations of computing education terminology," in *Proceedings of the 2015 ITiCSE on Working Group Reports*, ser. ITICSE-WGR '15. New York, NY, USA: ACM, 2015, p. 173–186. [Online]. Available: https://doi.org/10.1145/2858796.2858803

[15] B. Becker, *The Roles and Challenges of Computing Terminology in Non-Computing Disciplines*. New York, NY, USA: ACM, 2021. [Online]. Available: https://doi.org/10.1145/3481282.3481284

[16] Y. Lei and M. Allen, "English language learners in computer science education: A scoping review," in *Proceedings of the 53rd ACM Technical Symposium on Computer Science Education V. 1*, ser. SIGCSE 2022. New York, NY, USA: ACM, 2022, p. 57–63. [Online]. Available: https://doi.org/10.1145/3478431.3499299

[17] B. A. Becker, "Parlez-vous java? bonjour la monde != hello world: Barriers to programming language acquisition for non-native english speakers," in *30th Workshop of the Psychology of Programming Interest Group - PPIG '19*, 2019. [Online]. Available: www.brettbecker.com/publications

[18] S. Alaofi and S. Russell, "A validated computer terminology test for predicting non-native english-speaking cs1 students' academic performance," in *Australasian Computing Education Conference*, ser. ACE '22. New York, NY, USA: ACM, 2022, p. 133–142. [Online]. Available: https://doi.org/10.1145/3511861.3511876

[19] C. Mönch and S. Markic, "Science teachers' pedagogical scientific language knowledge—a systematic review," *Education Sciences*, vol. 12, no. 7, p. 497, 2022. [Online]. Available: https://doi.org/10.3390/educsci12070497

[20] K. Erath, J. Ingram, J. Moschkovich, and S. Prediger, "Designing and enacting instruction that enhances language for mathematics learning: a review of the state of development and research," *ZDM Mathematics Education*, vol. 53, pp. 245–262, 2021.

[21] J. Meyer and R. Land, "Threshold concepts and troublesome knowledge: Linkages to ways of thinking and practising within the disciplines," in *Improving Student Learning: Theory and Practice - 10 Years on*, C. Rust, Ed. Oxford Brookes University, 2003, pp. 412–424.

[22] K. Erath, S. Prediger, U. Quasthoff, and V. Heller, "Discourse competence as important part of academic language proficiency in mathematics classrooms: The case of explaining to learn and learning to explain," *Educational Studies in Mathematics*, vol. 99, no. 2, pp. 161–179, 2018. [Online]. Available: https://doi.org/10.1007/s10649-018-9830-7

[23] B. Du Boulay, "Some difficulties of learning to program," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 57–73, 1986.

[24] A. Altadmri and N. C. Brown, "37 million compilations: Investigating novice programming mistakes in large-scale student data," in *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: ACM, 2015, p. 522–527. [Online]. Available: https://doi.org/10.1145/2676723.2677258

[25] R. Caceffo, P. Frank-Bolton, R. Souza, and R. Azevedo, "Identifying and validating java misconceptions toward a cs1 concept inventory," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2019, p. 23–29.

[26] L. Ma, "Investigating and improving novice programmers' mental models of programming concepts." Ph.D. dissertation, Citeseer, 2007.

[27] D. Sleeman, R. T. Putnam, J. Baxter, and L. Kuspa, "Pascal and high school students: A study of errors," *Journal of Educational Computing Research*, vol. 2, no. 1, pp. 5–23, 1986. [Online]. Available: https://doi.org/10.2190/2XPP-LTYH-98NQ-BU77

[28] E. Middendorff, B. Apolinarski, K. Becker, P. Bornkessel, T. Brandt, S. Heißenberg, and J. Poskowsky, *Die wirtschaftliche und soziale Lage der Studierenden in Deutschland 2016. Zusammenfassung zur 21. Sozialerhebung des Deutschen Studentenwerks durchgeführt vom Deutschen Zentrum für Hochschul- und Wissenschaftsforschung*. Berlin: Bundesministerium für Bildung und Forschung (BMBF), 06 2017. [Online]. Available: tinyurl.com/yhutdmkj

[29] J. Bergin, M. Stehlik, J. Roberts, and R. Pattis, *Karel J. Robot: A gentle introduction to the art of object-oriented programming in Java*. Dream Songs Redwood City, 2005.

[30] D. J. Leiner, "Sosci survey (version 3.2.12)," 2020. [Online]. Available: https://www.soscisurvey.de/

[31] R. S. Duran, J.-M. Rybicki, A. Hellas, and S. Suoranta, "Towards a common instrument for measuring prior programming knowledge," in *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. New York, NY, USA: ACM, 2019, p. 443–449.

[32] A. Venables, G. Tan, and R. Lister, "A closer look at tracing, explaining and code writing skills in the novice programmer," in *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*. New York, NY, USA: ACM, 2009, p. 117–128.

[33] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating the viability of mental models held by novice programmers," in *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 2007, p. 499–503.

[34] M. R. Armat, A. Assarroudi, M. Rad, H. Sharifi, and A. Heydari, "Inductive and deductive: Ambiguous labels in qualitative content analysis," *The Qualitative Report*, vol. 23, no. 1, pp. 219–221, 2018.

[35] P. Mayring, *Qualitative content analysis: theoretical foundation, basic procedures and software solution*, Klagenfurt, 2014.

[36] Intento and e2f, "The state of machine translation 2022: An independent multi-domain evaluation of mt engines," Tech. Rep., 2022. [Online]. Available: https://inten.to/machine-translation-report-2022/

[37] A. Reuneker, "N-gram generator. retrieved mar 14 2023," 2019. [Online]. Available: https://www.reuneker.nl/files/ngram/

[38] M. Corney, R. Lister, and D. Teague, "Early relational reasoning and the novice programmer: Swapping as the "hello world" of relational reasoning," in *Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114*. AUS: Australian Computer Society, Inc., 2011, p. 95–104.

[39] T. Pelchen and R. Lister, "On the frequency of words used in answers to explain in plain english questions by novice programmers," in *Proceedings of the Twenty-First Australasian Computing Education Conference*. New York, NY, USA: ACM, 2019, p. 11–20.

[40] S. Holland, R. Griffiths, and M. Woodman, "Avoiding object misconceptions," in *Proceedings of the Twenty-Eighth SIGCSE Technical Symposium on Computer Science Education*. New York, NY, USA: ACM, 1997, p. 131–134.

[41] F. K. Oser, C. Näpflin, C. Hofer, and P. Aerni, *Towards a Theory of Negative Knowledge (NK): Almost-Mistakes as Drivers of Episodic Memory Amplification*. Dordrecht: Springer Netherlands, 2012, pp. 53–70. [Online]. Available: https://doi.org/10.1007/978-90-481-3941-5_4

8. Application Study I: Connection between Natural Language and Programming Language as Base for Instructional Videos

Bibliographic Information

This contribution is based on the following publication, the version below being the camera-ready version.

Reprinted, with permission, from:

> Svana Esche and Karsten Weihe. 2023. Choosing a Didactic Basis for an Instructional Video: What Are the Implications for Novice Programmers?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023)*, July 8–12, 2023, Turku, Finland. Association for Computing Machinery, New York, NY, USA, 450-456. `https://doi.org/10.1145/3587102.3588795`

The contribution of the author of this thesis is summarized as follows.

> *"As corresponding author, Svana Esche has contributed in all phases, including developing the research questions, conceptual research design, planning research activities, data collection, data analysis and interpretation, and writing the manuscript. Karsten Weihe provided valuable feedback as a co-author during the revision part of the writing process of this publication."*

Choosing a Didactic Basis for an Instructional Video: What Are the Implications for Novice Programmers?

Svana Esche
svana.esche@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

Karsten Weihe
karsten.weihe@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Germany

ABSTRACT

Much work on instructional videos in computing education focuses on the overall impact and technical aspects of videos, such as motivation and length. However, it might be significant how the underlying pedagogical theory, the didactic basis, determines the delivery of the content. We conducted a randomized experiment to investigate the research question: How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers? Our data included two cohorts of 133 and 428 CS1 students from the Fall semesters of 2021/22 and 2022/23, respectively. In cohort 1, videos based on language-sensitive teaching led to significantly better results in writing code in object orientation for novices with medium basic skills than videos based on worked examples. This result could not be replicated in cohort 2. We found no effect on novice self-efficacy in either cohort.

CCS CONCEPTS

• **Social and professional topics → CS1**.

KEYWORDS

CS1, programming, Java, instructional video

ACM Reference Format:
Svana Esche and Karsten Weihe. 2023. Choosing a Didactic Basis for an Instructional Video: What Are the Implications for Novice Programmers?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587102.3588795

1 INTRODUCTION

Instructional videos have gained prominence and prevalence in introductory programming courses. This trend has been increasing since 2020 due to the pandemic; see the increasing number of publications in the ACM Digital Library [18]. Computing education research (CER), for example, has examined what technical aspects of videos capture students' attention [12], the domain of the videos themselves [37], and videos with or without quizzes [16].

However, each video has an underlying pedagogical theory, the *didactic basis*, which determines the delivery of the content. The didactic basis could also effect the benefits of an instructional video. As far as we are aware, the effects of different didactic bases have not been compared before. This study examines the effects on learner performance in terms of code writing and self-efficacy. The latter is also interesting because of its various connections to other constructs [32] and because it is a "legitimate outcome in itself" [40]. We do not assume that the videos have the same effect on all novices, since they indeed have heterogeneous levels of skills. Based on the previous considerations, the research question is:

RQ How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers? [1]

To this end, we conducted an experimental study using Item Response Theory (IRT) [4] to ensure the validity and reliability of the test instruments used. For the experiment, we narrowed down the content of the instructional videos to Java as programming language (PL) and object orientation (OO). The reasons for this are: Java is one of the most common PL in CS1 [9] and therefore relevant to many CS1 courses. Parts of OO are particularly difficult content in CS1, e.g., references and pointers [17], and inheritance [10]. In addition, Xinogalos described a research gap in teaching approaches to OO [38].

2 THEORIES

Instructional videos are intended to provide scaffolding for novices. We therefore describe Vygotsky's theory concerning this scaffolding (Sect. 2.1), the implementation of the videos through various didactic bases (Sect. 2.2), and the refinements of the RQ (Sect. 2.3).

2.1 Vygotsky's Zone of Proximal Development

We approach the concept of scaffolding through Vygotsky's theoretical concept of ZPD, which stands for "zone of proximal development" [36], meaning the zone of next development. We briefly describe what the term ZPD encompasses. There is an initial zone where learners can solve the task independently. In the ZPD, learners cannot solve the task independently, but only with support. In the third zone, learners fail to solve the task, either with or without support. One of two perspectives of ZPD is the "pedagogy" [39]. Pedagogy is about taking action to help students reach a higher zone. Vygotsky used the following phrase to explain how students act out of ZPD: "the imitative performance of some intellectual operation that is reasoned and based on understanding" [39].

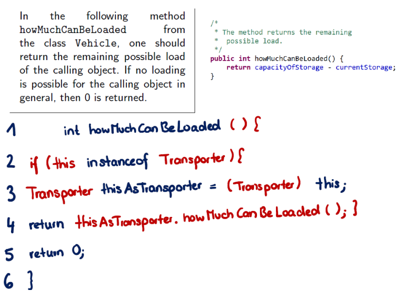
[1] This paper builds upon an extended abstract [8]. Neither the theories, the full methodology, nor the second cohort in general were included there.

Next, we apply the concept of ZPD to instructional video support. The video illustrates to novices how to solve a particular problem. Thus, the novices see in the video what thought process they could *imitate* afterwards. This imitation could lead to positive effects in the code writing (CW) task because it provides scaffolding. This imitation will probably have different effects on the novices because their basic skills of OO are different. The novices with good basic skills do not need imitation because they can solve the task by themselves, i.e., falling into the first zone of the ZPD concept. In contrast, the students with low basic skills lack the necessary understanding. Thus, imitation would *not* be “based on understanding” [39]. These novices fall into the third zone and the instructional video will not be able to help them if they are to solve a later task independently.

2.2 Didactic Bases

We present two specific didactic bases, namely the worked examples (WE) approach and the language-sensitive teaching (LST) approach adopted from mathematics education. We also show how we integrated these into the corresponding videos, see Fig. 1.

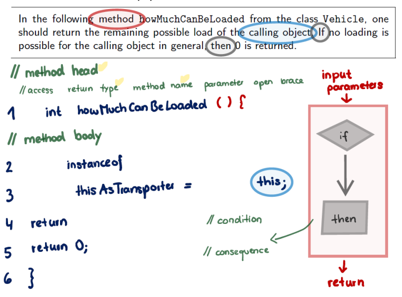
The field of research on worked examples is wide and the possibilities for designing examples are numerous [25]. Muldner et al. [25] have identified three strands of research, namely examples of both tracing and generating code, and examples that combine the two. A subordinate strand is how to integrate worked examples into the teaching learning process. We focus on code generation examples and their integration due to our RQ. For example, studies have shown the indifference between live coding and static examples [29], the effectiveness of subgoals [22], and the different ways of sequencing examples and problems [25]. There are also, especially for OO, defined evaluation criteria for worked examples [2]. Based on these findings, our WE instructional video included a static example with verbal subgoals, additional explanations, and OO criteria followed. In Fig. 1a, the screenshot shows the static example with the given code in blue and the gaps to be filled in red.


In the following method howMuchCanBeLoaded from the class Vehicle, one should return the remaining possible load of the calling object. If no loading is possible for the calling object in general, then 0 is returned.

```
/*
 * The method returns the remaining
 *  possible load.
 */
public int howMuchCanBeLoaded() {
    return capacityOfStorage - currentStorage;
}
```

```
        int howMuchCanBeLoaded ( ) {
if (this instanceof Transporter ) {
Transporter thisAsTransporter = (Transporter) this;
return thisAsTransporter. howMuchCanBeLoaded ( ); }
return 0;
}
```

(a) WE as basis


In the following method howMuchCanBeLoaded from the class Vehicle, one should return the remaining possible load of the calling object. If no loading is possible for the calling object in general, then 0 is returned.

```
// method head
 // access   return type   method name   parameter   open brace
    int   howMuchCanBeLoaded ( ) {
// method body
        instanceof
         thisAsTransporter =                  this;
 return
 return 0;
}
```

(b) LST as basis

Figure 1: Screenshots of videos with their didactic basis.

The term LST refers to an approach that combines language, reasoning, and, in the case of CS, PL with the goal of scaffolding. The term language here refers to the written and spoken language of learners and in the classroom. In mathematics education, LST is a current research trend for which six important design principles have already been developed [7]. These ensure that language learning serves as a catalyst for learning. One principle, for example, is the linking of language, content, and levels of representation [7], which is based on Bruner’s theory of combining different forms of representation [3]. As far as we know, there is so far empirical evidence of the benefits of this approach only in a sample [27]. In CW, the multiple representations are the code itself, a highly symbolic, structured and dense representation, and the visualizations. There are also the language of the task description and the PL, in which the solution is to be implemented. Our LST video connects the relevant phrases and associated code by circling them, see Fig. 1b. It explicitly tells the viewer that these connections exist and that they are heuristics. It represents a visualization for combining multiple representations. Unlike the static representation in the WE video, the LST video fills in the gaps in the code dynamically. LST is a scaffolding approach. Proficient learners do not need these heuristics, because they already know the connections between phrases and concepts, whether implicit or explicit. The LST approach provides intermediate levels that these learners no longer need, but that may be of use to learners who still need them.

2.3 Implications for Refinement of the RQ

The concept of ZPD led us to the following considerations. We assumed that the effects of the instructional videos would be seen in students with medium basic skills who were likely to be in the ZPD. Students below this level lack the basics, so the video’s help would not be fruitful. Even for students who can solve the task on their own, there will be no difference. We expected a ranking between the videos, namely that the LST video would lead to higher CW and self-efficacy scores. We based this assumption on the transfer of researched efficacy in mathematics didactics [27]. These considerations led to four hypotheses as a refinement of the RQ:

- H1 In general, i.e. for all novices regardless of their basic skills, the didactic basis has no effect on the CW performance.
- H2 Instructional videos based on LST lead to better CW performances than those based on worked examples for novices with medium basic skills.
- H3 Novices watching an LST instructional video have higher self-efficacy scores than when watching a video with worked examples.
- H4 The effect stated in H3 is especially true for novices with medium basic skills.

3 STATE OF THE ART

3.1 Instructional Videos

The use and exploration of instructional videos has increased in recent years. This trend is intensifying due to the pandemic since 2020; see the increasing number of publications in the ACM Digital

Library [18]. Guo et al. [12] studied how video production decisions affect student engagement. As a result, they formulated seven recommendations for video production that have been cited both inside and outside CER. These include, for example, a video length of under six minutes and digital handwritten drawings [12]. In addition to the more technical implementation mentioned above, further research has also focused on the area of videos themselves [37] and the comparison of videos with and without quizzes [16]. Moreover, research examined which learners watch videos [24] and how they compare to learners attending traditional lectures [13]. To the best of our knowledge, how the content is presented or prepared, i.e. the underlying didactic basis, is an open question.

3.2 Learner Performance

Code writing is an integral part of introductory programming courses and falls under the broader term of code literacy, which is currently the focus of research interest [21]. However, learners have a variety of misconceptions and other difficulties related to writing code, which are summarized in a review [28]. When measuring CW tasks, the effort required to measure them reliably and validly is high due to the trade-offs between syntax, semantics, and style. This has led to the search for related tasks such as code completion. Code completion involves filling in blanks [31] or the Parsons Puzzle, which Du et al. have summarized in a review [5]. Code completion is an emerging trend to complement CW tasks, especially since these tasks correlate well with CW tasks [5, 31].

Self-efficacy is "an individual's belief in their ability to complete a task" [11]. We only give an overview of what has been learned so far, as this is a well-researched area in CER. Low scores on self-efficacy may have an impact on the high dropout rates in CS, e.g., [34]. Steinhorst et al. [32] reviewed research on the interaction of self-efficacy with other concepts such as belonging as part of retention, e.g., [35]. In addition, self-efficacy is related to mindset [11], longer-term interest, and novice performance in CS [19], the latter of which interact with self-efficacy [20]. However, self-efficacy and personal experience do not always interact [14]. Promoting self-efficacy is considered "a legitimate outcome in itself" [40] and is therefore integrated into our RQ. There are two widely used tests for measuring self-efficacy in CER: (1) the *Computer Programming Self-Efficacy Scale* (CPSES) [30] as a specific test with 32 items, with an updated version of 20 items [32]; (2) the subset of the general but widely used *Manual for the Use of Motivated Strategies for Learning Questionnaire* (MSLQ) [26] with eight items.

4 METHODOLOGY

4.1 Participants

Our two cohorts included students enrolled in the CS1 course at a central European university. Cohort 1 included 133 students from 2021/22 and cohort 2 included 428 students from 2022/23, both for fall semesters. At the time of the study, there were 792 and 916 submitted homework assignments, respectively, representing the active population of the courses. Thus, the response rates were 16.92% and 46.72%, respectively. We excluded students for not responding (cohort 1: 3, cohort 2: 0) and for uploading CW data in an incorrect format (cohort 1: 4, cohort 2: 23). For cohort 2, we excluded additional participants: 26 because they were part of cohort 1 and therefore repeated the CS1 course, another 11 because they participated more than once, and 1 because the use of their data was not allowed. Thus, our samples are n=126 and n=367, respectively.

In cohort 1, we did not ask for demographic data because the ethics committee did not approve. In cohort 2, we solved this problem by adding a separate, optional demographic questionnaire. Twice as many students completed this questionnaire as in the actual study. The 844 students had a median age of 20, 63% were CS majors, 28% identified as female, 1% as diverse, 37% had a migration background, and 50% had CS as a school subject. Due to separation, this distribution may differ from that of the actual study.

The course structure was the same for both cohorts: The second author taught the 14-week courses and was assisted by 30 student teaching assistants. The central theme was teaching programming in Java. This included the basics of OO as well as static and dynamic types, generics, and error handling. Students received points for 13 individual CW homework assignments, with half of the total points required for admission to the exam. There was also an optional programming group project and a non-optional written exam.

Seven weeks after the start of the course, students voluntarily participated in the study. Participants in the first cohort had the chance to receive one of 20 vouchers, each containing a small amount of money as an incentive, while participants in the second cohort received a small number of bonus points for the course. The participation period was nine weeks in cohort 1 and two weeks in cohort 2. The long first participation period was chosen to attract a larger number of participants. Three of these nine weeks were Christmas vacations, so no classes were held during these weeks.

Students participated anonymously to ensure privacy. At the beginning of the study, participants received a description of the subject, process, duration, and benefits of the study. After that, they could proceed only if they gave informed consent.

4.2 Experimental Design

We conducted a randomized experiment in the form of an online questionnaire consisting of four phases. A randomized experiment provides high reliability and validity of the effects of the independent variables. We also tested the results using a second cohort. Tasks and videos took place in the real scenario of car sharing, which addressed the third level of understanding in OO [6]. The novices did not have to write the classes themselves, but they had to use them (phase I) and modify them (phase IV). We conducted the experiment separately for each cohort and listed all values side by side. For phases I and III, we performed an item response theory (IRT) analysis using the open-source software jMetrik [23]. Our steps were: (1) Determination of an appropriate subset of items based on item fit statistics such as infit and outfit. In addition, principal factor analysis of standardized residuals was considered as a post hoc test; (2) Identification of the IRT model that best describes the data; (3) Determination of skill levels based on individual ability.

Phase I: Basic Skills. This phase determined the basic skill level according to OO. The *Fill in blanks* task with seven gaps focused on objects, references, classes, and inheritance. In these gaps, we identified 16 components as binary items. We chose this task as its performance correlates with CW performance [31] and the minimum probability of guessing. IRT analysis contributed to valid measurement. First, IRT analysis showed that the task measured multiple abilities. We then divided the items into two subgroups: semantic and syntactic items. For both cohorts, the IRT 2PL model provided the best model fit. Based on performance in the semantic subgroup, we divided participants equally into preliminary skill levels 2 to 4, with 4 being the highest level. If a novice's performance in the syntactic subgroup was below a certain threshold, we lowered their skill level by 1. Participants with skill levels 2 or 3 formed the *medium basic skills* group.

Phase II: Instructional Videos. We randomly assigned participants to one of three groups: (1) a control group, *(no video)*, whose participants did not watch a video; (2) the *(WE-video)* group, who watched an instructional video using worked examples; and (3) the *(LS-video)* group, who watched an instructional video based on language-sensitive teaching. The first author produced both videos including transcripts, handwritten annotations, audio, and editing. Both videos addressed the same example problem, the difference between static and dynamic type, with the learning goal of supporting the CW task in phase IV. Other similarities include the use of captions, and almost the same length, i.e., 4:04 and 4:35 minutes, which was below the recommended length of 6:00 [12].

Phase III: Self-Efficacy. Novices rated their self-efficacy on the next CW task on a 7-point Likert scale. We chose the MSLQ [26] as an established and validated test. A Cronbach's α of .93 indicates that the items in the MSLQ self-efficacy domain are partially redundant [33]. We selected five of the eight items numbered 12, 15, 20, 21, and 29 and tailored them to our study. The CPSES tests [30, 32] were not selected because of their length. We used the rating scale model [1]. We looked at characteristic curves (CC) to determine which combination of rating scales gave the best measurement. The combination of ratings 1 and 2 and ratings 3 and 4 improved the shapes and orders of the CC. Our items had very good internal consistency as indicated by high Cronbach's α values (Cohort 1: α=.934; Cohort 2: α=.862). Thus, selection did not weaken reliability.

Phase IV: Code Writing. Participants could choose between CW templates for IntelliJ and Eclipse as IDEs, similar to their homework. They downloaded the template, worked on the task, and uploaded the solutions. We used the scoring scheme in Table 1 to reduce style bias and unfair subjective scoring.

Table 1: Scoring scheme for the skill level (SL) in Phase IV.

SL	Definition
1	*Compile error:* The code produces at least one compile error.
2	*Multiple incorrect formal parameters:* The code compiles, but there are at least constructors or methods that have more than one incorrect formal parameter.
3	*Single incorrect formal parameter:* The code compiles, but there is a constructor or method that has exactly one incorrect formal parameter.
4	*Failure to pass JUnit tests:* The code compiles, all constructors and methods have the correct formal parameters, but at least one JUnit test fails.
5	*Everything is correct:* The code compiles, all constructors and methods have the correct formal parameters, and all JUnit tests passed.

4.3 Statistical Analysis

We next list the statistical tests for the hypotheses. For phase III, the student's self-efficacy score was measured on an interval scale according to IRT analysis. Therefore, we conducted multiple t-tests with prior checking of the premises. For phase IV, the level of CW proficiency was measured on an ordinal scale. Thus, we performed multiple Mann-Whitney U tests (MWM) with prior two-sample Kolmogorov-Smirnov (KS) tests. When KS is not significant, the MWM actually compares medians. Effect size was calculated using Freeman's θ. Correction for multiple testing is required because there are multiple hypotheses, two for both phases III and IV, and three groups to compare for each hypothesis. Therefore, we adjusted the significance levels α with the Bonferroni correction from α=.05 to α=.0083; analogously, from α=.01 to α=.0017.

5 RESULTS

We group results according to the corresponding hypotheses given in Sect. 2.3. Hypotheses H1 and H3 refer to all novices, H2 and H4 to novices with medium basic skills as defined in Sect. 4.2.

First, we present results according to H1 and H2. We show the distributions of skill levels in the form of boxplots, see Fig. 2. The distributions between the two cohorts are *not* similar. Median scores differ more for the first cohort than the second cohort. Second, we tested whether these median differences were statistically significant, see Table 2. For H1, both distributions and medians did not differ statistically significantly between the groups. Thus, hypothesis H1 was confirmed for both cohorts. For H2, there are significant differences between groups. For cohort 1, the distributions between the *WE-video* and *LS-video* groups did not differ. Thus, the MWM actually compared the median of the groups. The median differed between the groups in terms of α=.05 with a large effect. The treatment group with LS-video had significantly better results. Thus, hypothesis H2 was confirmed for cohort 1. However, the results of cohort 2 did not confirm H2 as the differences were not significant.

Second, we present results according to H3 and H4. As a prerequisite for the t tests, we tested for normality using the Shapiro-Wilk test. In cohort 1, all treatment groups had a normal distribution. In contrast, two of six groups did not have a normal distribution. We still used the t-test because it is robust to this violation of the normal distribution given the large sample size. We tested differences in self-efficacy for significance with multiple t-tests, see Table 3, using Levene's test to test homoscedasticity. For both cohorts, the mean scores did not differ statistically significantly between the groups. Therefore, hypotheses H3 and H4 could *not* be confirmed.

6 DISCUSSION

6.1 Answers towards the Research Question and its Hypotheses

The research question was: *How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers?* We conducted a randomized experiment with two cohorts to answer this question. We have distinguished between worked examples and language-sensitive teaching as a didactic basis. Next, we discuss our four hypotheses (see Sect. 2.3) based on Vygotsky's concept of ZPD.

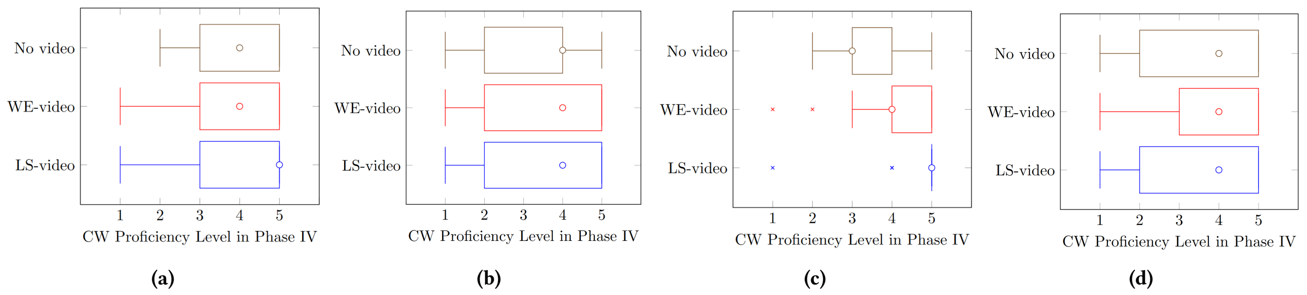


Figure 2: Boxplots of CW skill levels in phase IV with medians as circles and outliers as crosses: All novices, (a) 1st Cohort (n=126) and (b) 2nd Cohort (n=367); novices with medium basic skills, (c) 1st Cohort (n=62) and (d) 2nd Cohort (n=172).

Table 2: Comparing medians for CW in phase IV, grouped by hypothesis (H) with all novices (H3) or novices with medium basic skills (H4), Kolmogorov-Smirnov test (KS), Mann-Whitney U test, sample size (n), and Freeman's θ with confidence interval (CI). With the Bonferroni correction, we mark the statistical significance according to α=.05 with an asterisk, α=.01 with a double.

					Mann-Whitney U test				Freeman's θ	
H	**Cohort**	**Compared Groups**	**Compared Medians**	**KS**	**U**	**Z**	**p**	**n**	θ	**95% CI**
H1	1	No Video vs. WE-Video	4 vs. 4	1.000	906.5	-0.123	.903	86	.015	[.004, .268]
H1	1	No Video vs. LS-Video	4 vs. 5	.913	697.5	-1.051	.293	80	.128	[.008, .363]
H1	1	WE-Video vs. LS-Video	4 vs. 5	.513	776.5	-1.316	.188	86	.156	[.008, .400]
H1	2	No Video vs. WE-Video	4 vs. 4	.422	6932.0	-0.067	.947	237	.005	[.003, .170]
H1	2	No Video vs. LS-Video	4 vs. 4	.857	6510.5	-1.015	.311	238	.073	[.004, .210]
H1	2	WE-Video vs. LS-Video	4 vs. 4	.788	7793.0	-0.994	.322	259	.071	[.003, .208]
H2	1	No Video vs. WE-Video	3 vs. 4	.1237	185.5	-1.8048	.0711	46	.299	[.028, .595]
H2	1	No Video vs. LS-Video	3 vs. 5	.0025*	70.5	-3.4434	.0006**	39	.617	[.303, .870]
H2	1	WE-Video vs. LS-Video	4 vs. 5	.0153	93.0	-2.8553	.0043*	39	.495	[.145, .762]
H2	2	No Video vs. WE-Video	4 vs. 4	.283	1622.0	-0.298	.767	116	.015	[.003, .265]
H2	2	No Video vs. LS-Video	4 vs. 4	.510	1426.5	-0.529	.600	110	.057	[.004, .265]
H2	2	WE-Video vs. LS-Video	4 vs. 4	.459	1674.5	-0.344	.732	118	.035	[.003, .245]

First, we present the answers to H1 and H2. Hypothesis H1 was confirmed, i.e. for all novices the didactic basis has no effect on the CW performance. Hypothesis H2 was only confirmed for cohort 1, i.e. the instructional video with language-sensitive teaching led to better CW performance than the video based on worked examples *for novices with medium basic skills*. However, we could not replicate this confirmation in cohort 2. In response to RQ, we could not confirm that didactic bases affect CW performance at all. We discuss three possible reasons why the results could not be replicated.

(1) The experiment was designed to use the Fill in blanks task in phase I to determine the students' ZPD. It seems that determining the ZPD requires more fine-tuning. Vygotsky himself attached great importance to the determination of ZPD. He referred to "diagnosis" [39] along with the aforementioned "pedagogy" as scaffolding as the two perspectives of ZPD.

(2) Participants differed in providing Java code with compilation errors: 4% (cohort 1) versus 21% (cohort 2) for skill level 1 in phase IV. This large difference also affected the test of hypothesis H2, which could not be confirmed in cohort 2. Possible explanations are that participants in the two cohorts might differ in their motivation and basic knowledge about using IDEs. Motivation could be influenced by the fact that bonus points are awarded for participation instead of vouchers. However, cohort 2 participants spent more time on the CW task than cohort 1 participants (39:10 vs. 31:03 minutes).

(3) In Sect. 4.2 we stated that the Fill in blanks task as a code completion task and the CW task correlate well [31]. Sindre [31] calculated the correlation between scores on nine code completion tasks and five CW tasks and arrived at a Pearson's ρ of .80. Only two of the nine code completion tasks were Fill in blanks tasks. Thus, it could be that Fill in blanks tasks did not achieve the same level of correlation as all the code completion tasks combined. In this case, our assumption that phase I measured basic CW skills was incorrect. We repeat the correlation calculation with our data. For Fill in blanks tasks, Sindre [31] gave points for each correctly filled gap and added them. We ignore that summing ordinal data is problematic. Our data resulted in low Spearman's ρ values (cohort 1, ρ=.309; cohort 2, ρ=.366). These values are much lower than the Pearson's ρ of .80 reported by Sindre [31]. We conclude that the question of under what circumstances Fill in blanks tasks and CW tasks correlate is still open. In summary, the non-replication can be partly explained by the fact that the determination of ZPD in CW still needs to be improved.

Second, we present the answers to H3 and H4. None of these hypotheses could be confirmed, because all statistical comparisons

Table 3: Comparing means for self-efficacy in phase III, grouped by hypothesis (H) with all novices (H3) or novices with medium basic skills (H4), Levene's test, t-test with degrees of freedom (df), and Cohen's d with its confidence interval (CI).

				Levene's Test		t-Test			Cohen's d	
H	**Cohort**	**Compared Groups**	**Compared Means**	**F**	**p**	**T**	**df**	**p**	**d**	**95% CI**
H3	1	No video vs. WE-video	.429 vs. .040	0.735	.394	0.503	85	.616	.108	[-.314, .529]
H3	1	No video vs. LS-video	.429 vs. .368	0.341	.561	0.080	80	.936	.018	[-.415, .451]
H3	1	WE-video vs. LS-video	.040 vs. .368	0.368	.818	-0.407	85	.685	-.087	[-.508, .334]
H3	2	No video vs. WE-video	.124 vs. -.480	0.713	.399	1.429	232	.154	.188	[-.070, .445]
H3	2	No video vs. LS-video	.124 vs. -.148	0.007	.934	0.613	234	.541	.080	[-.176, 336]
H3	2	WE-video vs. LS-video	-.480 vs. -.148	0.595	.441	-0.817	254	.414	-.102	[-.346, .143]
H4	1	No video vs. WE-video	-.766 vs. .263	0.146	.704	-1.208	45	.233	-.353	[-.927, .226]
H4	1	No video vs. LS-video	-.766 vs. -.483	1.028	.317	-0.333	39	.741	-.106	[-.727, .517]
H4	1	WE-video vs. LS-video	.263 vs. -.483	0.252	.618	0.829	38	.412	.265	[-.336, .893]
H4	2	No video vs. WE-video	.107 vs. -.003	0.269	.605	0.207	112	.836	.039	[-.329, .406]
H4	2	No video vs. LS-video	-.107 vs. -.404	0.478	.491	1.013	108	.313	.193	[-.182, .567]
H4	2	WE-video vs. LS-video	-.003 vs. -.404	1.463	.229	0.783	114	.435	.146	[-.219, .510]

of self-efficacy results were not significant. Self-efficacy and performance were found to influence each other [20]. Thus, novices' performance in phase I would have influenced their self-efficacy. Gorson and O'Rourke [11] identified situations that negatively impact student self-efficacy, such as "spending a long time on a problem". We examine the time spent on the task and its relationship to self-efficacy scores. In phase I, the median time spent on the task was 08:10 minutes for cohort 1 and 10:25 minutes for cohort 2. With Spearman's rank correlation coefficient, there is a significant but weak negative correlation between time spent and self-efficacy rating (1. cohort r_s=-.259, p=.003; 2. cohort r_s=-.106, p=.043). This effect is consistent with previous results [11]. Thus, because of the strong effect of phase I, the videos had no effect on self-efficacy.

6.2 Implications for Teaching and Research

Instructors (not excluding us) might recognize that didactic basis is not yet the defining characteristic of good student adjustment. Instead, diagnosis is important, namely, diagnosing where the student's ZPD lies so that the intended positive effects of scaffolding can be realized. Future research could investigate whether there are certain didactic bases that help students better than others at certain stages of their learning. We emphasize that adaptation during learning phases is meant, not adaptation of didactic principles to learning styles. The latter are now discredited as pseudoscience [15]. We take this idea further and summarize the previously described implications under the term adaptive learning. This means that instructional materials and tasks are continuously adjusted to the learner's current level of knowledge. For example, students who lack the basics watch a video reviewing the basics before the actual video. This area is currently receiving attention in the e-learning field of education and seems promising for CER as well.

6.3 Threats to Validity

The novices examined in this study were self-selected. We compensated for this by randomizing the treatment and setting a code completing task rather than self-reporting the level of basic skill.

The study was conducted halfway through the semester. Students who dropped the CS1 course early did not participate in the study. All students were from the same university, and the videos were all in the same (non-English) natural language. Including other universities, and especially transcribing the videos into other languages, would have led to better generalizability of the results.

In addition, the study contained a single CS1 content: OO with classes, objects, inheritance, and the difference between static and dynamic types in Java. Replication with other CS1 content would be beneficial to study the effects of instructional videos.

7 CONCLUSIONS

Instructional videos are an essential component of online teaching, blended learning, and flipped classrooms. In addition, instructional videos are one approach among others for scaffolding. However, previous research has focused on the technical or motivational effects of videos. This study examined the didactic basis and its effects on code writing performance and self-efficacy. Below, we summarize the research question and its answer.

RQ *How does the didactic basis of an instructional video affect code writing performance and self-efficacy given the basic skill of novice programmers?* **Answer** In a first cohort, different didactic bases indeed led to differences in code writing performance. However, we could not replicate this result in a second cohort. Moreover, we did not find differences in self-efficacy. More specifically, for novices with medium basic skills, videos based on language-sensitive teaching led to significantly better results in writing code compared to videos based on worked examples. For novices in general, we found no difference between these didactic bases.

8 ACKNOWLEDGEMENTS

The authors thank the "Qualitätsoffensive Lehrerbildung", a joint initiative of the Federal Government and the Länder which is funded by the Federal Ministry of Education and Research, under Grant 01JA1814, and supported in part the work of the first author. The authors are responsible for the content of this publication.

REFERENCES

[1] David Andrich. 1978. Application of a psychometric rating model to ordered categories which are scored with successive integers. *Applied psychological measurement* 2, 4 (1978), 581–594.

[2] Jürgen Börstler, Henrik B. Christensen, Jens Bennedsen, Marie Nordström, Lena Kallin Westin, Jan Erik Moström, and Michael E. Caspersen. 2008. Evaluating OO Example Programs for CS1. *SIGCSE Bull.* 40, 3 (June 2008), 47–52. https://doi.org/10.1145/1597849.1384286

[3] Jerome S. Bruner. 1967. *Toward a Theory of Instruction.* Belknap Press of Harvard University, Cambridge, Massachusetts.

[4] Christine DeMars. 2010. *Item response theory.* Oxford University Press.

[5] Yuemeng Du, Andrew Luxton-Reilly, and Paul Denny. 2020. A Review of Research on Parsons Problems. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20).* ACM, New York, NY, USA, 195–202. https://doi.org/10.1145/3373165.3373187

[6] Anna Eckerdal and Michael Thuné. 2005. Novice Java Programmers' Conceptions of "Object" and "Class", and Variation Theory. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05).* ACM, New York, NY, USA, 89–93. https://doi.org/10.1145/1067445.1067473

[7] Kirstin Erath, Jenni Ingram, Judit Moschkovich, and Susanne Prediger. 2021. Designing and enacting instruction that enhances language for mathematics learning: A review of the state of development and research. *ZDM–Mathematics Education* 53, 2 (2021), 245–262.

[8] Svana Esche. 2022. Linking of Language and Programming and its Effects on Code Writing and Self-Efficacy in CS1. In *1. Nachwuchs-Konferenz der Didaktik der Informatik.* Fachgruppe DDI der Gesellschaft für Informatik, 11–13.

[9] Onyeka Ezenwoye. 2018. What Language? - The Choice of an Introductory Programming Language. In *2018 IEEE Frontiers in Education Conference (FIE).* IEEE, San Jose, CA, USA, 1–8. https://doi.org/10.1109/FIE.2018.8658592

[10] Ken Goldman, Paul Gross, Cinda Heeren, Geoffrey Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Identifying Important and Difficult Concepts in Introductory Computing Courses Using a Delphi Process. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education* (Portland, OR, USA) *(SIGCSE '08).* ACM, New York, NY, USA, 256–260. https://doi.org/10.1145/1352135.1352226

[11] Jamie Gorson and Eleanor O'Rourke. 2020. Why Do CS1 Students Think They're Bad at Programming? Investigating Self-Efficacy and Self-Assessments at Three Universities. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20).* ACM, New York, NY, USA, 170–181. https://doi.org/10.1145/3372782.3406273

[12] Philip J. Guo, Juho Kim, and Rob Rubin. 2014. How Video Production Affects Student Engagement: An Empirical Study of MOOC Videos. In *Proceedings of the First ACM Conference on Learning @ Scale Conference* (Atlanta, Georgia, USA) *(L@S '14).* ACM, New York, NY, USA, 41–50. https://doi.org/10.1145/2556325.2566239

[13] Petri Ihantola, Juho Leinonen, and Matti Rintala. 2020. Students' Preferences Between Traditional and Video Lectures: Profiles and Study Success. In *Koli Calling '20: Proceedings of the 20th Koli Calling International Conference on Computing Education Research* (Koli, Finland) *(Koli Calling '20).* ACM, New York, NY, USA, Article 29, 5 pages. https://doi.org/10.1145/3428029.3428561

[14] Päivi Kinnunen and Beth Simon. 2012. My program is ok – am I? Computing freshmen's experiences of doing programming assignments. *Computer Science Education* 22, 1 (2012), 1–28. https://doi.org/10.1080/08993408.2012.655091

[15] Paul A. Kirschner. 2017. Stop propagating the learning styles myth. *Computers & Education* 106 (2017), 166–171. https://doi.org/10.1016/j.compedu.2016.12.006

[16] Lisa L. Lacher, Albert Jiang, Yu Zhang, and Mark C. Lewis. 2018. Including Coding Questions in Video Quizzes for a Flipped CS1. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18).* ACM, New York, NY, USA, 574–579. https://doi.org/10.1145/3159450.3159504

[17] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (Caparica, Portugal) *(ITiCSE '05).* ACM, New York, NY, USA, 14–18. https://doi.org/10.1145/1067445.1067453

[18] ACM Digital Library. 2023. *Publication Date for the Query "instructional video novice".* https://tinyurl.com/m9xad82m

[19] Alex Lishinski and Joshua Rosenberg. 2021. All the Pieces Matter: The Relationship of Momentary Self-Efficacy and Affective Experiences with CS1 Achievement and Interest in Computing. In *Proceedings of the 17th ACM Conference on International Computing Education Research* (Virtual Event, USA) *(ICER 2021).* ACM, New York, NY, USA, 252–265. https://doi.org/10.1145/3446871.3469740

[20] Alex Lishinski, Aman Yadav, Jon Good, and Richard Enbody. 2016. Learning to Program: Gender Differences and Interactive Effects of Students' Motivation, Goals, and Self-Efficacy on Performance. In *Proceedings of the 2016 ACM Conference on International Computing Education Research* (Melbourne, VIC, Australia) *(ICER '16).* ACM, New York, NY, USA, 211–220. https://doi.org/10.1145/2960310.2960329

[21] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education* (Larnaca, Cyprus) *(ITiCSE 2018 Companion).* ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[22] Lauren E. Margulieux, Briana B. Morrison, and Adrienne Decker. 2019. Design and Pilot Testing of Subgoal Labeled Worked Examples for Five Core Concepts in CS1. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (Aberdeen, Scotland Uk) *(ITiCSE '19).* ACM, New York, NY, USA, 548–554. https://doi.org/10.1145/3304221.3319756

[23] J. Patrick Meyer. 2014. *Applied Measurement with jMetrik.* Routledge, Florence.

[24] Colin Moore, Lina Battestilli, and Ignacio X. Domínguez. 2021. *Finding Video-Watching Behavior Patterns in a Flipped CS1 Course.* ACM, New York, NY, USA, 768–774. https://doi.org/10.1145/3408877.3432359

[25] Kasia Muldner, Jay Jennings, and Veronica Chiarelli. 2022. A Review of Worked Examples in Programming Activities. *ACM Trans. Comput. Educ.* 23, 1, Article 13 (dec 2022), 35 pages. https://doi.org/10.1145/3560266

[26] Paul R. Pintrich, David A. F. Smith, Teresa Garcia, and Wilbert J. McKeachie. 1991. *A Manual for the Use of the Motivated Strategies for Learning Questionnaire (MSLQ).* Technical Report. National Center for Research to improve Postsecondary Teaching and Learning, Ann Arbor.

[27] Susanne Prediger and Lena Wessel. 2013. Fostering German-language learners' constructions of meanings for fractions—design and effects of a language- and mathematics-integrated intervention. *Mathematics Education Research Journal* 25, 3 (jun 2013), 435–456. https://doi.org/10.1007/s13394-013-0079-2

[28] Yizhou Qian and James Lehman. 2017. Students' Misconceptions and Other Difficulties in Introductory Programming: A Literature Review. *ACM Trans. Comput. Educ.* 18, 1, Article 1 (Oct. 2017), 24 pages. https://doi.org/10.1145/3077618

[29] Adalbert Gerald Soosai Raj, Pan Gu, Eda Zhang, Arokia Xavier Annie R, Jim Williams, Richard Halverson, and Jignesh M. Patel. 2020. Live-Coding vs Static Code Examples: Which is Better with Respect to Student Learning and Cognitive Load?. In *Proceedings of the Twenty-Second Australasian Computing Education Conference* (Melbourne, VIC, Australia) *(ACE'20).* ACM, New York, NY, USA, 152–159. https://doi.org/10.1145/3373165.3373182

[30] Vennila Ramalingam and Susan Wiedenbeck. 1998. Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analyses of Novice Programmer Self-Efficacy. *Journal of Educational Computing Research* 19, 4 (1998), 367–381. https://doi.org/10.2190/C670-Y3C8-LTJ1-CT3P

[31] Guttorm Sindre. 2020. Code Writing vs Code Completion Puzzles: Analyzing Questions in an E-exam. In *2020 IEEE Frontiers in Education Conference (FIE).* IEEE, 1–9. https://doi.org/10.1109/FIE44824.2020.9273919

[32] Phil Steinhorst, Andrew Petersen, and Jan Vahrenhold. 2020. Revisiting Self-Efficacy in Introductory Programming. In *Proceedings of the 2020 ACM Conference on International Computing Education Research* (Virtual Event, New Zealand) *(ICER '20).* ACM, New York, NY, USA, 158–169. https://doi.org/10.1145/3372782.3406281

[33] Mohsen Tavakol and Reg Dennick. 2011. Making sense of Cronbach's alpha. *International journal of medical education* 2 (2011), 53.

[34] F. Boray Tek, Kristin S. Benli, and Ezgi Deveci. 2018. Implicit Theories and Self-Efficacy in an Introductory Programming Course. *IEEE Transactions on Education* 61, 3 (2018), 218–225. https://doi.org/10.1109/TE.2017.2789183

[35] Nanette Veilleux, Rebecca Bates, Cheryl Allendoerfer, Diane Jones, Joyous Crawford, and Tamara Floyd Smith. 2013. The Relationship between Belonging and Ability in Computer Science. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13).* ACM, New York, NY, USA, 65–70. https://doi.org/10.1145/2445196.2445220

[36] Lev Semenovich Vygotsky. 1978. *Mind in society: Development of higher psychological processes.* Harvard university press.

[37] Michael Whitney and Bryan Dallas. 2019. Captioning Online Course Videos: An Investigation into Knowledge Retention and Student Perception. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (Minneapolis, MN, USA) *(SIGCSE '19).* ACM, New York, NY, USA, 511–517. https://doi.org/10.1145/3287324.3287347

[38] Stelios Xinogalos. 2015. Object-Oriented Design and Programming: An Investigation of Novices' Conceptions on Objects and Classes. *ACM Trans. Comput. Educ.* 15, 3, Article 13 (July 2015), 21 pages. https://doi.org/10.1145/2700519

[39] V. K. Zaretskii. 2009. The Zone of Proximal Development: What Vygotsky Did Not Have Time to Write. *Journal of Russian & East European Psychology* 47, 6 (2009), 70–93.

[40] Daniel Zingaro. 2014. Peer Instruction Contributes to Self-Efficacy in CS1. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (Atlanta, Georgia, USA) *(SIGCSE '14).* ACM, New York, NY, USA, 373–378. https://doi.org/10.1145/2538862.2538878

9. Application Study II: Assessment of Answers towards Student Queries about Code with a Focus on Language

Bibliographic Information

This contribution is based on the following publication, the version below being the camera-ready version.

Reprinted, with permission, from:

Svana Esche. 2024. Rubric for the Quality of Answers to Student Queries about Code. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education (SIGCSE '24)*, March 20-23, Portland, OR, USA. Association for Computing Machinery, New York, NY, USA, 331–337. `https://doi.org/10.1145/3626252.3630918`

Rubric for the Quality of Answers to Student Queries about Code

Svana Esche
svana.esche@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hessen, Germany

ABSTRACT

Novice programmers need adequate support to succeed in their courses. This support requires both pedagogical content knowledge and general pedagogical knowledge. These requirements apply to all support staff, e.g., instructors and teaching assistants (TAs). Here we focus on support in the form of answers to student queries about code. We have developed a rubric to assess the quality of answers provided by support staff. In this paper, we present the theoretical framework behind the rubric, the full rubric itself, and two evaluation approaches. First, we evaluated the rubric internally by using it to assess 85 written answers from TAs. From these, we included two sample excerpts and their evaluation using the rubric. Second, our external evaluation included interviews with experts (n=13), which we analyzed using qualitative content analysis. These interviews revealed positive aspects, aspects that could be improved, and other areas of application such as support for reflection.

CCS CONCEPTS

• **Social and professional topics → Computer science education**.

KEYWORDS

assessment; rubric; programming; vignette; PCK; training

ACM Reference Format:
Svana Esche. 2024. Rubric for the Quality of Answers to Student Queries about Code. In *Proceedings of the 55th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE 2024), March 20–23, 2024, Portland, OR, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3626252.3630918

1 INTRODUCTION

The study of teaching is at the heart of research on introductory programming [19]. Teaching and those who teach, including instructors, teachers, and teaching assistants (TAs), should be considered together. However, the training and professional development of these supporters and their teaching quality are not mentioned in the above review [19]. Nevertheless, the reviews for both teachers [26] and TAs [23] show that this area has indeed been researched.

In this paper, we focus specifically on supporters' written answers and their quality to university students' queries about code. Written answers are widely used in asynchronous online courses, such as forums and emails, and are also used in written feedback on assignments. Although they are widely used, to our knowledge, there is no tool to assess the quality of answers at different levels. However, we believe that high-quality answers are important to support student learning because they are likely to build a solid, rather than superficial, knowledge base. Therefore, an assessment tool could help train supporters in computer science (CS).

Two competencies are among the relevant requirements for quality support: General Pedagogical Knowledge (GPK) and Pedagogical Content Knowledge (PCK). Both define a set of skills popularized primarily by Shulman [33], who coined these terms. GPK is generalizable to all subjects, whereas PCK is subject-specific. The concept of PCK has been influential and computer education research (CER) is not exempt from it [12]. In contrast, research on GPK, particularly on measuring GPK skills, is less developed [36]. However, GPK has been shown to be important for successful teaching [35].

Our goal is to develop an assessment tool for the quality of written answers. The tool should assess different levels for aspects of quality. We chose to use a rubric as a framework because it inherently includes multiple levels. Levels are necessary to make sound distinctions for feedback and give additional semantic information about the GPK and PCK shown. We present students' queries about the code as vignettes because they allow us to elicit GPK and PCK competencies [3]. In pedagogy, a vignette is a short, self-contained scene that depicts a realistic pedagogical situation. In this case, it is a small code snippet with an associated student query. Since we are focusing on university students, the programming language (PL) used for the code snippets should be text-based. Here, Java was chosen as a widely used PL. We also evaluate our rubric, including assessment of answers and expert opinion on perceived support. For the latter, we focus on TAs and PCK competencies as familiar areas for experts. If the tool adequately supports experts as users, it has achieved an important goal. Our research goals are therefore:

RG1. Development of a rubric that assess the quality of written answers to student queries about code.

RG2. Evaluation of the rubric by (1) assessing the quality of answers and (2) obtaining expert opinion on how it can support structured assessment of TAs' PCK competencies.

1.1 Process of Development and Evaluation

First, we developed our vignettes based on a preliminary think-aloud study we conducted. In the interviews, students expressed their thoughts on code snippets and programming tasks. Second, we developed our rubric with informal feedback loops through consultation with CER colleagues. Third, we tested our rubric internally using TA responses and then externally through expert interviews. Finally, we modified our rubric slightly to incorporate the opinions of experts. This paper includes that modified version.

2 RELATED WORK

We first outline the theoretical framework of GPK and PCK (2.1). Consistent with our focus, we examine previous uses of vignettes (Sect. 2.2) and rubrics to measure different skill levels (Sect. 2.3).

2.1 Theoretical Framework of GPK and PCK

We begin with the influential work of Shulman [32, 33]. He coined and defined the terms GPK and PCK. For Shulman, GPK was knowledge that "appear[s] to transcend subject matter" [32, p. 8]. However, he too considered GPK primarily for classroom management. In contrast, Voss et al. [36] provided a broader model for GPK. Their model also included teaching methods, classroom assessment, learning process, and individual characteristics. König et al. [16] derived a similar model, which explicitly incorporates structure. Teaching methods are particularly relevant to our focus because written answers are one method of answering student queries. However, no sub-dimensions of teaching methods appear in the previous models. In contrast, structure seems to be a relevant dimension for our rubric, as answers without structure tend to be inadequate.

We turn to the PCK, which Shulman has described as "the most useful forms of representation of those ideas, the most powerful analogies, illustrations, examples, explanations, and demonstrations" [33, p. 9]. For him, then, PCK was a subject-specific collection of ways of teaching. From this collection we derive subordinate themes for teaching methods. For example, we see the general use of 'analogies' as part of GPK, since their use is not subject-specific. The specific analogies, in turn, are themselves subject-specific, like cookie cutters and cookies for classes and objects in CS [30]. There are also other models based on Shulman, namely that of Geddis [9], as well as independent models such as that of Magnusson et al. [20]. Geddis [9] added the students' preconceptions and misconceptions and the factors that make a topic easy or difficult. From Magnusson et al. [20] only the "knowledge of instructional strategies" component is relevant to us, since we focus on teaching methods. However, its categories were specific to natural sciences, which does not include CS.

In CER, teaching methods are also included in the CS-specific competency model of the KUI project, e.g., [13]. It was one of 15 categories of the dimension "Aspects of Teaching and Learning (ATL)". There, teaching methods included organizational and methodological aspects and subject-specific teaching methods. With respect to our context, we see no directly extractable aspects for adequacy.

We conclude for the written answer as teaching method: Teaching methods are essential for both GPK and PCK models. For our context, we derive structure, analogies, illustrations, and examples as tentative rubric categories.

2.2 Use of Vignettes

Vignettes are an effective tool for assessing teachers' understanding of instructional strategies in general [14]. They also allow for the simultaneous assessment of GPK and PCK [3]. They are used in a variety of disciplines [14], including CER. CER uses vignettes explicitly [39, 40] or implicitly described as situations [27] or scenarios [34]. These studies assess PCK skills in the context of misconceptions [27, 39, 40] or debugging [34]. Research examines to identify real classroom situations to derive vignettes [28]. In these studies [27, 34, 39, 40], the vignette contains an external description, e.g., descriptions that a student has a problem and asks how to help. In this case, the student's query is not presented from their point of view. The vignettes also include code snippets [34, 39] and class diagrams [27]. The programming languages used vary from Scratch and Python [39] to Scratch alone [34] and pseudocode [27].

We focus on vignettes that explicitly include queries from the student's perspective. Likewise, we do not use block-based programming like Scratch in our university context. The only paper that does not use block-based programming uses a binary assessment [27]. This contradicts our goal of assessing various skill levels. Thus, neither the vignettes nor the assessments can be used here.

2.3 Rubrics

Rubrics are tools that inherently provide multiple levels: "A rubric is a coherent set of criteria for students' work that includes descriptions of levels of performance quality on the criteria." [2, p. 4].

As far as we know, there is no rubric in the CER for assessing the various levels of PCK or GPK. The closest thing we have to a rubric is the KETTI project [7] competency model for PCK. Their model is based on the KUI group's model [13], but is explicitly tailored to TAs. In additional materials, the KETTI project provides more detailed descriptions on one level, namely the ideal level [31].

In contrast, there are PCK rubrics for all science subjects, e.g., for physics [5] and chemistry [21]. Their categories derive from the theoretical frameworks described earlier in Sect. 2.1.

We focus on their levels as a characteristic feature of rubrics. The levels range from three-point [5] to four-point scales [21]. For example, the three-point scale indicates whether the corresponding category is included in none, some, or all cases. We have found no theoretical framework for these levels. Thus, we conclude that the distinctions and descriptions of the levels are based on experience.

The above rubrics refer to contexts other than those of our study, which deals with written answers. Therefore, these rubrics cannot serve as an answer to our research goals. Furthermore, we are not aware of any rubric that assesses GPK competencies at multiple levels. This underscores the need to develop a separate rubric.

3 RUBRIC

3.1 Theoretical Framework

We begin with the context (3.1.1). The application to the theoretical framework (2.1) leads to the categories (3.1.2) and levels (3.1.3).

3.1.1 *Context of Answers.*

We focus on written answers to student queries related to programming tasks, especially queries asked in vignettes. These vignettes contain a student's query or problem based on a code snippet or programming task. The vignettes do not contain background information about the institution, standards, or curriculum. The code snippet or programming task is given, but the teaching methods of the course are unknown. Thus, information about the student is provided only by the query asked. Two sample vignettes are given in Table 1, with the third vignette omitted for space reasons. The assessment can only be based on the written answers. It cannot be based on the inner thoughts, reflections and consciousness of the author. These inner processes include the important PCK competencies that Geddis [9] included in their model, namely, awareness of misconceptions and students' prior conceptions. None of these competencies can be derived from the written answers and therefore are not part of our rubric.

3.1.2 Categories. First, we adopt the concept of structure from the model of König et al. [16]. We divide this concept into *coherence* and *meta-level explanations*. Coherence means that the answer has an internal structure. Meta-level explanations are explicit overview descriptions of what is done next in the answer. Thus, they make the internal structure explicit. We take coherence as a fundamental category. Coherence is the absolute minimum that a written answer must have. Similarly, we also take *completeness* as a fundamental category. An incoherent and incomplete answer is not adequate and therefore does not meet the basic requirements. We call these two categories *basic categories*. All other categories are not necessary in the sense of an absolute minimum, called *additional categories*.

Second, we adopt the categories from Shulman's [33] definition of PCK, namely illustrations, analogies, and examples. For these, we prefer to list references that belong to CER. We do this to highlight CS-specific research and the relevance of the category to CS.

We add illustrations, but call them *multiple representations*. With this naming we emphasize the importance of additional representations besides text and code. The use of multiple representations has a long tradition in general education [4] and in CER [25]. We add examples directly with their original name. They also have a long tradition at CER [25], which continues today [24]. The widespread use of examples by teachers and textbooks shows that "examples are a critically important part of learning to program" [18]. We add analogies, but call them *metaphors*, analogous to the usage in CS. Large portions of the technical language in CS are themselves metaphors [6]. Metaphors are used frequently [30]. However, they are less explicitly explored, with exceptions, e.g., [1, 30, 38].

Third, we include two other categories. They have a common language focus. Language is important because it is both a tool for thinking and for communication [11]. The former was already addressed by Vgyotsky [37] and is still relevant for teaching today [15]. Lemke [17] argues that talking about science is doing science through the medium of language. These considerations demonstrate the importance of considering language in teaching and learning. For CER, Diethelm et al. [8] give suggestions on how to incorporate language considerations in CS classrooms. However, language does not yet play a role in the earlier models of GPK and PCK.

We consider *student language* as a category embedded in the broader concept of classroom language. Teaching "should explicitly consider and address [...] the language prerequisites of the students" [15, p. 186]. At CER, research on language, particularly classroom language, is underdeveloped [8]. In our case, there are two ways to address and integrate student language: (1) starting the answer with the student's language, (2) or relating the answer back to the student's language. This integration aims to support better connection and understanding of the answer.

The last language category links the code and its used language. This idea of *linking* is based on research in mathematics education. There, symbols are linked and related to the language used, e.g., [29]. We give an example of how this linking might look in programming. Consider the following task: "Write a program that prints the sum from 1 to n for all numbers n from 1 to 50." Experienced programmers intuitively know that for tasks with the scheme "for all x, do this" a loop is a possible solution to the task, just as "if [...] else" refers to conditional statements. In languages other than English, this is not trivial. There, these words do not directly correspond to the keywords used in the programming language. This knowledge about linkages can be taught explicitly. In addition, this category contains explicit description terms for symbols such as the assignment character for = in Java. In this way, the described linkage is a scaffolding approach.

3.1.3 Levels. Different levels offer greater potential than a binary assessment. In particular, level descriptions allow qualitative feedback on how to improve and what that might look like. This idea is based on Hattie's [10] model of feedback, which consists of feed up (the ideal), feed back (the is-state), and feed forward (the next step).

For levels, we first discern whether the aspect of the category is included. If it is not, we assess Level 1 as the lowest level. The mere integration of the aspect refers to the GPK competencies. For example, the mere integration of a metaphor belongs to the GPK because it is not subject-specific. However, the incorporation of an adequate version is subject-specific and therefore belongs to the PCK. In the case of our example, a supporter with sufficient PCK competencies knows the limitations and pitfalls of containers as variables. Misleading or implicit uses thus belong to Level 2. If all uses are adequate, they represent Level 3 as the highest level.

3.2 Presentation

The rubric consists of two basic categories (B) and six additional categories (A), each with three levels, abbreviated L1, L2, and L3. For each category, *the author demonstrates competence in formulating a written answer of adequate quality by [...]*

*B-I. [...] formulating a **coherent** answer, i.e., a semantically meaningful text whose parts are logically connected.*
L1 The answer is not coherent. There are jumps between different topics between (almost) all the individual sentences.
L2 The answer is partially coherent. There are jumps between topics between some individual sentences.
L3 The answer is coherent, with no jumps between topics.

*B-II. [...] formulating a **complete** answer, i.e., fully explaining the concept associated with the student's query and/or fully addressing the specific query or problem.*
L1 The answer is incomplete: It consists of merely referring to (lecture) material or general problem-solving strategies or merely stating the code's solution to the problem.
L2 The answer is partially complete: It explains the related concept of the student's query or problem and/or addresses the specific query or problem. But these answers or addresses are only partially complete, because they omit important parts.
L3 The answer is complete: Except for negligible details, it fully explains the concept associated with the student's query or problem, and/or it fully addresses the specific query or problem.

A-I. [...] integrating ***meta-level explanations****, i.e., giving an overview of each part of the answer as a meta-level explanation. Only after the overview is given, that part of the answer is further elaborated.*
L1 There are no meta-level explanations.
L2 There are meta-level explanations, but not before each part.
L3 There are meta-level explanations before each part.

A-II. [...] integrating ***multiple representations****, i.e., another supporting medium in addition to text and code, such as a trace table.*
L1 There are no multiple representations.
L2 The multiple representations included are misleading. For example, there are technically incorrect aspects, missing labels, or unexplained abbreviations.
L3 The multiple representations included are not misleading.

A-III. [...] including ***concrete examples****, i.e., an example whose terms are assigned concrete values, such as 5 for a variable. In contrast, an example is not concrete if the example uses only terms at an abstract level without associating those terms with concrete values.*
L1 The answer does not contain concrete examples.
L2 The answer contains at least one specific example. Even so, it is unrelated to the general concept before, after, or parallel to it.
L3 The answer contains at least one concrete example that is linked to the general concept. The example could be realized before, after or in parallel with the general concept.

A-IV. [...] using ***metaphors****, analogous ideas with which students are already familiar, to explain a topic in the answer.*
L1 There are no metaphors.
L2 The metaphors included are misleading. For example, the area of origin is not common knowledge or the use of the metaphor is limited, but these limitations are not addressed in the answer.
L3 The metaphors included are not misleading.

A-V. [...] incorporating the ***student's language****, i.e., considering the expressions and terms used by the student and explicitly including them in the answer by referring to them. In this way, the answer docks with the student's language.*
L1 The answer does not include the student's language.
L2 The answer implicitly includes the student's language by using the same expressions or terms, but without addressing the fact that they come from the student's language.
L3 The answer explicitly includes the student's language by using the same expressions or terms and by addressing the fact that they come from the student's language.

A-VI. [...] ***linking the language to the programming language****, i.e., the explicit linking of the language used for the description with its counterpart in the programming language (PL). However, the description of the execution of, for example, an if statement does not belong here if there is no explicit connection between the term and the concept in the PL. For example, the term 'for all' is associated with a loop and 'if' with a condition. In addition, linking also includes explicit descriptions of symbols or keywords of the PL.*
L1 There is no link between the language and the PL.
L2 The answer implicitly establishes a link by inserting typical terms, such as 'until' to explain a conditional loop.
L3 The answer explicitly establishes a link.

3.3 Context of Rubric Usage

We see the use of the rubric in the context of professional development (PD). Here we include all who support students in post-secondary education, such as instructors and teaching assistants. During PD, supporters answer students' queries about the code, presented in the form of vignettes. The rubric allows for an assessment of the quality of the answers Thus, the purpose of the rubric is to base the feedback that supporters receive on the assessment of the rubric. In this way, we aim to provide a more solid basis for feedback to supporters. This feedback includes how to improve and what that might look like. This context does not exclude other uses, although they were not our main context. Other uses include plenary discussion during PD or classroom observation.

4 EVALUATION

4.1 Internal with Teaching Assistants

We used the rubric as a final assessment after a two-day TA training workshop specifically tailored for TAs supporting the CS1 course. The content of the workshop included (mis)conceptions of typical CS1 topics such as variables, loops, and objects. It also covered strategies for explaining these topics using various representations.

We collected 85 written answers from n=30 TAs, which were based on three vignettes. All TAs gave their informed consent to use their data and participated anonymously. In this case, the local ethics board did not require review based on local regulations. The vignettes addressed variable swap, conditional statements, and loops. The prompt was: *"Formulate a response to the student's request for help. The text box below is available for this purpose. You can use your own software to write text and create graphics."*

To give an example, we apply the rubric to two excerpts, which we list with their vignettes in Table 1. In E1, we see no jumps between topics between sentences, leading to level 3 for B-I. At level 3 for B-II, the answer must explicitly address the student's query, which is here: "That's pointless, isn't it?". The TA does not address this query, but provides a complete execution of the code. Thus, we assign level 2. The second sentence is a meta-level explanation of what comes next. E1 does not use them before each part, leading to level 2 for A-I. E1 contains only text and code, but no multiple representations. Thus, we award level 1 for A-II. The TA uses concrete values and explains in parallel each line of code in general and its effect on the concrete values. Thus, we assign level 3 for A-III. We assess the textual mention of "stickers" as a metaphor, but not as a multiple representation. For the latter, a medium other than text and code must be used. The sticker metaphor is misleading because the boxes permanently carry the variable name in the form of a sticker. This metaphor cannot be extended to reference data types either. Thus, we assign level 2 for A-IV. Neither is the student's language addressed, nor is there a link between the language and the programming language (PL), leading to level 1 for A-V and A-VI. In contrast, student language is explicitly addressed in E2. The TA introduces it by inserting "You say" and indicating that the statement that follows is from the student. The TA also explicitly links the descriptions to the PL keywords by asking the student, e.g., "What is the 'if'?". Thus, we assign level 3 for A-V and A-VI.

Next, we summarize the results for all 85 written answers that were scored by the author. Table 2 lists the percentage of levels

Table 1: Vignettes for the variable swap and conditional statements with excerpts from corresponding written answers.

Vignettes		Excerpts from Written Answers, written by TAs

V1: "I do not understand the meaning of the code. Overall, a=a and b and c also have value a. That's pointless, isn't it?"	`// a,b,c are of type int`
	`c = b;`
	`b = a;`
	`a = c;`
		E1: *"Well, it's not that simple. Go through it with numbers. Let's say a=1, b=2, c=3. Now in the first step c=b, that means that c=2 is valid, because b=2. [...] The variable names a, b, c should be visualized like stickers, which are attached to a box. In this case the content of the box is always a number (e.g. 1,2,3). If we say c=b, then we take the sticker of b and attach it to the box to which sticker c is attached.*

V2: In the code below, I would have said that if k is 5, block A is executed, otherwise B is executed. Now a friend told me that is wrong. This has me confused. I don't understand what is wrong with it."	`// k is of type int`
	`if (k == 5) {`
	` // statement block A`
	`}`
	`// statement block B`
		E2: *"Try to imagine what you are reproducing or what language elements you are using to convert the given code into natural language. You say, IF k is equal to 5, block A is executed, OTHERWISE B. What is the "if", and what is the "otherwise" keyword in the programming language? And are they both present in this code? [...]"*

Table 2: Scored levels (L) for all categories in n=85 answers.

L	B-I	B-II	A-I	A-II	A-III	A-IV	A-V	A-VI
1	13%	16%	75%	86%	59%	86%	47%	53%
2	38%	40%	19%	4%	11%	14%	15%	29%
3	49%	44%	6%	11%	31%	0%	38%	18%

for each category. The percentage of level 1 ratings in the basic categories was low, 13% and 16%, resp. In the author's opinion, the percentages are too high for TAs to adequately support students because incoherent or incomplete answers are not adequate. There were also categories where more than half of the answers were level 1: meta-level representations (A-I), multiple representations (A-II), metaphors (A-IV), and linking language to PL (A-VI). In contrast, about one-third contain examples and/or address the student's language at the highest level. We take these results as feedback for the TA workshop to improve, especially in the low-rating categories.

4.2 External with Experts

We interviewed experts who were mainly CER instructors, but also CS teachers with CER experience. They were selected based on their experience with PCK in research or teaching. We invited 27 experts individually by e-mail, 13 of whom participated. They came from four countries, three in Europe and one in North America. All gave consent for their data to be used and participated anonymously. As before, no review by the local ethics committee was required. We interviewed two additional experts to test the interview process.

We used a subset of six responses from our total set of 85 responses, as described above. This subset covers all three vignettes. During the interview, each expert was presented with a vignette and an accompanying written response from TA. They were asked to rate the PCK of TA using the think-aloud method. The experts then re-evaluated the written answer again using the rubric. Follow-up questions included (1) general experiences with the rubric, (2) how the rubric supported them, and (3) possible areas of application.

We analyzed the data using deductive-inductive qualitative content analysis [22]. Predefined themes were concrete positive and negative aspects and areas of application. We inductively formed new categories and revised these after five interviews. We derived the main categories by abstracting from the categories after completing all interviews. For inter-coder agreement, another team member coded the same data using the previous categories. We then checked the agreement qualitatively, as recommended [22]. There was excellent agreement on the negative aspects and good agreement on the other two. Disagreements were resolved through consensus discussions.

As a result, we derived 58 categories, 16 for the positive aspects, 27 for the negative aspects, and 15 for the areas of application. For the positive aspects, we formed five main categories: (P1) analytical breakdown of the construct PCK with 92%, (P2) support for assessment with 76%, (P3) supporting layout and features with 46%, (P4) support for reflection with 38%, and (P5) support for writing an answer with 15%. We present the three most frequently mentioned categories. Each category was mentioned by six or seven experts. Expert E8 stated: "I think that can really help to proceed analytically." This excerpt was coded as *highlighting the various PCK aspects for analytical approach (P1-1)*, which also belongs to P1, which focuses on the analytical division. For expert E3, the rubric "helps as support [...] that one forgets nothing". We coded it as *highlighting the various PCK facets so as not to overlook them (P1-2)*. For expert E9, the rubric would "provide some form of structuring opportunity", coded as *structuring the assessment (P2-1)*.

For the negative aspects, we formed six main categories: (N1) improvement of layout and features with 84%, (N2) missing PCK categories with 76%, (N3) improvement of level-design with 46%, (N4) risk of sticking to the given schema with 38%, and (N5) limited focus and (N6) training effort with 30% each. The two most frequently mentioned categories were *improvement of individual statements required (N1-1)* and *completeness of PCK category as an open question (N2-1)*, mentioned by eight and six experts, respectively. For N1-1, for the expert E4 "some things would have to be just a little more precise". For N2-1, expert E3 was wondering "if this is complete". Of the total 27 categories, 22 were mentioned by only one or two experts. Thus, most of them were not apparent or relevant to most experts.

For the areas of application, we formed five main categories. (A1) feedback and reflection and (A2) transfer to other disciplines and addressees with 61% each, (A3) learning activities with 46%, (A4) extended context of the answers with 38%, and (A5) planning with 23%. Exemplifying reflection (A1), expert E1 stated: "I wouldn't just use it to assess someone, but it can also be given to the students so that they can reflect on their own practice." In summary, experts addressed four dimensions: *activities* for which the rubric is used (e.g., reflection, planning), *context* in which the rubric is used (e.g., lesson observations), *addressees* for whom the rubric is used, and *discipline* in which the rubric is used, such as mathematics.

5 DISCUSSION

Lessons learned of what did work. In general, we met our research goals of developing and evaluating our rubric. In developing the rubric, we drew on previous theoretical frameworks for categories and levels. Indeed, this deductive approach worked well and we would use it again in developing similar rubrics. Our two-pronged approach to rubric evaluation also worked well. First, we assessed 85 written answers from 30 TAs and provided examples of rubric use. In this way, we had a broad and diverse range of answers to which we could apply the rubric. Second, the expert interviews revealed a variety of positive and negative aspects as well as areas of application. By including them, we refined wording. Overall, the expert opinions contributed to a rich picture of evidence on how the rubric can be used and the possibilities it offers. For example, to the added values of our tool belong the provision an analytical breakdown and not to overlook aspects. Expert interviews revealed that the context of use is even wider than we had assumed, e.g., as an aid to reflection and planning. Their opinions also showed the ways in which the rubric is of interest to practitioners.

Lessons learned of what did not work. Despite our efforts and multiple iterations, it is difficult to provide clear and sound formulations. Thus, we had to fine-tune the descriptions after the expert interviews. The rubric also could not cover all aspects of quality. The experts suggest including other PCK aspects as well. These include, for example, the teaching of strategies and the appropriateness of the language used in answering. However, we believe that our rubric provides a valuable baseline tool for assessing response quality. We encourage other practitioners to supplement the rubric with their own categories for their context of use. We also aimed to ensure that the rubric can be used without training. However, the expert evaluation showed that training is indeed needed. We hope that the rubric will still be valuable despite the need for training.

Novelty of our tool. In CER, there are no other tools that assess the quality of GPK and PCK or PCK alone at different levels. However, quality assessment is an essential part of training. It allows for informed feedback for both supporters and those who delivered the training. These aspects also make our tool a relevant contribution to CER. For comparison, the closest thing we can use is the KETTI project's collection of descriptions [31]. Both their categories and those of our rubric are based on previous theoretical frameworks. However, their categories are based on the competency model of the KUI group [13] and explicitly refer to the PCK of teaching assistants. We, on the other hand, refer to all individuals who support students, regardless of their role. In addition, our rubric differs from their tool in that we provide for multiple levels rather than referring to one level as the ideal level. The ideal level describes the goal or, in Hattie's [10] terms, the feed up. In describing the feed up, however, it is not clear which steps are on the way to the goal. Our rubric contains descriptions for these intermediate steps.

Direct use. Our tool allows other practitioners to use it directly. Direct use is facilitated by the following aspects: First, the rubric is listed in its entirety in the paper, see Section 3.2. Second, as noted by the experts, some training effort is required, but the brevity of the rubric reduces this effort. To compensate for the training effort, we have also shown examples of the use of the rubric, see Section 4.1. This demonstration of use is intended to show the considerations we make when using the rubric. Third, the rubric, with its categories and levels, is not based on any particular programming language. Even though the code snippets presented relate to Java, they can easily be applied to other programming languages. Finally, no additional equipment or software is required. Thus, there are no costs involved and no compatibility with existing systems to consider. Overall, direct use is open to all practitioners.

Limitations. We see two general limitations: the artificial setting and the small datasets for evaluation. First, in real situations, TAs as supporters might have answered differently. However, creating real situations for training requires more effort, both in terms of time and human resources. In addition, high levels on the rubric do not necessarily equate to high quality teaching in practice. Our assessment is only a snapshot at a point in time and may not be representative of support skills as a whole. Second, the number of written responses assessed, 85, is not overwhelming. As they were all written by TAs from the same university, the evaluation results may not transfer to other supporters. For experts, we included experts from different continents to mitigate this limitation. We could have included more experts. However, most categories, 53 of the total 58, were identified in the first nine interviews. This indicates theoretical saturation and a sufficient sample size.

Future Research. Our rubric could serve as a general framework for developing topic-specific rubrics. As an example of such a rubric, one might focus on loops, since they have special requirements for appropriate representations. A flowchart should include a backward arrow to illustrate the iterative aspect. A special rubric could also distinguish between appropriate and less appropriate metaphors. Also open is the question of how many additional categories an answer should contain so that students still find the inclusion helpful.

6 CONCLUSION

This tool paper describes a rubric as a measurement tool to assess the quality of supporter responses to student questions about code. We included both the theoretical framework behind the rubric and the rubric as a whole. This included all descriptions of the different categories and levels of the rubric. We also presented the results of the evaluation, both of using the rubric for assessment and of interviewing experts. We believe that other practitioners will benefit from using the rubric in training supporters. We also encourage practitioners and researchers to use the tool as a foundation and add their own categories for their context and trainings.

REFERENCES

[1] Briana Bettin, Linda Ott, and Julia Hiebel. 2022. Semaphore or Metaphor? Exploring Concurrent Students' Conceptions of and with Analogy. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE '22)*. ACM, New York, NY, USA, 200–206. https://doi.org/10.1145/3502718.3524796

[2] Susan M. Brookhart. 2013. *How to Create and Use Rubrics for Formative Assessment and Grading.* ASCD, Alexandria, Virginia.

[3] Dorothee Brovelli, Katrin Bölsterli, Markus Rehm, and Markus Wilhelm. 2014. Using Vignette Testing to Measure Student Science Teachers' Professional Competencies. *American Journal of Educational Research* 2, 7 (2014), 555–558. https://doi.org/10.12691/education-2-7-20

[4] Jerome S. Bruner. 1967. *Toward a Theory of Instruction.* Belknap Press of Harvard University, Cambridge, Massachusetts.

[5] Coréne Coetzee, Marissa Rollnick, and Estelle Gaigher. 2020. Teaching Electromagnetism for the First Time: a Case Study of Pre-service Science Teachers' Enacted Pedagogical Content Knowledge. *Research in Science Education* 52 (2020), 357–378. https://doi.org/10.1007/s11165-020-09948-4

[6] T. R. Colburn and G. M. Shute. 2008. Metaphor in computer science. *Journal of Applied Logic* 6, 4 (2008), 526–533. https://doi.org/10.1016/j.jal.2008.09.005

[7] Holger Danielsiek, Peter Hubwieser, Johannes Krugel, Johannes Magenheim, Laura Ohrndorf, Daniel Ossenschmidt, Niclas Schaper, and Jan Vahrenhold. 2017. Kompetenzbasierte Gestaltungsempfehlungen für Informatik-Tutorenschulungen. In *INFORMATIK 2017*, Maximilian Eibl and Martin Gaedke (Eds.). Gesellschaft für Informatik, Bonn, 241–254. https://doi.org/10.18420/in2017_18

[8] Ira Diethelm, Juliana Goschler, Timo Arnken, and Sue Sentance. 2023. Language and Computing. In *Computer Science Education. Perspectives on Teaching and Learning in School* (2 ed.), Sue Sentance, Erik Barendsen, Nicol R. Howard, and Carsten Schulte (Eds.). Bloomsbury Publishing, London, 167–182.

[9] Arthur N. Geddis. 1993. Transforming subject-matter knowledge: the role of pedagogical content knowledge in learning to reflect on teaching. *International Journal of Science Education* 15, 6 (1993), 673–683. https://doi.org/10.1080/0950069930150605

[10] John Hattie and Helen Timperley. 2007. The power of feedback. *Review of educational research* 77, 1 (2007), 81–112.

[11] Vivien Heller and Miriam Morek. 2015. Academic discourse as situated practice: An introduction. *Linguistics and Education* 31 (2015), 174–186. https://doi.org/10.1016/j.linged.2014.01.008

[12] Aleata Hubbard. 2018. Pedagogical content knowledge in computing education: A review of the research literature. *Computer Science Education* 28, 2 (2018), 117–135. https://doi.org/10.1080/08993408.2018.1509580

[13] Peter Hubwieser, Johannes Magenheim, Andreas Mühling, and Alexander Ruf. 2013. Towards a Conceptualization of Pedagogical Content Knowledge for Computer Science. In *Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (ICER '13)*. ACM, New York, NY, USA, 1–8. https://doi.org/10.1145/2493394.2493395

[14] Carolyn Jeffries and Dale W. Maeder. 2005. Using Vignettes To Build and Assess Teacher Understanding of Instructional Strategies. *The Professional Educator* 27 (2005), 17–28.

[15] Sebastian Kempert, Lennart Schalk, and Henrik Saalbach. 2019. Übersichtsartikel: Sprache als Werkzeug des Lernens: Ein Überblick zu den kommunikativen und kognitiven Funktionen der Sprache und deren Bedeutung für den fachlichen Wissenserwerb. *Psychologie in Erziehung und Unterricht* 66, 3 (2019), 176–195. https://doi.org/10.2378/peu2018.art19d

[16] Johannes König, Sigrid Blömeke, Lynn Paine, William H. Schmidt, and Feng-Jui Hsieh. 2011. General Pedagogical Knowledge of Future Middle School Teachers: On the Complex Ecology of Teacher Education in the United States, Germany, and Taiwan. *Journal of Teacher Education* 62, 2 (2011), 188–201. https://doi.org/10.1177/0022487110388664

[17] Jay L Lemke. 1990. *Talking science: Language, learning, and values.* Ablex Publishing Corporation, Norwood, New Jersey.

[18] Andrew Luxton-Reilly, Paul Denny, Diana Kirk, Ewan Tempero, and Se-Young Yu. 2013. On the Differences between Correct Student Solutions. In *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '13)*. ACM, New York, NY, USA, 177–182. https://doi.org/10.1145/2462476.2462505

[19] Andrew Luxton-Reilly, Simon, Ibrahim Albluwi, Brett A. Becker, Michail Giannakos, Amruth N. Kumar, Linda Ott, James Paterson, Michael James Scott, Judy Sheard, and Claudia Szabo. 2018. Introductory Programming: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion)*. ACM, New York, NY, USA, 55–106. https://doi.org/10.1145/3293881.3295779

[20] Shirley Magnusson, Joseph Krajcik, and Hilda Borko. 1999. Nature, Sources, and Development of Pedagogical Content Knowledge for Science Teaching. In *Examining Pedagogical Content Knowledge: The Construct and its Implications for Science Education*, Julie Gess-Newsome and Norman G. Lederman (Eds.). Springer Netherlands, Dordrecht, 95–132. https://doi.org/10.1007/0-306-47217-1_4

[21] Elizabeth Mavhunga and Marissa Rollnick. 2013. Improving PCK of Chemical Equilibrium in Pre-service Teachers. *African Journal of Research in Mathematics, Science and Technology Education* 17, 1_2 (2013), 113–125. https://doi.org/10.1080/10288457.2013.828406

[22] Philipp Mayring. 2021. *Qualitative Content Analysis: A Step-by-Step Guide.* Sage, London.

[23] Diba Mirza, Phillip T. Conrad, Christian Lloyd, Ziad Matni, and Arthur Gatin. 2019. Undergraduate Teaching Assistants in Computer Science: A Systematic Literature Review. In *Proceedings of the 2019 ACM Conference on International Computing Education Research (ICER '19)*. ACM, New York, NY, USA, 31–40. https://doi.org/10.1145/3291279.3339422

[24] Kasia Muldner, Jay Jennings, and Veronica Chiarelli. 2022. A Review of Worked Examples in Programming Activities. *ACM Trans. Comput. Educ.* 23, 1, Article 13 (dec 2022), 35 pages. https://doi.org/10.1145/3560266

[25] B. A. Myers. 1986. Visual Programming, Programming by Example, and Program Visualization: A Taxonomy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '86)*. ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/22627.22349

[26] Lijun Ni, Gillian Bausch, and Rebecca Benjamin. 2023. Computer science teacher professional development and professional learning communities: a review of the research literature. *Computer Science Education* 33, 1 (2023), 29–60. https://doi.org/10.1080/08993408.2021.1993666

[27] Laura Ohrndorf and Sigrid Schubert. 2013. Measurement of Pedagogical Content Knowledge: Students' Knowledge and Conceptions. In *Proceedings of the 8th Workshop in Primary and Secondary Computing Education (WiPCSE '13)*. ACM, New York, NY, USA, 104–107. https://doi.org/10.1145/2532748.2532758

[28] Ursula Pieper and Jan Vahrenhold. 2020. Critical Incidents in K-12 Computer Science Classrooms - Towards Vignettes for Computer Science Teacher Training. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education (SIGCSE '20)*. Association for Computing Machinery, New York, NY, USA, 978–984. https://doi.org/10.1145/3328778.3366926

[29] Susanne Prediger and Lena Wessel. 2013. Fostering German-language learners' constructions of meanings for fractions—design and effects of a language-and mathematics-integrated intervention. *Mathematics Education Research Journal* 25, 3 (2013), 435–456. https://doi.org/10.1007/s13394-013-0079-2

[30] Joseph P. Sanford, Aaron Tietz, Saad Farooq, Samuel Guyer, and R. Benjamin Shapiro. 2014. Metaphors We Teach By. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14)*. ACM, New York, NY, USA, 585–590. https://doi.org/10.1145/2538862.2538945

[31] Niclas Schaper, Alexander Brune, Jan Vahrenhold, Johannes Magenheim, Peter Hubwieser, and Daniel Ossenschmidt. 2022. *KETTI: Kompetenzerwerb von Tutorinnen und Tutoren in der Informatik Kompetenzmodell.* Technical Report. KETTI. https://www.uni-muenster.de/imperia/md/content/ketti/ketti-kompetenzmodell.pdf

[32] Lee Shulman. 1987. Knowledge and teaching: Foundations of the new reform. *Harvard educational review* 57, 1 (1987), 1–23.

[33] Lee S Shulman. 1986. Those who understand: Knowledge growth in teaching. *Educational researcher* 15, 2 (1986), 4–14.

[34] Jennifer Tsan, David Weintrop, and Diana Franklin. 2022. An Analysis of Middle Grade Teachers' Debugging Pedagogical Content Knowledge. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE '22)*. ACM, New York, NY, USA, 533–539. https://doi.org/10.1145/3502718.3524770

[35] Hannah Ulferts. 2019. The relevance of general pedagogical knowledge for successful teaching: Systematic review and meta-analysis of the international evidence from primary to tertiary education. *OECD Education Working Papers* 212 (2019). https://doi.org/10.1787/ede8feb6-en

[36] Thamar Voss, Mareike Kunter, and Jürgen Baumert. 2011. Assessing teacher candidates' general pedagogical/psychological knowledge: Test construction and validation. *Journal of educational psychology* 103, 4 (2011), 952. https://doi.org/10.1037/a0025125

[37] James V. Wertsch. 1990. Dialogue and dialogism in a socio-cultural approach to mind. In *The dynamics of dialogue*, Klaus Marková, Ivana; Foppa (Ed.). Harvester Wheatsheaf, New York, London, 62–82.

[38] John Woollard. 2005. The Implications of the Pedagogic Metaphor for Teacher Education in Computing. *Technology, Pedagogy and Education* 14, 2 (2005), 189–204. https://doi.org/10.1080/14759390500200201

[39] Aman Yadav and Marc Berges. 2019. Computer Science Pedagogical Content Knowledge: Characterizing Teacher Performance. *ACM Trans. Comput. Educ.* 19, 3, Article 29 (May 2019), 24 pages. https://doi.org/10.1145/3303770

[40] Aman Yadav, Marc Berges, Phil Sands, and Jon Good. 2016. Measuring Computer Science Pedagogical Content Knowledge: An Exploratory Analysis of Teaching Vignettes to Measure Teacher Knowledge. In *Proceedings of the 11th Workshop in Primary and Secondary Computing Education (WiPSCE '16)*. ACM, New York, NY, USA, 92–95. https://doi.org/10.1145/2978249.2978264

10. Application Study III: Development and Validation of the Natural Language Computing Test (NLCT)

Bibliographic Information

This contribution is based on the following publication, the version below being the camera-ready version.

Reprinted, with permission, from:

Svana Esche. 2024. Testing Programming Aptitude through Commonsense Computing. In *Proceedings of the 26th Australasian Computing Education Conference (ACE '24)*, January 29-February 2, 2024, Sydney, NSW, Australia. Association for Computing Machinery, New York, NY, USA, 104-113. `https://doi.org/10.1145/3636243.3636255`

Testing Programming Aptitude through Commonsense Computing

Svana Esche
svana.esche@tu-darmstadt.de
Technical University of Darmstadt
Darmstadt, Hessen, Germany

ABSTRACT

Background. Programming aptitude tests are of interest since the beginning of computing education research. Many novices have no experience with programming languages before their first course. Yet they have different levels of commonsense computing.

Research Question. *How successful is a commonsense computing test based on natural language as a programming aptitude test?*

Method. We developed the Natural Language Computing Test (NLCT) as such a test. Our quantitative data consisted of CS1 students (N=681) who completed the NLCT during the winter 2022/23 semester. We analyzed our test with three methods. These were inter-rater agreement, item response theory, and appropriateness as predictive factor for student success.

Findings. The NLCT performed well in terms of inter-rater agreement and accuracy, according to item response theory analysis. However, the test was a weak predictor of student success as measured by correlation.

Implications. A test based solely on natural language can succeed as a programming aptitude test. Thus, a programming aptitude test need not be based on prior knowledge of programming languages or related sciences such as mathematics. However, iterative improvement of the developed test is warranted so that it can be used with less personnel effort.

CCS CONCEPTS

• **Social and professional topics → Student assessment**.

KEYWORDS

aptitude; assessment; CS1; item response theory; predict

ACM Reference Format:
Svana Esche. 2024. Testing Programming Aptitude through Commonsense Computing. In *Australian Computing Education Conference (ACE 2024), January 29-February 2, 2024, Sydney, NSW, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3636243.3636255

1 INTRODUCTION

Novices differ in many ways. A frequently studied feature of heterogeneity is prior experience with programming languages. A significant proportion of novices have no prior experience with programming languages. The proportion varies in the studies between 36% [8] (data provided by the authors), 50% [15], and 68% [42].

However, the lack of prior experience does not mean that these novices do not have the thought structures necessary for programming. These thought structures can be described with the term "commonsense computing" [7]. Chen et al. [7] defined this term as "what students know about computing concepts before having formal instruction". They also started a research project with six episodes on this topic, e.g., [39]. In their research project, they investigated the "natural resources students bring to computer science" [7]. Since we focus on programming, commonsense computing knowledge is examined in terms of programming knowledge and Boolean logic. Boolean logic is important for conditionals and is part of commonsense computing [39]. Thus, commonsense computing is defined here as *the programming and logic knowledge that students bring with them before they receive formal instruction.*

Chen et al. [7] showed that 57% of novices who had no experience with programming languages actually had commonsense computing knowledge, as measured by correctness. Novices express their commonsense computing knowledge through natural language. In contrast to programming language, all novices have prior experience with natural language, although the skills are likely to vary. In addition to general natural language skills, novices probably differ in their ability to express themselves in natural language when it comes to demonstrating their commonsense computing knowledge.

In general, the combination of thought and language and the connection between them was already described by Vygotksy [40]. To the same extent, linguists describe language "as a tool of thinking" [13], thus also expressing the connection between thought and language. Commonsense computing is a specific part of thinking.

In this study, we assume that the extent to which novices employ commonsense computing knowledge provides information about their aptitude for acquiring programming language skills. According to Merriam Webster's dictionary, an aptitude test is "a standardized test designed to predict an individual's ability to learn certain skills." [26]. Here, these skills correspond to programming language skills.

Since the 1950s, aptitude tests for programming have been of interest in computing education research [33]. However, recent studies have also addressed how to create and validate aptitude tests for programming [11, 19, 23, 25, 32, 36]. Most of these tests involve related skills such as mathematics and logic [32], or the use of predictions about time and state [19]. With our focus on natural language, only three tests come close. However, two of them, [23, 25], are not standardized as required by the definition of aptitude tests.

The one remaining test, the PAT [16, 36, 38], uses natural language in only two of a total of five tasks. Thus, there are validated tests for non-language-based tests, but not for commonsense computing tests that use only natural language.

We test how successful our new test will be by showing empirically how accurate it is and how accurately it measures. For the former, we examine inter-rater agreement. For the latter, we perform an item response theory (IRT) analysis [27]. Briefly, IRT is a statistical method for analyzing test data to assess the reliability and difficulty of individual test items. Its advantages include greater accuracy in measuring individual differences, more efficient item selection, and greater precision in assessing student ability. We are also investigating our new test in terms of its suitability as a predictor of student success in CS1.

1.1 Research Intent

In our view, the construct *commonsense computing* can be measured with a standardized test. Here we adopt the view of positivism. Our goal is therefore to develop and analyze a programming aptitude test that measures this construct using natural language tasks. We call this test *Natural Language Computing Test (NLCT)*. Our general research question is:

(RQ) *How successful is a commonsense computing test based on natural language as a programming aptitude test?*

Based on the previous considerations, we concretize our research question by composing the following three questions: *How does the NLCT perform in terms of ...*

(RQ1) *... evaluation based on inter-rater agreement?*
(RQ2) *... evaluation based on item response theory?*
(RQ3) *... being a predictive factor for student success in CS1?*

2 STATE OF THE ART

In this study, we develop and examine a new programming aptitude test. We give an overview of previous tests (Sect. 2.1) and success factors in general (Sect. 2.2). How the contributions of our study are located there is discussed in Sect. 6.1 and 6.2.

2.1 Programming Aptitude Tests

We focus on standardized measurement of novices' programming aptitude. The tests considered here do not assume any knowledge of programming languages. They are also explicitly targeted at novices in introductory programming courses in post-secondary education, in short CS1. Our focus differs from the related area of assessing computational thinking (see the review by Tang et al. [37]). The measures used there target elementary and secondary school students, a different target group than CS1. Surveys of prior programming experience, such as the programming language learned and lines of the longest program [17], are not our topic, nor are programming aptitude tests for business and professional applications.

In total, we discuss six tests. In 1986, Huoman [16] developed the Programming Aptitude Test (PAT) written in Finnish. An English equivalent is also available [38]. The test contained five items on algorithmic reasoning, logic, and programming using natural language. The item types were free text answers, including two long and three short answers. Huoman studied n=69 participants, 24 of whom had no prior programming experience and 45 of whom had limited Pascal knowledge. The PAT was replicated in 2002 [38] with n=33 and in 2019 [36] with n=62. Huoman [16] did not analyze PAT results as a success factor. However, the first replication yielded a Pearson correlation of 0.513 between PAT and the exam grade, and the PAT predicted no more than 25% of the exam grade [38]. Smith et al. [36] examined the variance explained by the PAT scores. Examining only the PAT scores as factors, their data yielded a value of 20% for the midterm exam scores and a value of 16% for the final exam scores (data provided by the authors).

Also in 1986, Mayer et al. [25] presented a cognitive test. This test dealt with problem translation, procedural comprehension, general ability, and arithmetic computation. It was divided into eight different subareas. Neither the number of items nor the scoring scheme were listed. As an example, Evans and Simkin [9] used similar test items based on this test for examining predictive success factors. Mayer et al. [25] studied n=57 participants in a CS1 course that used Basic as the programming language. The reported correlation coefficients between the eight sub-areas and exam score ranged between 0.16 (verbal ability) and 0.56 (word problem solution) [25].

In 2006, Lorenzen and Chang [23] used essays from CS1 students based on beginnings of the logic game Mastermind© to measure programming aptitude. No test details were described, only that it was a CS1 course.

Ringenberg et al. articulated the same goal as our new test, namely "to measure the core knowledge required to excel in computer programming without the student ever having been exposed to programming" [32, p. 3]. This was addressed through the topics of mathematics, algorithmic thinking, and logic. Their second test version contained eleven multiple-choice items and three free-text items for entering a number. They studied about 350 CS1 students for the first version and about 450 students for the second version. A subset of twelve multiple-choice items was used in two MATLAB programming courses for engineering students [28, 31]. Ringenberg et al. [32] reported several correlations, namely between test scores and four exams as well as the overall course grade. For the latter, the correlations were 0.33 and 0.39 for the two test versions. The specific coefficient and the statistical significance level were not reported.

Leal [19] took a different approach in their aptitude test. They focused on time, state, and causality as abstract concepts that they considered essential to programming. Their test measured novice programmers' ability to predict the behavior of balls in a physical simulation using these concepts. The test contained 30 dichotomous items of increasing difficulty. Leal studied n=57 participants in a CS1 course that used C as the programming language. Leal reported a correlation of 0.31 between test score and mid-term grade. The information about the specific coefficient and statistical significance level has been lost (data provided by the author).

Most recently, Harris [11] invented an assembler-like programming aptitude language (PAL) in 2014. PAL followed the procedural paradigm and included memory cells and operations such as store, add, read, and print. Students first completed a tutorial (PATT) to learn PAL. Then they completed eight code-tracing and code-writing tasks using PAL. Harris studied PAL with n=23 students in a CS1 course using C#. The correlation between the score of PATT and the midterm score was 0.881 for all questions and 0.974 for the average score of questions 2 to 8. The specific coefficient and statistical significance level were not reported.

In total, three of the six tests [11, 19, 23] are not used in any other publication. We consider the other tests. Huoman's test [16] was replicated, but Smith et al. [36] used a different scoring scheme because it was not digitized. Only similar test items from Mayer et al. [25] were used [9]. Replications of the test by Ringenberg et al. [32] used only some of the items and could not replicate the earlier results [28, 31]. Their test has a high proportion (81%) of multiple-choice items. Guessing the correct solution can lead to a bias and weaken the accuracy of the measurement. One advantage is that their test is the only one that achieves a sample size of 100 or more. The sample sizes of the other studies tend to be small, less than 70. Three of these tests, namely [11, 16, 23], would have benefited from a discussion of inter-rater agreement. All three use open-ended questions such as writing text or code that cannot be evaluated in a fully automated fashion.

As a result of studying previous tests, the NLCT should have the following characteristics: It should (1) measure commonsense computing directly, (2) all items and scoring schemes can be requested, and (3) a portion of the items are scored fully automatically to increase inter-rater agreement and reduce personnel effort. We would also like to test the NLCT with a large sample.

2.2 Success Factors in CS1

Success factors are of interest since the beginning of computing education research. Primarily, we refer to a systematic literature review on predictive factors that includes 357 reviewed papers from 2010 to mid-2018 [12]. The large number of papers published since mid-2018 demonstrates continued interest. One example of many is the ongoing development of the PreSS model [30], which enables automatic and early prediction based on student characteristics. The characteristics include, among others, programming self-efficacy, mathematical ability and age. In the review, the overarching categories cover a broad spectrum: demographic, personal, academic, behavioral, and institutional factors [12]. The definition of success also varies, and we list only the three most commonly predicted values in descending order: course grade or score, exam/post-test grade or score, and course grade range [12]. Hellas et al. [12] categorize the methods used in the papers as classification, clustering, mining for patterns, and statistical computing. However, this is only a brief description of this area.

Our focus on commonsense computing in natural language led to a closer look at language as a predictive factor. Hellas et al. [12] used language as a predictive factor category. This included 11 of the 357 papers. According to the authors, the exact assignment of papers to factor categories can no longer be traced. The results are mixed for native and non-native English speakers, with the latter having additional barriers [3]. For students from Indiana, USA, both English unit and verbal scores on the scholastic aptitude test correlated significantly with their grade in the introductory programming course [20]. Byrne and Lyons [6] examined the English and foreign language skills of Irish students. These did not correlate significantly with exam scores in their introductory programming course. For non-native English speakers, English proficiency had the greatest impact on explaining differences in Chinese middle school students' programming [29]. South African students' English scores did not correlate significantly with their performance in an introductory programming course [2]. Ameri et al. [1] included language in the form of scores in English and reading on American College Testing in their prediction framework without listing its specific effects.

In summary, the research base on language as a predictive factor is thin and has yielded mixed results. We would like to introduce a new aspect for future research that could be a possible reason for the mixed results. When considering the influence of language, we should distinguish between language in the prosaic sense, e.g., English literature essays, and language in STEM subjects with its ideal characteristics such as precise, structured, logical, and operational.

3 NATURAL LANGUAGE COMPUTING TEST (NLCT)

The NLCT aims to measure commonsense computing as defined in Sect. 1. In programming, basic knowledge includes reading, tracing and writing code. When transferring this knowledge to knowledge expressed in natural language, each of the preceding knowledge has its own equivalent. The natural language version of code is a complex instruction that is structured and precise. Thus, reading code corresponds to abstracting complex instructions, tracing corresponds to following, and writing corresponds to formulating. In conclusion, this knowledge, together with logical reasoning, actually represent commonsense computing and therefore forms the basis for the NLCT items.

Here the items require reasoning in *procedural* programming, expressed in natural language. The generalizability to declarative programming languages is therefore not necessarily given. However, most programming languages used in CS1 courses are procedural languages [34]. The NLCT therefore covers most CS1 courses.

Below, we briefly describe the development and pilot phases. The first version consisted of nine items (11/2021). It was tested using n=9 think aloud protocols with students who had no prior programming experience. In the process, we iteratively improved the wording of the items to achieve better understanding. The second version (04/2022) was tested along with the coding manual with an additional n=38 students with no prior programming experience. Three items were then removed because almost all participants correctly formulated the long open answer. The tasks were too easy, but required a lot of reading time for scoring. The following version (05/2022) with six items and the corresponding coding manual was tested by two colleagues from our CS department via expert review. They gave some small suggestions for improvement in the wording of the items. They confirmed the content validity, i.e., that the NLCT indeed measured commonsense computing. The final version (05/2022) was tested with n=24 experienced CS students. These students were expected to excel at the NLCT because they had learned the programming and logic skills required to solve the items - and they did. This also ensured the content validity. This version[1] can be summarized as follows:

- **Aim:** The NLCT aims to measure the extent to which novice programmers are proficient in commonsense computing that is expressed in natural language.

[1]The entire test with coding manual can be requested from the authors.

- **Operational definition of measured construct:** Programming and logic knowledge that students bring with them before they receive formal instruction. For expression in natural language, these are abstracting, following and formulating instructions and logical reasoning.
- **Target population:** NLCT is designed for novices in CS1 courses based on a procedural programming language before or during the first week of the course.
- **Instrument type:** Performance test with single-choice items with two possible answers, short-answer items, ordering items, and matching items.
- **Length and estimated completion time:** 6 items; 40 min.

3.1 Items

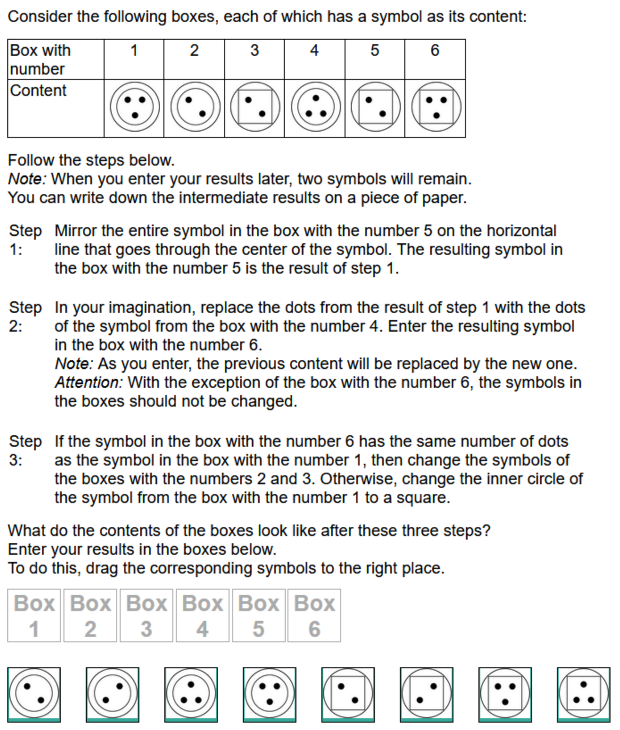
Next, items L1 to L6 are presented. Each of them addresses a genuine computing construct such as the iteration in items L3 to L6. All items use only natural language and no pseudocode or programming language. For space reasons, we only include screenshots for items L2, L5 and L6.

Item L1 deals with logical reasoning and Boolean expressions. We used the sandwich task introduced by Herman et al. [14] and replicated with a larger sample [39]. In this task, a good sandwich must follow three logical rules and students must decide and explain whether certain sandwiches are good or not. For example, in L1, one logical rule was: "The sandwich contains cheese if and only if it contains ham." An example of a sandwich to be evaluated was a sandwich that consisted of bread and ham. In L1, we used single-choice tasks to decide whether a sandwich was good or not (based on all three rules) and short-answer tasks to justify this decision. The short answers must be evaluated manually.

Item L2 uses the natural language equivalent of code tracing, namely following following instructions, see Fig. 1. In previous studies, participants had to follow step-by-step instructions to change the contents of a given number of boxes, e.g., [4, 9, 25]. These items are usually arithmetic calculations such as addition. Thus, measurement is confounded with arithmetic skills. It is also problematic if only one value is required to evaluate the item. We made two changes: (1) All box values must be given so that partially correct solutions can also be rewarded, as used by Evans and Simkin [9]. This change increases the variance of the item, so the item is likely to better discriminate between participants' abilities than before. (2) We replaced numbers with symbols to avoid confusion with arithmetic skills. The symbols include only circles, squares, and dots and therefore do not reveal specific knowledge. We used only clearly defined operations, such as swapping and replacing box contents. The values of the boxes are evaluated fully automatically.

Item L3 uses following instructions as the natural language version of reading code. The code to be followed describes how a robot moves on the edge squares of a 4x4 field and drops coins when its line of sight points up or down. For this purpose, it uses two nested loops. Both robot and field are similar to those in item L5. In total, participants must correctly place six coins on the 4x4 square by clicking on the correct squares. The evaluation is fully automated.

Item L4 implements code tracing and "Explain in plain English" [22], both common activities in CS1 courses. We have used a modified textual form of item F.5 [4], in which a counter is iteratively



Consider the following boxes, each of which has a symbol as its content:

Box with number	1	2	3	4	5	6
Content						

Follow the steps below.
Note: When you enter your results later, two symbols will remain.
You can write down the intermediate results on a piece of paper.

Step 1:	Mirror the entire symbol in the box with the number 5 on the horizontal line that goes through the center of the symbol. The resulting symbol in the box with the number 5 is the result of step 1.
Step 2:	In your imagination, replace the dots from the result of step 1 with the dots of the symbol from the box with the number 4. Enter the resulting symbol in the box with the number 6. *Note:* As you enter, the previous content will be replaced by the new one. *Attention:* With the exception of the box with the number 6, the symbols in the boxes should not be changed.
Step 3:	If the symbol in the box with the number 6 has the same number of dots as the symbol in the box with the number 1, then change the symbols of the boxes with the numbers 2 and 3. Otherwise, change the inner circle of the symbol from the box with the number 1 to a square.

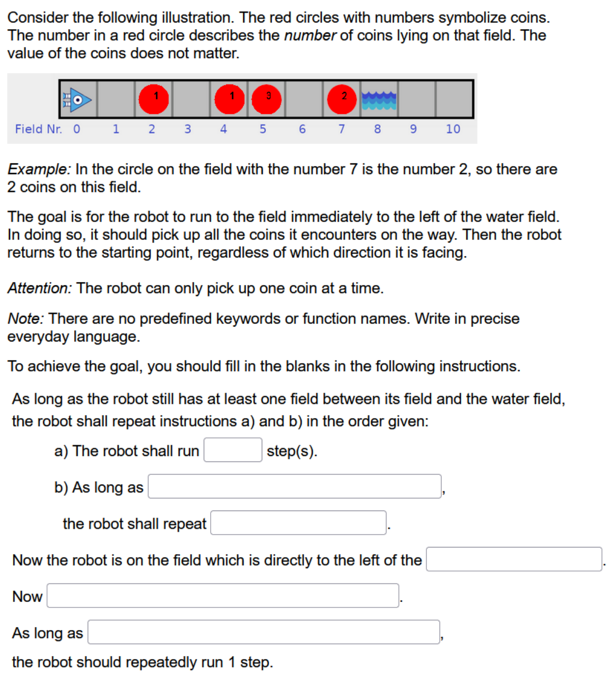
What do the contents of the boxes look like after these three steps?
Enter your results in the boxes below.
To do this, drag the corresponding symbols to the right place.

Figure 1: Item L2 focusing on following instructions.

incremented and added to a result variable. Unlike Bergin [4], we asked for all values were asked. Item L4 also required abstracting from the specific procedure to the purpose of the procedure, which is to add the integers from 1 to n. With these two changes, we aimed to more accurately measure participants' competencies. The answers for the abstraction must be evaluated manually.

Item L5 transmits the specific code completion task, called "Skeleton Code" [21] or "Fill in Blanks" [35], into a natural language version, see Fig. 2. Completion tasks have a good correlation to writing code and cover the entire difficulty spectrum [35]. Here, the code describes how a robot picks up coins on its way to a certain position and then moves to its starting position. To do this, the code uses two nested loops and another independent loop. A total of six gaps must be filled in and their short answers evaluated manually.

Item L6 corresponds to a Parsons puzzle that uses only natural language, see Fig. 3. Thus, the advantages of Parson's puzzles, namely the low effort for evaluation and the correlation with code-writing tasks [35], can be used. Here the correct code describes a loop over an array filled with symbols as in L2. Item L6 has seven correct steps and two distractors. The distractors should provide additional difficulty for better item discrimination. All steps were ordered randomly in the questionnaire. In natural language, where the words "as long as" are used, the scope of the statements is not necessarily clear. Therefore, we have integrated the scope of the statements with "beginning of the repetition block" and "end of the repetition block" in the statements themselves. The number of correct solutions is two, since the first two initialization steps are independent of each other. The evaluation can be fully automated.



Consider the following illustration. The red circles with numbers symbolize coins. The number in a red circle describes the *number* of coins lying on that field. The value of the coins does not matter.

Example: In the circle on the field with the number 7 is the number 2, so there are 2 coins on this field.

The goal is for the robot to run to the field immediately to the left of the water field. In doing so, it should pick up all the coins it encounters on the way. Then the robot returns to the starting point, regardless of which direction it is facing.

Attention: The robot can only pick up one coin at a time.

Note: There are no predefined keywords or function names. Write in precise everyday language.

To achieve the goal, you should fill in the blanks in the following instructions.

As long as the robot still has at least one field between its field and the water field, the robot shall repeat instructions a) and b) in the order given:

a) The robot shall run step(s).

b) As long as ,

the robot shall repeat .

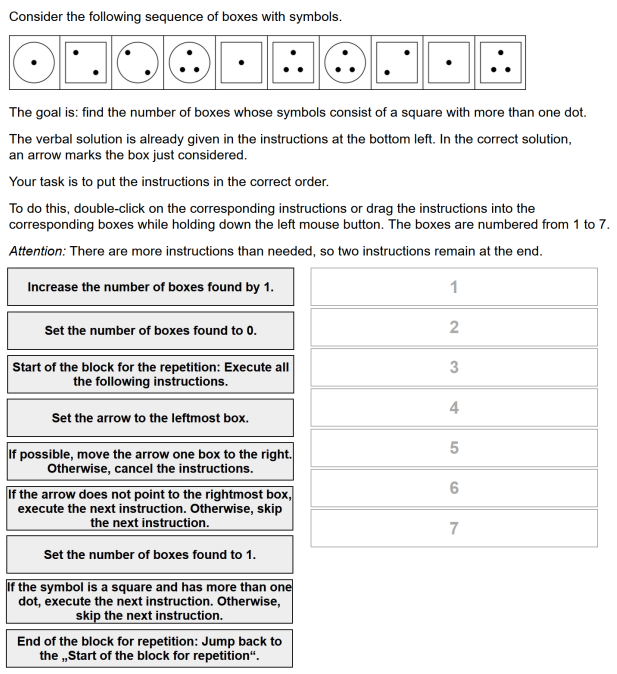
Now the robot is on the field which is directly to the left of the .

Now .

As long as ,

the robot should repeatedly run 1 step.

Figure 2: Item L5 focusing on formulating instructions.



Consider the following sequence of boxes with symbols.

The goal is: find the number of boxes whose symbols consist of a square with more than one dot.

The verbal solution is already given in the instructions at the bottom left. In the correct solution, an arrow marks the box just considered.

Your task is to put the instructions in the correct order.

To do this, double-click on the corresponding instructions or drag the instructions into the corresponding boxes while holding down the left mouse button. The boxes are numbered from 1 to 7.

Attention: There are more instructions than needed, so two instructions remain at the end.

Increase the number of boxes found by 1.	1
Set the number of boxes found to 0.	2
Start of the block for the repetition: Execute all the following instructions.	3
Set the arrow to the leftmost box.	4
If possible, move the arrow one box to the right. Otherwise, cancel the instructions.	5
If the arrow does not point to the rightmost box, execute the next instruction. Otherwise, skip the next instruction.	6
Set the number of boxes found to 1.	7
If the symbol is a square and has more than one dot, execute the next instruction. Otherwise, skip the next instruction.	
End of the block for repetition: Jump back to the „Start of the block for repetition“.	

Figure 3: Item L6 focusing on determining the order of instructions.

4 METHOD

4.1 Participants

The sample aimed to be representative of CS1 participants in general. To what extent we have achieved this, we discuss in Sect. 6.3. It included 700 students enrolled in the CS1 course at the authors' university in the winter 2022/23 semester. We translated the NLCT into German for them. At the time of the study, 1106 students submitted homework, representing the active population of the course. Thus, the response rate was 63%. We excluded 13 participants for not giving consent to use their data, another for submitting only random strings, and two more for omitting all responses. This resulted in 684 participants for RQ1 and RQ2. Only 681 students provided data on homework for student success and could be analyzed for RQ3.

The ethics committee required that demographic data be collected in a separate, optional questionnaire. This split ensured compliance with privacy regulations. However, more participants, 979, provided their demographic data. The split made it difficult to draw conclusions about our study participants. The ages of the 979 participants ranged from 13 to 78 (Md=20), with 50% between 19 and 22 years old. Explaining the age range, our course was open to gifted students still in school and retirees. Regarding gender, 22% participants identified as female, 75% as male, 0.4% as diverse, and 2% did not report. About 35% affirmed a migrant background, 58% did not, and 6% did not report. About 52% had CS as a school subject, 48% did not, and due to rounding, 0.3% did not report.

Learning to program in Java is the central theme of our 14-week CS1 course. The instructor was assisted by 30 student teaching assistants (TA). Topics include, among others, the basics of object-orientation, static and dynamic types, error handling, and generics. Students received points for 14 individual homework assignments, with half of the total points required for admission to the exam. There was an optional programming group project prior to the exam. The exam was written online with proctoring.

The participation period was 19 days. Start was the first day of the lecture. Last day was the submission date of the first homework assignment. Participation was optional. Participants received a small number of bonus points for completeness, not correctness. At the beginning of the study, participants were given a description of the subject, procedure, duration, and benefits. Participants indicated whether we could use their data. The study complies with the ACM Publications Policy on Research Involving Human Participants and Subjects. Participants completed the NLCT in 31 minutes (median), which was less than the estimated completion time of 40 minutes.

4.2 Statistical Analysis (RQ1 and RQ3)

In RQ1, inter-rater agreement (IRA) of the NLCT was addressed. Examining IRA for the entire dataset would have been too burdensome. Therefore, we focused on a subset. We analyzed data from the first 300 participants who completed the questionnaire. We excluded five participants from this subset because they did not provide consent to use their data. Of the remaining participants, we randomly selected 50. One author and 14 teaching assistants (TAs) independently rated the responses of these 50 participants using the coding manual. Thus, 15 raters were considered for IRA. We wanted to ensure that the coding manual stood on its own. To achieve this, the TAs received no training on the coding manual. We then calculated Krippendorff's α for each item and for the test as a whole. We calculated 95% confidence intervals for each α

value using bootstrapping with 10,000. Krippendorff [18] recommended relying only on variables with $\alpha \geq 0.8$, and variables with $0.667 \leq \alpha < 0.8$ should be used only for tentative conclusions.

For RQ3, we defined student success in CS1 as *percentage on homework assignments*. For the first examination, we drew a scatterplot with the NLCT on the x-axis and the predicted variable (homework) on the y-axis. As reminder, both variables are metric. If the variables are normally distributed, we can use the Bravais-Pearson correlation coefficient; otherwise, Spearman's rank order coefficient is appropriate. Here, the Shapiro-Wilk test was used to test for normal distributions. In either case, the selected coefficient value is reported with its 95% confidence interval. We also tested whether the correlation coefficient was significantly different from 0.

4.3 Item Response Theory Analysis (RQ2)

Item response theory (IRT) is a statistical modeling approach to estimating examinees' abilities based on their responses to test items [27]. It has a long history in the science of measurement in various disciplines. In our case, an examinee corresponds to a student participant and the test items are those of the NLCT. The need to use IRT analysis arises for the following reason: Some test instruments assign sub-points to test items and thus collect raw data in the form of ordinal data. Neither the distances between the sub-points nor the difficulty of the items are the same. Therefore, we cannot sum the raw numbers or calculate the mean of the raw data. IRT provides a solution because the mathematical models allow us to convert raw data into metric-scaled data. In this way, we can describe both the person's ability and the item's difficulty on a metric scale. IRT distinguishes between models based on the number of their parameters. Models with one parameter calculate only the difficulty of the items. Models with two parameters additionally compute item discrimination, i.e., a measure of the differential ability of an item. Ideally, high discrimination parameters are desirable to detect subtle differences in examinee ability.

The IRT analysis has two prerequisites. First, all test items measure the same underlying construct, which is referred to as unidimensionality. In the NLCT, the construct being measured is commonsense computing. We conducted a confirmatory factor analysis (CFA) to ensure unidimensionality. For CFA, the model fit statistics examined are the χ^2 test, root mean square error of approximation (RMSEA), and standardized root mean square error (SRMR) as an index of poor fit, and the comparative fit index (CFI) as an index of good fit. We used established cutoff values for interpretation [41]. We interpreted standardized factor loadings and aimed for them to be equal to or greater than 0.5 [10].

Second, IRT models require local independence of individual items. This means that the items of the test are statistically independent of each other when controlling for the same underlying construct. For the NLCT, the construct is the commonsense computing. To meet this requirement, the items should not build on each other. Yen's [43] Q3 statistic is commonly used to check for local independence. The Q3 values calculated for each item pair should be close to 0 to confirm local independence. However, there is no established rule of thumb, but a variety of arbitrary rules.

For our ordinal, unidimensional data, we considered the following three IRT models: the Partial Credit Model (PCM), the Generalized Partial Credit Model (GPCM), and the Graded Response Model (GRM). All models compute the various difficulty parameters of the test item. The PCM model assumes that the discrimination parameter is the same for all items and sets it to a value of 1. In contrast, the GPCM and the GRM allow different discrimination parameters for different items and compute them. The GRM differs from the PCM and the GPCM in that the thresholds of an item must be strictly ordered and it is a cumulative model. Thus, the GRM calculates the cumulative probability that a person's responses will be classified as category 1 or higher, for example.

First, we used the simplest model, PCM, to analyze item fit. We identified mismatched items and then adjusted the coding manual or collapse categories to remove these mismatches. Only when all items fit well did we select the more appropriate model for the data. We analyzed item fit in two ways: We examined the threshold order and the item characteristic curve (ICC) shapes. The threshold order must be strictly monotonically increasing. This is consistent with the fact that as ability increases, a person receives a higher score on the corresponding item. An ICC maps the person's ability to the probability of which category of the item is scored. The ICC must be shaped so that each scoring category has a separate "hill" [5]. A separate "hill" means that each scoring category has an interval of person ability in which that scoring category has the highest probability of being scored.

Second, we examined the model fit of the three models using two information criteria: The Akaike information criterion (AIC) and the Bayesian information criterion (BIC). The information criteria aim to identify the least complex model that still describes the data well. Both use penalty terms to avoid overfitting the data. In comparison, the penalty term of the BIC depends on the sample size, namely $ln(n)$, and is therefore larger for large samples.

Last, we combined item difficulty and student ability using a Wright Map [5]. We evaluated them as follows: (1) Item thresholds should cover the full range of difficulty. No item should not be so difficult to be far above a person's ability. Large ranges within persons' abilities without items are undesirable. (2) Ceiling effects are not desired. Here a participant answers all items correctly. Then no item has a higher difficulty level than the highest ability of this person. This person's ability score may be a little or a lot higher than the current score. The measurement error is then potentially unlimited. (3) Similarly, floor effects are not desired. Here, participants do not solve any item partially correctly. At least one item should be so easy to solve (partially) that all participants succeed.

5 RESULTS

5.1 Inter-Rater Agreement Evaluation (RQ1)

We calculated the corresponding Krippendorff's α values for each item and for the entire test, see Table 1. Only item L1 did not meet the requirement of $\alpha \geq 0.8$ but its value was close to it.

5.2 Item Response Theory Evaluation (RQ2)

5.2.1 Verifying Requirements. IRT analysis requires both unidimensionality and local independence. First, we examined unidimensionality using confirmatory factor analysis (CFA). The NLCT yielded a

Table 1: Inter-rater agreement (IRA) with n=15 raters using Krippendorff's α: An item with a dagger (†) is fully automated. An item in bold has an α-value of less than 0.8.

Item	α	95% Confidence Interval
L1	**0.770**	[0.754, 0.786]
L2†	1	[1, 1]
L3†	1	[1, 1]
L4	0.829	[0.818, 0.838]
L5	0.878	[0.870, 0.885]
L6†	1	[1, 1]
NLCT	*0.927*	*[0.924, 0.930]*

good model fit. At χ^2(9, N=684)=6.802, p=0.658, the p-value was not significant, consistent with our goal of good model fit. Both poor fit indices indicated good model fit, as they yielded low values: The RMSEA had a value of 0.000 with a confidence interval (CI) of [0.000, 0.035], meeting the requirement to be less than 0.06; the SRMR had a value of 0.022, meeting the requirement to be less than 0.08. The CFI also showed a good model fit with a value of 1.000, which must be greater than 0.95. The only negative point was that not all factor loadings met the requirement of more than 0.5. Item L3 had a factor loading of 0.336, whereas the other loadings ranged from 0.506 (L1) to 0.653 (L6). We examined the raw responses of item L3 in detail. Some of the participants had problems with the handling of the software when inserting or deleting coins. We therefore decided to exclude item L3 from the NLCT. This exclusion resulted in better fit indices: χ^2(5, N=684)=2.326, p=0.802, RMSEA=0.000 and CI [0.000, 0.034], SRMR=0.015. Only CFI did not improve, having previously reached its highest value.

Second, we examined the local independence using Yen's Q3 [43]. These values ranged from -0.311 (L2 and L4) to -0.129 (L4 and L6) with a median of -0.170. Overall, these values show that the NLCT met the requirement of local independence.

5.2.2 Analyzing item fit. We used the simplest model, PCM, to analyze item fit. First, we examined the thresholds of each item. For example, the first threshold is the ability of a person to be classified in the lowest category with the same probability as in the second lowest category. The thresholds should be strictly monotonically increasing, since a higher ability should be associated with a higher score. We have listed the thresholds in Table 2. There, three entries are in bold because they did not meet the requirements.

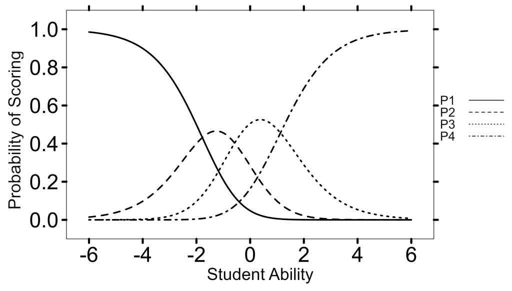
We then examined the item characteristic curves (ICC) of all items. All items with misfitting thresholds also had a misfit in their ICC. Here, some of their scoring categories were completely below the curves of the other scoring categories. We list one example for each a good and a poor fitting ICC, see Fig. 4.

In summary, the NLCT had two good fitting items (L1, L5) and three poor items (L2, L4, L6). The scoring of the NLCT (with the exception of L1) was structured to assign a score to errors made. Therefore, the categories could be recombined, see Sect. 5.2.3.

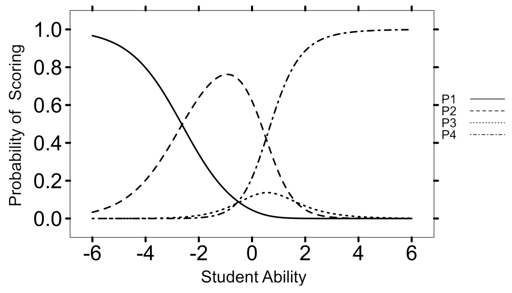
5.2.3 Adjusting coding manual. We made the following adjustments to the coding manual to improve the fit of the three items with poor fit. The original scoring included four categories, ranging

Table 2: Threshold values (b-values), where bold entries indicate problems as not strictly monotonically increasing.

Item	b1	b2	b3
L1	-2.423	-1.355	-0.591
L2	-1.313	-0.072	**-0.452**
L4	-2.631	1.655	**-0.599**
L5	-1.779	-0.577	1.129
L6	-2.164	-0.351	**-0.474**



(a)



(b)

Figure 4: Examples of item characteristic curves for items with good fit (a) L5 and with poor fit (b) L4.

from 0 as the lowest to 3 as the highest. For items L2 and L4, we combined categories 1 and 2 into one category. The adjusted score included three categories ranging from 0 as the lowest to 2 as the highest. For item L6, the thresholds indicated that it was a relatively easy item because they were all below 0. We then changed the mapping between error counts and scoring category. We made the item more difficult by replacing the original score (4+ errors → score 0, 2-3 → 1, 1 → 2, 0 → 3) with the adjusted score (3+ errors → score 0, 1-2 → 1, 0 → 2). As a result, the thresholds for each item were in the correct order. All shapes of the ICC also conformed to the requirements, i.e., each category had its own "hill".

5.2.4 Choosing adequate model. We describe which of the three models we have chosen. In general, for both AIC and BIC, the smaller the value, the better the fit of the model. The best model differed because AIC would suggest the GPCM and BIC would suggest PCM, see Table 3. However, the differences between the PCM and GPCM models for the AIC were very small. We then

Table 3: IRT models compared using information criteria.

Model	AIC	BIC
PCM	6625.4	6684.3
GPCM	6621.5	6698.4
GRM	6629.4	6706.4

examined the discrimination parameters. As a reminder, in GPCM these parameters are calculated and in PCM they are fixed at 1.

With the exception of item L1 with a value of 0.792, all parameters ranged from 1.156 (L4) to 1.339 (L6). All parameters were at most 0.339 units away from 1 and were on average 0.241 units away. The parameters therefore deviated more than slightly from 1. Thus, we chose the GPCM as IRT model. Moreover, the NLCT measures accurately in terms of distinctiveness as four of five items have values greater than 1. For discrimination parameters it holds: The larger the value, the better the particular item can discriminate between the person's abilities.

5.2.5 *Combining items and students.*

We then calculated and combined item difficulty and student ability. In IRT, this combination is usually represented in a Wright map, see Fig. 5. On the left, the map showed the distribution of student abilities, and on the right, the thresholds for each test item. For student ability, the NLCT had a mean of 0 with a range of -2.228 to 1.551. Only the ceiling effect occurred. This affected 29 participants (4%), which was a small fraction. There was no floor effect. Item thresholds occurred more frequently between difficulty values 0 and -2.5. There were no thresholds between L2-b2 (0.187) and L4-b2 (0.911). Thus, the items are somewhat skewed toward the easier items. Overall, the NLCT provides an accurate measurement in terms of IRT.

5.3 Test as Predictive Success Factor (RQ3)

We present the relationship between the test score in NLCT and student success defined as percentage on homework in Fig. 6. Both variables were not normally distributed according to significant Shapiro-Wilk tests with $p < 0.001$ in each case. Thus, Spearman's ρ was used as correlation coefficient. The value of ρ was was 0.365 with [0.296, 0.430] as 95% confidence interval. This was significantly different from 0 with $p < 0.001$. There were novices with low ability as measured by the NLCT but high scores on the homework assignments. There were also novices with high abilities but low scores on the homework assignments. These two groups were in contrast to the general relationship: higher ability on the NLCT was associated with higher student success. Especially these students lowered the value of the correlation coefficient. Overall, the suitability of the NLCT as a predictive factor was therefore limited.

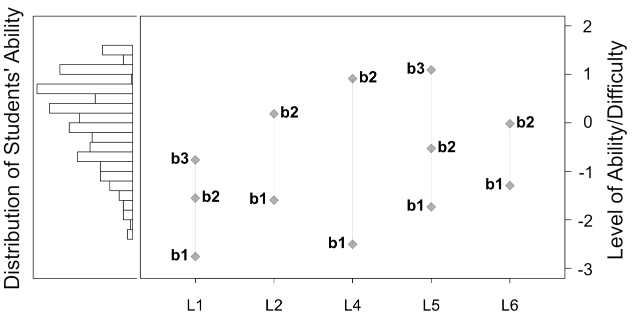


Figure 5: Wright map for the NLCT, showing the relationship between participants' test scores and item thresholds.

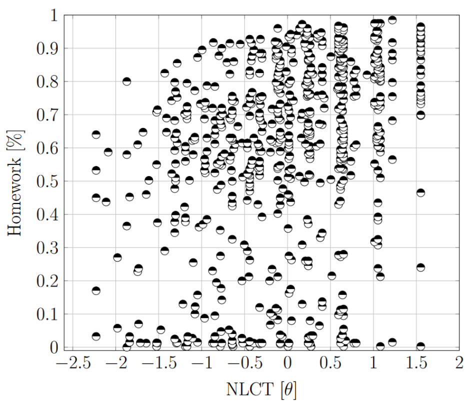


Figure 6: Scatterplot of the NLCT with percentage on homework assignments (Homework).

6 DISCUSSION

6.1 Answers towards the Research Questions

We asked how a commonsense computing test based on natural language (NLCT) could be successful as a programming aptitude test. The performance criteria were inter-rater agreement (RQ1), measurement accuracy by item response theory analysis (R2), and suitability as a predictor of student success in CS1 (RQ3). We extend the answers to RQ1 and RQ2 to the discussion of the NLCT as a standardized instrument. Standardized instruments should meet the requirements of reliability (i.e., consistency of measurement) and validity (i.e., measuring what is intended to be measured).

For RQ1, the results show that the NLCT has sufficient inter-rater agreement (IRA) as the NLCT coding manual stands alone for most items. The IRA results led to good reliability, as the scoring is (almost) independent of the raters.

For RQ2, adjustments to the coding manual were needed. After that, the items had no inconsistencies and their difficulty parameters covered the range of students' abilities. The NLCT would be improved if there were more items for higher ability students. Overall, we can conclude that the NLCT does indeed accurately measure its construct, commonsense computing, but still has room for improvement. As for reliability, our results on unidimensionality show internal consistency of the items. As for validity, the items measure commonsense computing through the direct application of computing tasks, such as code reading and tracing code. In addition, we included expert assessments to ensure validity.

For RQ3, the low Spearman's ρ value indicates a weak relationship between the NLCT and student success in terms of percentage on homework assignments. This is a weaker relationship than the reported values of explained variance in similar tests would suggest.

Previous studies using PAT as a test reported values of explained variance ranging from 25% [38] to 16% [36] for final exam scores and 20% for midterm exam scores [36].

Finally, we address how programming aptitude tests in general can serve as predictive factors. In Sect. 5.3, we described two groups of students who differ from the general relationship between test-taking ability and student success. We conclude: Even novices who show low ability on the test can improve and perform well during the semester. Novices who do well on the test need to study the content continuously and work on homework to perform well. They cannot rest on their initial performance. Thus, programming aptitude tests that rely on static performance on tests are generally limited as predictive factors. As a recommendation for instructors, we draw the following conclusion: use these tests to diagnose the initial knowledge of these novices, identify novices who need support, and offer support.

Overall, the NLCT performs well in terms of IRA and measurement accuracy based on IRT analysis. However, it performs poorly in terms of its ability to predict learning success in CS1.

6.2 Significance of Findings and Strengths of the Study

Programming aptitude tests are important in diagnosing the resources novices bring to the table. Appropriate assistance can be provided based on the diagnostic results. Such tests should be standardized instruments because they then benefit from validity, reliability, and comparability in other situations. We have developed a standardized instrument called NLCT to measure commonsense computing ability directly, without the use of programming languages. The term direct here refers to item types that address typical programming tasks such as reading, tracing, and completing code. In the field of programming aptitude testing, our study has several unique features: First, we used an item response theory approach to achieve accurate measurement. Moreover, this approach provided additional information on how well items can discriminate between students, which parts of the student ability domain are not covered by appropriate items with adequate difficulty, and how well the coding manual is designed with measurement in mind. Second, the much larger sample size (N=681) than previous studies in this area is one of the strengths of our study. Finally, we considered inter-rater agreement (IRA) with many raters (n=15) to verify that our tests met the requirements of standardized tests.

6.3 Limitations and Threats to Validity

As a limitation, the NCLT requires a lot of personnel, both to score the responses and to calculate the total score using item response theory. The first aspect can be mitigated by additional personnel, such as student teaching assistants, to help with scoring. The good inter-rater agreement results show that scoring can certainly be divided among different raters. The high requirement of personnel also lead to delays in providing feedback to students based on the test. In addition, the NLCT did not perform well in terms of its suitability as a predictive factor.

As an internal threat, student motivation is a confounding factor, especially in voluntary testing in general. Motivation affects both whether students are serious about taking the test and which students in the course are motivated to take the voluntary test. Thus, motivation is an internal threat that potentially affects our measured effects. Maturation could also affect student test score. Students who took the test later during the participation period might have benefited from the course content. We attempted to circumvent this factor by limiting the period to 19 days and aligning it with the submission of the first homework assignment.

As an external threat, the NLCT was used in only one course at one institution. In addition, the ethics committee required that demographic data be collected separately. Despite the high response rate of 63%, this is not a census of the course studied. Students with certain demographic characteristics may have systematically not participated in our study. In addition, we could not examine whether our items exhibit bias toward certain demographic characteristics. These include prior programming experience and general language proficiency. It is possible that only these students exhibit the high abilities measured by the NLCT. The items contain long descriptions in the language of our university. This language is not the first language for all of our students and for them there may be barriers. For these reasons, only preliminary results can be used for generalization to other universities, institutions, and participants.

6.4 Further Research Directions

We identify three strands of further directions: First, research could iteratively improve the NLCT by using the item response theory approach. Ideally, the improved test would contain only fully automated items, eliminating staff scoring and allowing direct feedback. Second, a parallel investigation of NLCT results and sources of potential bias could address the previously described threats of our study. This includes controlling for prior programming experience using a survey such as Smith et al. [36]. One could control for demographic bias by comparing the distribution of test scores between groups and conducting interviews with participants. One could also control for language proficiency bias by using general language tests. In general, replication and especially translation into other languages is desirable for possible generalizability. Third, research could develop a similar test for commonsense computing, but based on a different programming language paradigm than procedural. Similar in the sense of using item response theory to study the accuracy of measurements and items using only natural language.

7 CONCLUSION

Our study addresses the area of programming aptitude testing and the predictive factor for student success in introductory programming [12]. Methodologically, the study addresses the advancement and usage of standardized measurement instruments, which is currently underdeveloped in computing education research [24]. The test developed, the Natural Language Computing Test (NLCT), therefore combines all three perspectives. Unique features of our study include the use of item response theory to obtain more accurate measurement results and the much larger sample size, namely N=681. Our results show the advantages of the NLCT in terms of inter-rater agreement and measurement accuracy. However, the test is not a good predictor of student success. The NLCT could act as a starting point for iterative improvement of programming aptitude tests with fully automated evaluation.

REFERENCES

[1] Sattar Ameri, Mahtab J. Fard, Ratna B. Chinnam, and Chandan K. Reddy. 2016. Survival Analysis Based Framework for Early Prediction of Student Dropouts. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management (CIKM '16)*. ACM, New York, NY, USA, 903–912. https://doi.org/10.1145/2983323.2983351

[2] Glenda Barlow-Jones and Duan van der Westhuizen. 2017. Pre-entry Attributes Thought to Influence the Performance of Students in Computer Programming. In *ICT Education*, Janet Liebenberg and Stefan Gruner (Eds.). Springer International Publishing, Cham, 217–226.

[3] Brett A. Becker. 2019. Parlez-vous Java? Bonjour La Monde != Hello World: Barriers to Programming Language Acquisition for Non-Native English Speakers. In *30th Workshop of the Psychology of Programming Interest Group*. PPIG, Newcastle, UK, 40–52.

[4] Susan Bergin. 2006. *Statistical and machine learning models to predict programming performance*. Ph. D. Dissertation. National University of Ireland Maynooth.

[5] William J Boone, John R Staver, and Melissa S Yale. 2013. *Rasch analysis in the human sciences*. Springer, Dordrecht.

[6] Pat Byrne and Gerry Lyons. 2001. The Effect of Student Attributes on Success in Programming. In *Proceedings of the 6th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE '01)*. ACM, New York, NY, USA, 49–52. https://doi.org/10.1145/377435.377467

[7] Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, Kate Sanders, and Beth Simon. 2007. Commonsense Computing: Using Student Sorting Abilities to Improve Instruction. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '07)*. ACM, New York, NY, USA, 276–280. https://doi.org/10.1145/1227310.1227408

[8] Rodrigo Silva Duran, Jan-Mikael Rybicki, Arto Hellas, and Sanna Suoranta. 2019. Towards a Common Instrument for Measuring Prior Programming Knowledge. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE '19)*. ACM, New York, NY, USA, 443–449. https://doi.org/10.1145/3304221.3319755

[9] Gerald E. Evans and Mark G. Simkin. 1989. What Best Predicts Computer Proficiency? *Commun. ACM* 32, 11 (nov 1989), 1322–1327. https://doi.org/10.1145/68814.68817

[10] Joseph F. Jr. Hair, William C. Black, and Rolph E. Babin, Barry J. Anderson. 2019. *Multivariate data analysis* (8 ed.). Cengage, Boston.

[11] James Harris. 2014. Testing Programming Aptitude in Introductory Programming Courses. *J. Comput. Sci. Coll.* 30, 2 (dec 2014), 149–156.

[12] Arto Hellas, Petri Ihantola, Andrew Petersen, Vangel V. Ajanovski, Mirela Gutica, Timo Hynninen, Antti Knutas, Juho Leinonen, Chris Messom, and Soohyun Nam Liao. 2018. Predicting Academic Performance: A Systematic Literature Review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018 Companion)*. ACM, New York, NY, USA, 175–199. https://doi.org/10.1145/3293881.3295783

[13] Vivien Heller and Miriam Morek. 2015. Academic discourse as situated practice: An introduction. *Linguistics and Education* 31 (09 2015), 174–186.

[14] Geoffrey L. Herman, Lisa Kaczmarczyk, Michael C. Loui, and Craig Zilles. 2008. Proof by Incomplete Enumeration and Other Logical Misconceptions. In *Proceedings of the Fourth International Workshop on Computing Education Research (ICER '08)*. ACM, New York, NY, USA, 59–70. https://doi.org/10.1145/1404520.1404527

[15] Diane Horton and Michelle Craig. 2015. Drop, Fail, Pass, Continue: Persistence in CS1 and Beyond in Traditional and Inverted Delivery. In *Proceedings of the 46th ACM Technical Symposium on Computer Science Education (SIGCSE '15)*. ACM, New York, NY, USA, 235–240. https://doi.org/10.1145/2676723.2677273

[16] Juha Huoman. 1986. *Ohjelmointitaidon mittaaminen*. Master's thesis. Department of Computer Science, University of Joensuu.

[17] Päivi Kinnunen, Maija Marttila-Kontio, and Erkki Pesonen. 2013. Getting to Know Computer Science Freshmen. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research (Koli Calling '13)*. ACM, New York, NY, USA, 59–66. https://doi.org/10.1145/2526968.2526975

[18] Klaus Krippendorff. 2022. *The Reliability of Generating Data*. Chapman and Hall/CRC, Boca Raton, FL, USA. https://doi.org/10.1201/9781003112020

[19] José Paulo Leal. 2013. Testing the perception of time, state and causality to predict programming aptitude. In *2013 Federated Conference on Computer Science and Information Systems*. IEEE, Krakow, Poland, 721–726.

[20] R. R. Leeper and J. L. Silver. 1982. Predicting Success in a First Programming Course. In *Proceedings of the Thirteenth SIGCSE Technical Symposium on Computer Science Education (SIGCSE '82)*. ACM, New York, NY, USA, 147–150. https://doi.org/10.1145/800066.801357

[21] Raymond Lister, Elizabeth S. Adams, Sue Fitzgerald, William Fone, John Hamer, Morten Lindholm, Robert McCartney, Jan Erik Moström, Kate Sanders, Otto Seppälä, Beth Simon, and Lynda Thomas. 2004. A Multi-National Study of Reading and Tracing Skills in Novice Programmers. In *Working Group Reports from ITiCSE on Innovation and Technology in Computer Science Education (ITiCSE-WGR '04)*. ACM, New York, NY, USA, 119–150. https://doi.org/10.1145/1044550.1041673

[22] Raymond Lister, Beth Simon, Errol Thompson, Jacqueline L. Whalley, and Christine Prasad. 2006. Not Seeing the Forest for the Trees: Novice Programmers and the SOLO Taxonomy. In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education (ITICSE '06)*. ACM, New York, NY, USA, 118–122. https://doi.org/10.1145/1140124.1140157

[23] Torben Lorenzen and Hang-Ling Chang. 2006. MasterMind©: A Predictor of Computer Programming Aptitude. *SIGCSE Bull.* 38, 2 (jun 2006), 69–71. https://doi.org/10.1145/1138403.1138436

[24] Lauren Margulieux, Tuba Ayer Ketenci, and Adrienne Decker. 2019. Review of measurements used in computing education research and suggestions for increasing standardization. *Computer Science Education* 29, 1 (2019), 49–78. https://doi.org/10.1080/08993408.2018.1562145

[25] Richard E. Mayer, Jennifer L. Dyck, and William Vilberg. 1986. Learning to Program and Learning to Think: What's the Connection? *Commun. ACM* 29, 7 (jul 1986), 605–610. https://doi.org/10.1145/6138.6142

[26] Merriam-Webster.com Dictionary. 2023. "aptitude test". https://www.merriam-webster.com/dictionary/aptitude_test

[27] Insu Paek and Ki Cole. 2020. *Using R for item response theory model applications*. Taylor & Francis, Abingdon, Oxon.

[28] Branimir Pejcinovic, Melinda Holtzman, Phillip K Wong, and Gerald Recktenwald. 2017. Assessing student preparedness for introductory engineering and programming courses. In *2017 IEEE Frontiers in Education Conference (FIE)*. IEEE, Indianapolis, IN, USA, 1–5. https://doi.org/10.1109/FIE.2017.8190539

[29] Yizhou Qian and James D Lehman. 2016. Correlates of success in introductory programming: A study with middle school students. *Journal of Education and Learning* 5, 2 (2016), 73–83.

[30] Keith Quille, Soohyun Nam Liao, Eileen Costelloe, Keith Nolan, Aidan Mooney, and Kartik Shah. 2022. PreSS: Predicting Student Success Early in CS1. A Pilot International Replication and Generalization Study. In *Proceedings of the 27th ACM Conference on on Innovation and Technology in Computer Science Education Vol. 1 (ITiCSE '22)*. ACM, New York, NY, USA, 54–60. https://doi.org/10.1145/3502718.3524755

[31] Shanon Marie Reckinger. 2016. Implementation and evaluation of different types of peer learning instruction in a MATLAB programming course. In *2016 ASEE Annual Conference & Exposition*. American Society for Engineering Education, New Orleans, LA, USA. https://doi.org/10.18260/p.25561

[32] Jeff Ringenberg, Marcial Lapp, Apoorva Bansal, and Parth Shah. 2011. The Programming Performance Prophecies: Predicting Student Achievement in a First-Year Introductory Programming Course. In *2011 ASEE Annual Conference & Exposition*. ASEE Conferences. https://doi.org/10.18260/1-2--18930

[33] T. C. Rowan. 1957. Psychological Tests and Selection of Computer Programmers. *J. ACM* 4, 3 (jul 1957), 348–353. https://doi.org/10.1145/320881.320891

[34] Robert M Siegfried, Katherine G Herbert-Berger, Kees Leune, and Jason P Siegfried. 2021. Trends Of Commonly Used Programming Languages in CS1 And CS2 Learning. In *2021 16th International Conference on Computer Science & Education (ICCSE)*. IEEE, Lancaster, UK, 407–412. https://doi.org/10.1109/ICCSE51940.2021.9569444

[35] Guttorm Sindre. 2020. Code Writing vs Code Completion Puzzles: Analyzing Questions in an E-exam. In *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, Uppsala, Sweden, 1–9. https://doi.org/10.1109/FIE44824.2020.9273919

[36] David H. Smith, Qiang Hao, Filip Jagodzinski, Yan Liu, and Vishal Gupta. 2019. Quantifying the Effects of Prior Knowledge in Entry-Level Programming Courses. In *Proceedings of the ACM Conference on Global Computing Education (CompEd '19)*. ACM, New York, NY, USA, 30–36. https://doi.org/10.1145/3300115.3309503

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

[38] Markku Tukiainen and Eero Mönkkönen. 2002. Programming Aptitude Testing as a Prediction of Learning to Program. In *14th Workshop of the Psychology of Programming Interest Group*. PPIG, London, UK, 47–57.

[39] Tammy VanDeGrift, Dennis Bouvier, Tzu-Yi Chen, Gary Lewandowski, Robert McCartney, and Beth Simon. 2010. Commonsense Computing (Episode 6): Logic is Harder than Pie. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research (Koli Calling '10)*. ACM, New York, NY, USA, 76–85. https://doi.org/10.1145/1930464.1930479

[40] James V. Wertsch. 1990. Dialogue and dialogism in a socio-cultural approach to mind. In *The dynamics of dialogue*, Klaus Marková, Ivana; Foppa (Ed.). Harvester Wheatsheaf, New York, London, 62–82.

[41] Stephen G. West, Wei Wu, Daniel McNeish, and Andrea Savord. 2023. Model Fit in structural Equation Modeling. In *Handbook of structural equation modeling* (second ed.), Rick H. Hoyle (Ed.). Guilford Publications, New York, 184–205.

[42] Chris Wilcox and Albert Lionelle. 2018. Quantifying the Benefits of Prior Programming Experience in an Introductory Computer Science Course. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18)*. ACM, New York, NY, USA, 80–85. https://doi.org/10.1145/3159450.3159480

[43] Wendy M Yen. 1984. Effects of local item dependence on the fit and equating performance of the three-parameter logistic model. *Applied Psychological Measurement* 8, 2 (1984), 125–145.