Variables and variable naming in introductory programming education

Werf, V. van der

Chapter 1

Introduction

One of my students said to me: *"Miss, if naming already is an issue, we have a big problem!"* This quote comes from a whole-class discussion with young software developers-to-be, where the students debated about whether the topic of naming is an issue for software developers. For me, this statement perfectly reveals two important topics that illustrate the foundation of my research: (1) the relevance of naming practices to software developers and programmers, and (2) whether or not naming practices are hard to learn and apply for students and professionals.

However, before diving in deep, I will first highlight the context of this study. As the interest in programming skills is increasing in society, programming will inevitably become part of national curricula, and in some countries it already is. Through this dissertation, I aim to make programming more accessible to everyone, and contribute to inclusion of students and teachers from diverse backgrounds in programming education. Indeed I hope to inspire programming teachers to expand their focus beyond mathematics and problem solving. In particular, I approach the learning and teaching of a programming language from a natural language perspective. In this approach, I see natural language, that is already familiar to students, serving as a bridge between complex programming problems and the programming language itself.

Moreover, neuroscientific results indicate that language-related brain areas are more important for programming than brain networks related to problem-solving or mathematics [Prat et al., 2020, Endres et al., 2021b, Floyd et al., 2017]. Furthermore, classroom experiments demonstrate how training on technical reading abilities can improve programming skills [Endres et al., 2021a] and how reading code aloud, a strategy derived from natural language learning, helps with remembering syntax, which is an element of coding that beginners often struggle with [Hermans et al., 2018a, Swidan and Hermans, 2019]. Research on reading and explaining code has also shown that code reading skills correlate to code writing skills [Murphy et al., 2012a, Whalley et al., 2006].

From prior work, we can thus establish that learning a programming language is related to natural language and natural language skills. Keeping this in mind, my research focuses on one element of natural language found in written code, in particular, names, also known as identifiers or identifier names. Names are assigned to objects like variables and functions, among others –or the memory addresses associated with the objects– to represent a 'label' for the human reader, helping them remember what the variable refers to. For the non-programmer, these identifiers have a very similar function to labels or tags to represent the contents of, for example, a moving box.

Software engineering research has repeatedly confirmed the importance of identifier naming in reading and understanding code [Lawrie et al., 2006, Feitelson, 2023, Avidan and Feitelson, 2017, Hofmeister et al., 2017, Hofmeister et al., 2017, Schankin et al., 2018, Arnaoudova et al., 2016, Feitelson et al., 2022]. The main takeaway from these works is that programmers rely on names for their comprehension of code. In particular, research indicates that good names support a programmer's comprehension while bad names interfere with their understanding or can even be (unintentionally) misleading. This means that programmers may end up with an incorrect understanding of a program due

to naming issues, which makes them slower in writing code, understanding its function, or finding a bug. Especially general, non-specific names, such as 'length' [Feitelson, 2023] or 'result' [Schankin et al., 2018], appear problematic, as well as context-less letters and unclear abbreviations [Lawrie et al., 2007b, Lawrie et al., 2006, Hofmeister et al., 2017, Beniamini et al., 2017], or names that are too long or too similar to remember well [Binkley et al., 2009]. To make it worse, also full words can be unintentionally misleading, depending on context and the interpretation of the reader [Avidan and Feitelson, 2017, Arnaoudova et al., 2016, Feitelson, 2023, Feitelson et al., 2022]. It is safe to say that any name should be chosen carefully and cautiously.

In practice, programmers generally follow standard conventions with clear rules on, for example, when (not) to capitalize letters (like the *Elements of Java Style* and *Java Language Specification*) [Butler et al., 2015]. They furthermore use a limited vocabulary [Lawrie et al., 2006, Antoniol et al., 2002, Caprile and Tonella, 1999] and commonly use single letter names, like *i, e, s*, and *c*, which cover one-tenth to one-fifth of all names in C, Java and Perl projects [Beniamini et al., 2017, Gresta et al., 2021]. The most popular names are non-specific names such as 'value', 'result', and 'name'. Other names follow a *singular noun-phrase pattern* such as 'nextArea' or 'max_buffer_size', or are formulated as plural when they concern a certain collection (of lists, arrays, etc.) or data grouping [Newman et al., 2020]. In Scratch projects, a block-based programming language originally directed at children, names tend to be longer than in other languages with only one in twenty-five names covering just single letters like *i, x*, and *y* [Swidan et al., 2017].

While different names *can* be understood by the majority of developers [Feitelson et al., 2022], developers still choose to rename, often to narrow the meaning and support code comprehension [Peruma et al., 2018]. Moreover, one in four code reviews contains suggestions about naming [Allamanis et al., 2014], indicating that the original naming was often not clear enough or perhaps incorrect. Professional guidelines describe good naming as 'meaningful', 'clear', and 'concise', mention to use 'familiar names' within the domain, and use words that are 'present in a dictionary' (i.e., [Vermeulen et al., 2000]). This shows that some of the research on (good) naming practices has been incorporated. However, what these statements and suggestions mean in more detail – in other words, how names are to be chosen or how 'meaningful' naming is to be applied in which context – largely remains up to the developer's interpretation of the guidelines, the context, length, and purpose of the code, and the developer's creativity or professional requirements. Hence, choosing appropriate names is much less straightforward than many guidelines make it seem, or many programmers may think, even for professional developers.

As the research shows, developers are still choosing non-specific names that hinder code comprehension and remain affected by naming choices. This begs the question of how novices and learners are affected by naming practices that they encounter, especially if they also have not learned the meaning of certain single-letter names, which might be common and obvious to professionals. Although research has yet to investigate how exactly such naming practices influence learners *or* learning, I will go ahead and assume that all naming practices affect novice programmers more than experienced professionals. I am confident in making this assumption for two reasons. First, novices are easily overwhelmed by the many new aspects that learning a (new) programming language brings [Hermans, 2020], which pressures the cognitive load of students. As a consequence, they might be

even more dependent on natural language for the comprehension of programs or while learning unfamiliar programming constructs. Using good names could thus facilitate learning, while bad names may handicap them or even lead them astray. Second, novices may still hold certain misconceptions, such as wrongly believing that computers interpret or assign values based on the semantic meaning of variables' names, which leads them to incorrectly apply semantic assumptions to syntax [Kaczmarczyk et al., 2010].

Time to come back to the opening quote; *"Miss, if naming already is a problem, we have a big issue!"*. Logical reasoning shows that *if* naming practices are highly relevant to programmers (they are), *and* naming practices cause issues among students and professionals (they do too), *then* the topic is important within programming education and deserves appropriate attention from the community. Unfortunately, and in stark contrast to the extensive research on the effect of names on programming comprehension, very little research covers the topic of naming practices in programming education. Some efforts have been made to incorporate naming practices in rubrics for teaching code quality and assess students' assignments [Stegeman et al., 2014, Stegeman et al., 2016, Glassman et al., 2015]. Indeed, feedback on naming practices with good *and* bad examples is highly valued by students [Glassman et al., 2015], and feedback related to code quality is frequently asked for [Börstler et al., 2017]. This suggests that topics such as code quality and naming practices, might not get enough dedicated attention in educational settings, however, comprehensive investigations into this research area are lacking in the existing literature.

1.1 Research objectives

This dissertation aims to open a scholarly discussion on naming practices in programming education, examining how these practices are, can, or should be effectively used and integrated in teaching novices. It furthermore aims to provide practical advice for educators on how to incorporate naming practices in their courses to enhance understanding, improve code readability, and shape the development of future programming curricula. Hence, my work strives to influence practitioners in the fields of Computer Science and Software Engineering as well as those involved with teaching programming skills to novices and professionals.

Before diving into the topic of naming, my dissertation starts with a wider exploration of code comprehension through reading. In particular, the following research question is addressed first:

RQ1 What do novice programmers express in their answers when asked to explain given code segments in their own words?

Then, to contribute to a scholarly discussion and inform educators on the topic of naming practices in programming education, my research investigates how teachers perceive naming practices, how the topic is currently taught, and how the topic should be taught based on scientific evidence. Therefore, this dissertation also addresses the following research questions:

RQ2 How are variables and their naming practices introduced in beginner programming education and materials?

RQ3 What are teachers' beliefs and perceptions about naming practices and teaching them?

RQ4 How can we incorporate activities that focus on naming in beginner programming education?

1.2 Research design and sneak preview

To answer the research questions, my research implements a qualitative and exploratory approach and uses different types of data. **Table 1.1** shows an overview of the different studies and my research approach is further detailed below.

First, to investigate how novice programmers make sense of code reading exercises **(Chapter 2)**, I present students with short programs (also containing natural language) and ask them to explain what the code does in their mother tongue. Such tasks are also known as *explain-in-plain-english* (EiPE) tasks. To find patterns in these explanations **(RQ1)**, I perform an exploratory **artifact analysis**, addressing three aspects: the explanations' focus, which elements are (not) included, and whether any misconceptions are demonstrated. Among the findings was that students rely on the available natural language that is presented in print and input statements, and names of variables and functions. Particularly relevant to this thesis, I found that students are influenced (or distracted) by such natural language in their interpretation of a program's purpose. These findings highlight the importance of natural language within a code and piqued my interest in variable naming practices.

Then, to explore the current landscape of teaching approaches and learning activities that focus on variable naming practices **(RQ2; Chapter 3, Chapter 4, Chapter 5)**, I start by **interviewing** teachers about their perceptions of variable naming in general **(RQ3)**, their beliefs about good naming practices **(RQ3)**, and their approaches to teaching the topic **(RQ2) (Chapter 3)**. These interviews are analyzed through an **open-coding** process and reveal self-reported approaches. To confirm and extend these self-reported approaches, I also **observe** actual teaching practices and educational materials in popular online courses (*Massive Open Online Courses: MOOCs*) **(Chapter 4)** and programming textbooks for children and novices **(Chapter 5) (RQ2)**. All three studies consistently reveal a wide variation in how naming practices are taught, with a strong(er) focus on 'syntax rules' demanded by the programming language rather than on what meaningful naming is, or why it is relevant. Among teachers, there is a dominant belief that naming is not difficult and is learned 'naturally by example'. However, the examples that students are shown in course materials, and the explanations that are given to them (if any), are often uninformative and inconsistent. Moreover, feedback is rarely provided and students are not encouraged to pay good attention to naming practices as the emphasis lies on whether the code works. Whether the code is readable or adheres to code quality norms is regarded as secondary, as evidenced by teachers (implicitly or explicitly) and educational materials, which sometimes even deliberately state that naming is not important and you can choose any name you like.

These results reveal that the opportunities that students have to develop good naming practices are limited. Knowing that good naming practices are essential within the

Table 1.1: Overview of studies

	RQ	Focus			Data source(s)	Type of (qualitative) research *(output or theme)*
		Students	Teachers	Materials		
Ch. 2	1	x			code explanations	artifact analysis (*code comprehension, EiPE skills*)
Ch. 3	2,3		x		interviews	open-coding (*current perceptions, teaching approaches*)
Ch. 4	2			x	MOOCs	observation (*current educational content, teaching approaches*)
Ch. 5	2			x	textbooks	observation (*current educational content, teaching approaches*)
Ch. 6	4	x	x	x	workshop, interviews	design, implementation, analysis (*learning activities, guidelines*)

programming profession, my studies thus demonstrate an opening for a better implementation of naming practices in programming education. In particular, I see room for interactive activities that focus on discussing why naming is relevant and when is a name meaningful, rather than telling students that naming is important, referring them to guidelines, or focusing on a set of specific rules that might differ per programming language or context.

Hence, to inform educators on how they can tackle the topic of naming practices in their courses **(Chapter 6)**, I present the **design** of a set of learning activities **(RQ4)** following a *dialogic teaching approach*. These activities emphasize reflection and discussion and provide easy-to-implement opportunities for students to see and discuss the effect of different naming choices. For example, I present a ranking activity focused on a set of names which encourages reflection because it requires students to rely on their perceptions and opinions to evaluate what they consider appropriate or misleading. Discussing these rankings (students' opinions) in class provides an opportunity for students to experience that naming needs are not as straightforward as they seems at first sight. Through **implementing** and **testing** the designed activities **(RQ4)**, I determine several insights and recommendations regarding the adoption of naming practices in a curriculum, such as the importance of whole-class discussion and individual reflection, and the easy adaptation of activities to fit any course without much teacher investment. Moreover, the activities reveal potential issues and obstacles perceived by the students, such as that paying attention to naming is considered too time-consuming, inefficient, or even irrelevant, even when the importance of naming for a (second) reader is recognized by the students. These findings highlight the relevance of 'priming' students to adopt good naming practices before expecting them to 'figure it out themselves'.

1.3 Chapter outline

In the above section, I presented some of the chapters' highlights and how each study shaped my research journey. In this section, I outline the structure of the dissertation by providing an overview of each chapter.

Chapter 2 introduces the importance of code reading exercises in learning a programming language. In particular, EiPE tasks are discussed, as well as code comprehension in general. The research presented in this chapter provides insight into what novice students express in their explanations after reading a piece of code, and what these insights reveal about how the students comprehend code. I performed an **exploratory** analysis on four reading assignments extracted from a university-level beginner's course in Python programming and paid specific attention to (1) the core focus of student answers, (2) elements of the code that are often included or omitted, and (3) errors and misconceptions students may present. I found that students prioritize the output that is generated by print statements in a program, followed by control flow elements and function definitions. Some students omit (relevant) details on the code's purpose beyond the information conveyed through natural language, and their explanations are negatively affected when these names convey unhelpful or distracting information. This shows that students rely on natural language elements of the code when they are asked to explain a program, which shows that explaining a program does not necessarily mean that they have understood the code.

Chapter 3 introduces the importance of variable naming practices for code writing, comprehension, and debugging, while at the same time demonstrating that little is known about how variable naming is taught. The research presented in this chapter investigates naming beliefs, self-reported teaching practices, and observations regarding variable naming practices of teachers of introductory Python programming courses. I adopted an in-depth qualitative approach by **interviewing** ten teachers from secondary education and higher education and developed several themes to answer our research questions. Among various opinions and practices, I found that teachers agree on using meaningful names, but have conflicting beliefs about what is meaningful. Moreover, the described teaching practices do not always match teacher's views on meaningful names, and teachers rarely encourage students to use them. Instead, teachers express that naming practices should not be enforced and that students will develop them by example. Whereas some teachers report focusing solely on conventions, others deliberately dedicate time for students to engage with naming, create self-made guidelines, provide continuous feedback, or include naming exercises on exams. This chapter concludes that naming practices are not deliberately taught even though they influence program understanding and code quality, as there exist inconsistencies in teachers' self-reported naming practices.

Chapter 4 and **Chapter 5** focus on the concept of variables in general and on variable naming practices, and aim to understand how these are introduced in *Massive Open Online Courses* (MOOCs) **(Chapter 4)** and programming textbooks for children and adults **(Chapter 5)**. The research presented in these chapters investigates (1) which definitions and analogies are currently being used to explain the concept of variables, (2) which programming concepts are introduced alongside variables, and (3) if and how variable naming practices are introduced in the materials. To answer these questions, I gathered qualitative data related to variables and their naming by **observing** 17 MOOCs (Java, C, Python) and by analyzing 13 programming textbooks (Python, Scratch). Collected data include connections to other programming concepts, formal definitions and used analogies, and explanations and examples used to introduce variable naming practices. I found that analogies are often explained using the 'variables-as-a-box' analogy, although some books also introduce them as a 'place' or 'label'. The definition of a variable mostly focuses on storing information whereas other elements such as tracking or accessing information, computer memory, or flexible use and changing values remain underrepresented. I furthermore found differences between programming languages in the order in which variables and other concepts are introduced, but the most connected programming constructs are data types, program execution/control flow constructs, and operators/expressions. Finally, in both MOOCs and textbooks, I found inconsistent teaching of naming practices that focus on *syntax rules* which, when not adhered to, break the program. Most courses and books remain vague about –or display disagreeing notions on– 'what is a meaningful name', and present only a few examples of good and bad names. These observations match the teachers' self-reported approaches and perceptions, meaning that there is room for more deliberate attention to the meaning of variable names within the current landscape of teaching naming practices.

Chapter 6 addresses how to teach naming deliberately, without centralizing specific naming rules or styles, and instead focusing on discussing questions such as why is naming important and when is naming meaningful. The chapter presents a **dialogic teaching**

approach focused on teaching a critical reflection on naming practices through the **design** of five types of activities: (A) expressing perceptions and experiences, (B) creating names, (C) evaluating names through ranking, (D) comparing codes, and (E) locating a mistake. For this study, I developed, ran, and analyzed a one-hour workshop, which is presented here together with the experiences gained by teaching it to two courses. Ultimately, this chapter leads to recommendations for teachers and has a two-fold contribution: (1) a set of (adaptable) activities and exercises for supporting deliberate naming practices that assist teachers interested in adopting them into their curriculum; (2) insights regarding the student perspective on naming practices, derived from the activities, revealing potential issues and opportunities in teaching the topic.

Chapter 7 presents the general discussion of this thesis by highlighting several key findings, placing them in a wider context, and discussing relevant implications for both educators and academics within the field of Computer Science Education. The chapter finishes with a comprehensive summary of the dissertation's main conclusions and a list of my further recommendations for future research and educational practice.

1.4 Origin of chapters

All chapters of this thesis have been published as full papers in peer-reviewed conferences. Chapters Two, Three, Four, and Five are all empirical studies, and Chapter Six is published as an experience report. Besides formatting, no changes were made to the papers' original text or content.

Parts of this **introduction** are based on (1) a poster abstract and presentation at the conference of International Computing Education Research (**ICER'22**) in Lugano, Switzerland, titled *(How) Should Variables and Their Naming Be Taught in Novice Programming Education?*, by Van der Werf, Aivaloglou, Hermans, and Specht [van der Werf et al., 2022a]; and (2) a doctoral consortium poster and presentation at **KoliCalling'23** in Koli, Finland, titled *Fostering a natural language approach in programming education (Doctoral Consortium)*, by Van der Werf [van der Werf, 2024].

Chapter 2 was published as *What does this Python code do? An exploratory analysis of novice students' code explanations*, by Van der Werf, Aivaloglou, Hermans, and Specht, in the Proceedings of the 10th Computer Science Education Research Conference (**CSERC'21**), and presented online in 2021 [van der Werf et al., 2022b].

Chapter 3 was published as *Teachers' Beliefs and Practices on the Naming of Variables in Introductory Python Programming Courses*, by Van der Werf, Swidan, Hermans, Specht, and Aivaloglou, in the Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training (**ICSE-SEET'24**), and presented in Lisbon, Portugal in 2024 [van der Werf et al., 2024c].

Chapter 4 was published as *Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs*, by Van der Werf, Zhang, Aivaloglou, Hermans, and Specht, in the Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education (**ITICSE'23**), and presented in Turku, Finland in 2023 [van der Werf et al., 2023].

Chapter 5 was published as *Variables and Variable Naming in Popular Programming Textbooks for Children and Novices*, by Van der Werf, Hermans, Specht, and Aivaloglou, in the Proceedings of the 2024 ACM Virtual Global Computing Education Conference (**SIGCSE Virtual'24**), and presented at the online venue in 2024 [van der Werf et al., 2024b].

Chapter 6 was published as *Promoting Deliberate Naming Practices in Programming Education: A Set of Interactive Educational Activities*, by Van der Werf, Hermans, Specht, and Aivaloglou, in the Proceedings of the 2024 ACM Virtual Global Computing Education Conference (**SIGCSE Virtual'24**), and presented at the online venue in 2024 [van der Werf et al., 2024a].