Chapter 7

Discussion and Conclusions

I started this dissertation with the quote: *"Miss, if naming already is an issue, we have a big probem!"*. My work revealed that the way this topic is currently taught in computer science education is inconsistent, and arguably, insufficient for the desired outcome of (1) professional programmers, or (2) high-quality code. My research has shown that addressing the topic of naming practices in education is important, and therefore, it deserves appropriate attention from the community.

The current chapter aims to provide several openings to support scholarly discussion within the community on how the topic can or should be addressed in the future. It also provides an opportunity for educators to reflect on their current practices and aims to support them in how they can address the topic in their courses to (better) match their learning philosophy and objectives.

The chapter is structured as follows:

- A recap of the aim and the research questions addressed in this dissertation;
- A discussion of this dissertation's key findings, organized by research question;
- A concluding summary which includes highlighted recommendations.

7.1 Recap of research aim and questions

This dissertation addressed programming education with the assumption that natural language serves as a bridge between complex programming problems and the programming language itself. Specifically, I aim to open a scholarly discussion on how naming practices can or should be implemented in programming education. The contributions to these objectives were laid out through the following chapters and research questions:

RQ1 *What do novice programmers express in their answers when asked to explain given code segments in their own words?* **(Chapter 2)**

RQ2 *How are variables and their naming practices introduced in beginner programming education and materials?* **(Chapters 3, 4, 5)**

RQ3 *What are teachers' beliefs and perceptions about naming practices and teaching them?* **(Chapter 3)**

RQ4 *How can we incorporate activities that focus on naming in beginner programming education?* **(Chapter 6)**

7.2 Key findings in context

In this section, I answer the research questions by discussing my key findings in context. Key findings are presented in line with the topics of the individual research questions.

7.2.1 Natural language in code and code explanations

Chapter 2 explored students' code explanations in plain English to answer: [**RQ1**] **What do novice programmers express in their answers when asked to explain given code segments in their own words?** This chapter provided insight into what novice students express in their explanations after reading a piece of code, and what these insights tell us about how the students comprehend code.

I found that novice programmers rely on the natural language present in the code when they are asked to explain a given code snippet. This reliance helps them interpret a code correctly but can also distract or misguide them into incorrect beliefs about the code's function, code constructs, or individual lines of code. My results also hint that some of these mistakes are instruction-related, meaning that with a change in the educational material, natural language-related mistakes and misconceptions could be avoided. On the other hand, adding natural language also revealed students' fragile understanding of programming constructs, which can be used by educators to address misconceptions.

Key Finding 1 *Natural language affects novices' program comprehension and potential learning.* **(Ch. 2)**

With this key finding, I underpin the relevance of natural language in programming. Often programming is associated with mathematics and problem-solving skills and it is not to say that such skills are not important; they are undeniably relevant to programming. However, the influence of natural language on code comprehension fits within a wider research context focused on natural language skills and strategies within the domain of programming education. Indeed, research that introduces natural language acquisition strategies in programming education appears promising. For example, reading code aloud helps to remember syntax [Hermans et al., 2018a, Swidan and Hermans, 2019] and vocabulary acquisition techniques help secondary education students in learning programming [Veldthuis and Hermans, 2024]. Moreover, there are even indications that training technical reading skills also improves programming skills [Endres et al., 2021a].

My findings also fit within a body of literature that advocates more structured programming courses with an eye for skill progression. These works move away from teaching programming skills through immediate code writing to solve mathematical problems and instead move toward course materials that also (first) focus on reading and understanding code structures [Xie et al., 2019, Sheard et al., 2014, Lopez et al., 2008, Venables et al., 2009]. As was highlighted in such works, skills such as code reading, code comprehension, and code tracing are good predictors of code writing skills. Hence practicing code writing before learning to read or trace code may increase the difficulty of becoming proficient in programming. This highlights that code reading and comprehension skills, affected by the natural language present in the code, are prerequisite skills and deserve more attention in programming courses.

Additionally, according to Schulte's Block Model [Schulte, 2008], understanding a program means being able to build a bridge from the lowest types of information and entity size (text:atom) towards the higher categories of either dimension (goals:macro) (see Figure 2.1 in Chapter 2). In other words, the process of code comprehension entails *translating* the technical structures of a program to its social function. If indeed the natural language that is present in the code mediates the 'translation process', as my findings suggest, the academic community needs to further investigate how this finding can be used in educating novices with different backgrounds, including non-native English speakers.

My claim that natural language mediates comprehension can also be placed within the context that learning and motivation are affected when the used words or topic do not connect to the student's background. Indeed, research from the mathematics field is already familiar with the effect of language on so-called *story problems*. These are mathematical problems that require students to extract the relevant information from the story to solve a problem. Performance on such problems is affected by word choices and abstraction skills [Schley and Fujita, 2014, Mattarella-Micke and Beilock, 2010], meaning that elements such as names and objects (also known as *incidentals*) in the story can both help with comprehension and hinder it.

Some form of sensitivity to the words that are used, as I found in my study, was also found in computer science education research that opposed human-centered and thing-centered exercises as programming assignments [Christensen et al., 2021, Marcher et al., 2021]. These works found that problems focusing on humans rather than things are generally better understood and preferred by both women *and* men. Although my study did not investigate different student groups, it does further advocate for educators to be mindful regarding the topic and words they choose to use in example code. For example, I expect the effect of language to be bigger in unrepresented groups such as women, as objects can represent stereotypes of a group and deter those who do not identify with that group [Cheryan et al., 2009]. Hence, I invite researchers to dive into the potential differentiating effect of natural language in and around code on groups of minorities and varying levels of expertise.

On the other hand, completely obfuscating any natural language from a code limits the effect it has on comprehension. This could be especially useful to test how skilled a student is or has become in understanding 'pure' code structures. However, rather than testing programming skills, I am particularly questioning whether such obfuscation of any natural language from code is the most suited strategy for *teaching* novice learners, not in the least because within the profession, the use of natural language in code is often required to improve code quality and readability. Moreover, programming is hard to learn and students are already overwhelmed by the many new aspects involved with learning this new skill, so dismissing familiar elements likely increases difficulty for learning. Additionally, like with the mathematical story problems, abstraction skills –indisputably important for programmers– might be mediating the process. As such, training students to read code that includes natural language could facilitate the development of abstraction skills, while keeping unfamiliar elements to a minimum and the cognitive load limited. In this regard, future research could experiment with introducing (meaningful) natural language in code examples earlier or later on in the curriculum to investigate the effects on understanding new programming concepts and constructs as well as on abstraction skills. In this light, perhaps teaching students to structurally use comments, or introducing a more active practice of using sub-goal labels [Morrison et al., 2015] might be interesting too. The natural language that is already present in the code, through naming practices and input or output statements, may indeed mimic the effect of comments and sub-goal labels.

7.2.2 Variables and naming: current teaching practices

Chapter 3, Chapter 4, and Chapter 5 explore variable naming practices in particular and investigated teachers' perspectives, programming MOOCs, and programming books, to answer: [**RQ2**] **How are variables and their naming practices introduced in beginner programming education and materials?** These chapters painted a picture of the current landscape of how naming practices are taught, which served as the foundation for further investigation of how naming practices can and should be implemented. My observations revealed that educational materials and teachers (1) always give attention to the concept of variables relatively early in their course (materials) and (2) usually introduce naming practices along with the concept. Below I discuss my findings, for both topics separately.

Introducing the concept of variables

Generally, the concept of variables is introduced right after or just before data types and operators, although the order of introduction appears language-dependent. The concept is mostly described through the variable-as-a-box analogy, meaning that variables are typically explained as a box (or place) to store information in, often together with a visual representation. Python materials tend to include more diverse descriptions: they also introduce variables as a reference, name, or label. We observed that all materials focus their explanation on storing information, whereas other purposes such as keeping track of information, supporting code writing, interacting with information, or flexibility in using information, are rarely addressed. Only Scratch books mentioned that variables are called that way because their value can change. The common misconception that variables can store multiple values is rarely explicitly addressed.

Key Finding 2 *The introduction of the concept of variables is programming language-dependent. This is reflected by the chosen definitions and analogies and the position within the course compared to other programming constructs.* **(Ch. 4, 5)**

Key Finding 3 *The variable's purpose of storing information is extensively represented, whereas other functions or benefits of using variables are rarely mentioned, and potential misconceptions are rarely explicitly addressed.* **(Ch. 4, 5)**

Interpretations. These findings suggest that, while the concept of variables within the programming domain in itself is consistent across programming languages, the specific characteristics of these different languages may require a variety of approaches to teaching the concept, presumably because the use of variables may differ and their purpose might vary. However, while my findings give insights into current teaching practices, they cannot lead to definite conclusions about why they occur and therefore I stress that educational choices *seem* strongly influenced by the characteristics of the programming language that is taught. Whether this is the result of historical developments or the educator's preferences, it remains up to the community to decide whether it is desirable to adjust the teaching of general programming concepts to specific programming languages, especially considering the many different languages that are in use nowadays. Moreover, as of yet, it remains untested if any of the approaches result in a stronger understanding of the concept, compared to others, and if that result is indeed language-dependent.

Implications. Hence, our findings have implications for possible 'universal' learning trajectories, and the effect of a programming language on understanding programming concepts and constructs should be further investigated. Much in the same line, our findings also have implications for the transfer of knowledge across programming languages. Using a variation of descriptions and including a multitude of purposes related to variables could help the understanding of the concept. However, we need to be mindful of students' cognitive load, hence more research is needed to understand which descriptions are most helpful: just adding more or alternative descriptions to the curriculum is not appropriate. Additionally, our results cannot exclude the effect of learner age and prior knowledge or extended skills and vocabulary obtained before their first programming lessons. While most adults might not need an explicit explanation of the 'changing' characteristic of variables, younger students or students who lack knowledge of English vocabulary may not yet be familiar with the word 'variable' in general, thus needing more explicit explanations. Yet, such an explicit connection within an explanation could prevent misconceptions also among adult learners as many programming languages use an equal sign (=) to assign values to variables, which does not immediately suggest that such a value can change.

Limitations. Since relatively few courses and textbooks were analyzed compared to the total offer of programming courses, my observations might not be complete. Nevertheless, the systematic analysis was purposefully performed on a wide range of popular courses and programming textbooks. Because similar results were obtained from these different educational materials, I am confident that the findings are representative. Moreover, our tip-of-the-iceberg overview now gives way for additional research to look into patterns between student age groups, student backgrounds, or the taught programming languages.

Recommendations. Based on Key Findings 2 and 3, I recommend that educators experiment with an extended range of definitions that include purposes beyond just 'storing information' and pay attention to the misconceptions that may arise from their chosen analogies. As was already hinted in prior research [Hermans et al., 2018b], different analogies have different effects on (young) learners and it is still unclear which type of explanations are most useful for long-term learning and transfer to other languages.

Introducing naming practices

The investigated programming courses and educational materials provide inconclusive, inconsistent, or even conflicting information regarding naming practices, and also teachers indicate that they do not pay attention to the topic. Materials predominantly focus on specific *syntax rules* that prevent the code from breaking, and formatting styles such as when to capitalize letters or use underscores. Materials and teachers also refer to various community guidelines that are often specific to a programming language and not directly provided to the students. Teachers and materials rarely discuss the topic more in-depth or reflect on how to name appropriately, what it means to name "meaningfully", or why naming practices are important. There are indications that naming is taught through a *constructivist* pedagogical approach, in which students themselves discover by example what is good naming.

Key Finding 4 *Educators and teaching materials introduce naming practices inconsistently and they rarely address which, when, and why naming practices are (not) meaningful.* **(Ch. 3, 4, 5)**

This finding tells us that students are possibly insufficiently prompted to learn about the naming practices that are needed in their future careers. While a constructivist pedagogical teaching approach assumes that students learn from what they are confronted with through experiences and social interactions, any inconsistencies and discrepancies in (course) materials could prevent a coherent construction of knowledge. Indeed, the inconsistent teaching examples as observed in the course materials, programming textbooks, *and* as indicated by teachers, may (unwillingly) lead to the development of nonchalant attitudes toward naming practices. Instead of assisting student learning, students' current experiences with educational materials could thus hinder the adoption of a professional programmer's attitude. My finding therefore suggests that better attention needs to be paid to addressing and representing naming practices in programming education. It furthermore leaves an opening for adding a more explicit focus on *when* and *why* naming practices are important, which can be placed within the broader claim that programming education materials need to take a more structured approach. I propose that this is true whether or not a constructivist teaching philosophy is favored.

More implications and follow-up recommendations related to this key finding will be addressed in the next section, which addresses the educator's perspective. There I also demonstrate how the introduction of naming practices in course materials is embedded in the wider context of teachers' beliefs and assumptions, and what this means for students, teachers, and the academic community.

7.2.3 The educator's perspective: beliefs and strategies

Chapter 3 zoomed in on variable naming practices in particular and interviewed educators to answer: **[RQ3] What are teachers' beliefs and perceptions about naming practices and teaching them?** This chapter reveals several insights into how teachers think about naming practices and why they teach them the way they do. These insights serve as the foundation for further investigation of how naming practices can and should be implemented, explored later in the dissertation.

Teachers believe that names should be simple, straightforward, and intuitive, but there is no agreement on what this means in practice. As was already highlighted through **Key Finding 4**, this belief is not directly demonstrated in teaching approaches or materials. During the interviews, teachers mainly indicated to not explicitly incorporate the topic in their courses, nor encourage students to think critically about naming. They rarely grade naming practices or provide feedback to support students' self-reflection on their practices. Instead, they prioritize whether the student's code works and act from the assumption that naming is not difficult. The dominant philosophy is that naming practices are learned by example. At the same time, practical reasons prevent teachers from applying their conviction of good naming practices in the examples they use in their educational materials, including exercises, slides, or live coding sessions. While this inconsistency happens across educational levels, it is more prominent in university teaching than in secondary education, where a more direct (instructive) approach is applied: some teachers developed and adopted more active strategies to support their students, and reflection on naming practices is repeatedly woven back into the curriculum through continuous feedback, specific (naming) assignments, and dedicated attention discussing repeated mistakes.

Key Finding 5 *Educators express that naming practices are important and that names should be 'meaningful'. However, most indicate not to pay explicit attention to the topic and do not require good naming practices from their students.* **(Ch. 3)**

Key Finding 6 *Educators assume naming practices are not difficult and are learned by example. However, they also indicate using examples that are inconsistent with their belief, for example out of practical reasons.* **(Ch. 3)**

Key Finding 7 *Educators do not require –nor wish to enforce– specific naming styles and they rarely encourage good naming practices through feedback.* **(Ch. 3)**

While all teachers stress the importance of naming practices for programming and that students need to become proficient at naming, my findings show that these beliefs are not evidenced in their teaching approaches. The lack of a persistent teaching approach is in line with the observations presented in **Key Finding 4**. This means that it is unlikely for educators to pay (much) explicit attention to naming practices, even though they consider them relevant.

This inconsistency could be explained by that teachers overestimate how important they find naming practices, especially when it is compared to other programming topics. Indeed, some university teachers expressed that other programming topics are more difficult and deserve more time and attention, hence downplaying the relevance of naming practices in programming education. It is also possible that educators wished to express "socially acceptable" opinions. This is a common problem in research on attitudes and opinions, although this is usually more prominent in research on socially sensitive or political topics such as discrimination or the protection of the environment. While I cannot exclude that such an effect may have played a role, I can say that several teachers reported that they realized their inconsistencies through their reflections during the interview and expressed a desire to correct them. Some secondary education teachers also expressed that they initially underestimated the complexity of the topic and learned to address the topic more explicitly through experience.

Rather than overestimating the importance of naming practices in programming, I argue that most educators perhaps underestimate the complexity or relevance of the topic for (novice) students. The topic competes with other programming topics for the limited time in a curriculum and with teachers valuing these other topics as more important, naming practices end up drawing the short straw and are therefore (unintentionally) left out. My findings suggest that, even though naming practices are perceived as highly relevant within the community, most educators do not deem it necessary that explicit attention is given to the topic. However, my findings also suggest that this choice is usually made unconsciously, and more attention is given to the topic when teachers become more aware of the complexity of the topic for their students.

Nevertheless, the assumption that naming practices are not difficult and learned by example (**Key Finding 6**) supports the idea that the naming practices perhaps do not need explicit attention. However, to support learning when this assumption is true, the given examples should be consistent and match the teacher's philosophy on what is good naming. Instead, our findings suggest a mismatch between what students are expected to learn, and what educators are (unwillingly) teaching them. Rather than believing that educators do not care about the naming practices they teach their students, I argue that educators are generally not aware of the mismatch between their philosophy and practice.

Still, our finding that good naming practices are not required and rarely encouraged through feedback (**Key Finding 7**) also hints at the downplay of naming practices in programming education. Providing feedback on the topic might be considered unnecessary or too time-consuming, however, without any feedback, students lack the opportunity to check whether their naming can (or should) be improved. As a consequence, they may be led to believe that it is not important to use appropriate names in their code, let alone form a solid understanding of what good naming practices entail. Even when teachers tell their students that the topic is important, the lack of priority may suggest otherwise. Luckily, there are already initiatives that develop rubrics or tools to provide (large-scale) feedback on naming practices, and in a wider context, code quality. [Glassman et al., 2015, Börstler et al., 2017, Stegeman et al., 2014, Stegeman et al., 2016, van den Aker and Rahimi, 2024]. These can guide or inform teachers on how to incorporate feedback into their curriculum.

Implications for students. Unfortunately, it is yet unknown if the assumption that naming is learned by example is true. It is also unknown whether naming examples affect students of different ages differently. My research points to that students indeed copy examples they are shown in teaching materials and beyond, as teachers from secondary education especially highlight this. However, educators also warn that some of these students lack the understanding of why such names are chosen and copy them in inappropriate contexts. This begs the question of whether students should adopt a copy-paste strategy in the first place.

Teachers also relate 'bad names' with laziness or a lack of creativity on the student's part, rather than an inability to name appropriately. Based on my research findings (including **Key Finding 4**), it is possible that students never learned to name appropriately or do not care enough about it to pay attention to it themselves. This carelessness could reflect the inconsistent examples they were shown, which might lead students to believe that naming does not matter.

Implications for educators. Most importantly, educators and curriculum designers, including developers of educational materials such as MOOCs and books, need to be aware of their philosophy about (teaching) naming practices and check if their practices reflect what they wish students to take away from their teaching. When teaching time is limited and the topic of naming practices competes with other programming concepts, it is essential that the little information that is (indirectly) given conveys a consistent and deliberate message to aid the student's learning process.

Furthermore, to encourage students' understanding of why certain names are (not) appropriate in which context, it might be wise to incorporate reflection on given examples as part of the teaching materials. This not only teaches them about naming practices and their importance but also encourages students to think critically and use their creativity. The lack of reflection and creativity, mentioned by teachers as holding students back from adopting good naming practices, could reflect the struggle that students experience while writing code. This may leave students (too) overwhelmed with other aspects of their learning process, and as a consequence, it limits their reflection and creativity. In light of this, I suggest training these skills outside of code writing activities. Instead, teachers could incorporate reading and reflection activities before or after writing activities that allow students to compare and reflect on written (example) code.

Implications for the academic community. Our findings furthermore mean that our academic community needs to investigate how different examples influence the learning process, and where in this process it is best to introduce more reflection on naming. For example, we do not know if it is (more) beneficial for learning new programming concepts and constructs if meaningful names are used in explanations and examples, or if random names (foo) or letters (a, s, x) are used. We also do not know if it is useful for students to spend (more) time reflecting on naming examples to improve their program comprehension skills. However, we do know that names influence program comprehension and that students who show better programming skills generally use better naming practices and vice versa. Moreover, code quality is considered important, therefore, even *if* naming skills do not improve overall programming skills, they should be learned to become proficient as a developer.

Limitations. We interviewed only a limited number of educators and deliberately included teachers from different educational levels and countries. This means that individual teacher perceptions may not be entirely representative, and future research should follow up with a large-scale (international) questionnaire to generalize and compare target audiences, class sizes, and class duration. This is interesting because my work indicates that the topic of naming is –and perhaps should be– addressed differently across educational levels and we lack the empirical insights into what are appropriate ways to teach the topic. Since we only interviewed teachers involved in Python programming courses, and there are indications that different programming languages are taught differently regarding the topic of (variable) naming practices, such larger-scale quantitative research could also investigate and compare a larger set of programming languages, which can reveal relevant results regarding potential transfer with the acquisition of a second programming language.

7.2.4 Implementing activities focused on naming

Chapter 6 aimed to inform educators on how to address naming practices in their course, based on the findings that were obtained through Chapters Two, Three, and Four. This foundation was used to design a set of interactive learning activities, addressing the research question [**RQ4**] **How can we incorporate activities that focus on naming in beginner programming education?** This chapter presented the value of discussions and dialogue in teaching the topic and shed light on several student issues that prevent a solid professional programmer's attitude toward naming.

Teacher-led whole-classroom discussions, centralized around various naming examples in presented code are useful in introducing the whats, hows, and whys of naming practices. Moreover, they can reveal issues among students that hinder the adoption of desired naming practices, namely the concern that paying attention to the topic is too time-consuming, inefficient, and even irrelevant. If we are to educate skilled professionals, these issues need to be actively attended to, especially in a teaching context where teachers believe that students will figure out naming by themselves.

The in-activity dialogues allowed for an increased awareness through repeated reflection on how chosen names can be (unintentionally) misleading to other readers. These dialogues are supported by using a single code snippet for various activities with changed variable names and by using a variety of codes to create repeated practice with a wider range of examples. This flexibility makes the activities versatile for use in courses with varying programming languages, levels, and other varying contexts. Moreover, the examples that were discussed provided opportunities for students to develop a sense of what belongs to 'good naming practices' in different contexts, and they can serve as a form of feedback on the topic. Because the activities were designed and implemented in the context of vocational education classrooms with 16-year-old students, it might be challenging to implement them in a large-scale university setting. However, I see an opportunity for scaling with the use of online tools and flipped classroom approaches as comparable prior work on social annotation in introductory programming courses shows positive results [De Oliveira Neto and Dobslaw, 2024].

Since naming practices are highly context-dependent and influence code comprehension in various ways, I recommend that educators focus their teaching on encouraging students to develop a critical attitude towards naming practices through reading and reflection exercises. This way, students learn to develop a grounded perspective on the topic and recognize potential issues. This also prevents passive copy-pasting strategies and moves away from teaching specific (language-dependent) styles that students might need to unlearn later on in their careers. Future research should look into how interactive teaching approaches (supporting critical thinking, reflection, and naming choices through dialogue) influence the understanding of programming constructs and code writing. Research should also investigate if such interactive activities can best be introduced early on in courses or could have a place in later courses. This research could also further explore why certain (unspecific) names such as 'result' and 'outcome' are favored by students and how the use of different types of names affects other programming skills. This can further inform the direction of in-class discussions.

7.3 Concluding summary

This dissertation focused on natural language (elements) in programming education and skills. I found that natural language can serve as a bridge between complex programming problems and the programming language itself **(RQ1; Chapter 2)**. Yet, programming education rarely teaches students how to use and interpret any natural language that is present in a code: specifically, my research found that students are rarely taught how to name their variables and functions and how such names can interfere with their code comprehension **(RQ2; Chapter 3, 4, 5)**. Educators believe that using good naming practices is an important skill for professional programmers, but assume learning this skill is not difficult and is generally done naturally by example **(RQ3: Chapter 3)**. Students are therefore not required to use good names and receive little to no feedback on their naming practices. Nevertheless, code examples used in courses and textbooks do not always reflect what teachers describe as good naming practices **(RQ2; Chapter 3, 4, 5)**, hence, the expectation that students learn by example might be compromised. Interactive activities that include whole-class dialogue based on various naming examples can raise awareness for the topic's importance, allow students to experience the effect of (unintentionally) misleading names, and provide opportunities for feedback needed to develop one's understanding of good naming practices **(RQ4; Chapter 6)**. Moreover, such activities revealed issues among students that may prevent the adoption of good naming practices.

Based on these results I make the following recommendations:

- *More awareness of the complexity of naming practices and their effect on learning programming skills is needed among educators and computing education researchers.* While educators and professionals agree on the importance of naming practices for professional developers and high-quality code, the topic seems to be overlooked in teaching programming skills.
- *More work and reflection is needed on whether and how programming education needs to actively teach skills on naming practices.* We already know that these practices are important for code comprehension, code quality, overall programming skills, and professional expectations, but know little about how these practices are acquired and how they may affect the adoption of other programming skills.
- *Academics need to further investigate the effect of naming practices –and in a wider context also natural language, code quality, and readability– on the adoption of programming skills.* This should ideally also contribute to a structured learning trajectory with an appropriate focus on other aspects of programming beyond problem-solving and code-writing abilities.
- *Educators need to be(come) aware of their philosophy on how they assume their students learn and adopt new skills and appropriately align their teaching approach.* Currently, there seem to be worrying inconsistencies between what is intended to be taught and what is taught in practice. If naming practices can be taught by example, examples should be consistent throughout educational material.
- *I encourage the adoption of interactive activities to explicitly address student issues, naming difficulties, and professional expectations.* These activities can be easily

adapted to varying contexts, programming languages, and presumably also classroom sizes, hence requiring relatively little effort for teachers, while providing authentic and repeated moments of reflection for students. Moreover, the focus on discussion and reflection opens up space for teaching naming practices beyond specific (language-specific) guidelines.