Promoting Deliberate Naming Practices in Programming Education
A Set of Interactive Educational Activities

Van Der Werf, Vivian; Hermans, Felienne; Specht, Marcus; Aivaloglou, Efthimia

DOI
10.1145/3649165.3690115

Publication date
2024

Document Version
Final published version

Published in
SIGCSE Virtual 2024 - Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1

Promoting Deliberate Naming Practices in Programming Education: A Set of Interactive Educational Activities

Vivian van der Werf
v.van.der.werf@liacs.leidenuniv.nl
Leiden University
The Netherlands

Felienne Hermans
f.f.j.hermans@vu.nl
Vrije Universiteit Amsterdam
The Netherlands

Marcus Specht
M.M.Specht@tudelft.nl
Delft University of Technology
The Netherlands

Efthimia Aivaloglou
E.Aivaloglou@tudelft.nl
Delft University of Technology
The Netherlands

ABSTRACT

Despite extensive studies from the software engineering community on how naming practices influence programming behavior, the topic receives little attention in education. Prior work indicated little agreement on good naming because it depends on many factors. Students are told that "naming is important" and "should be meaningful," yet its practical implementation is rarely discussed and feedback is lacking. The current work presents a dialogic teaching approach focused on teaching a critical reflection on naming practices through five activity types: (A) perceptions and experiences, (B) create names, (C) evaluate through ranking, (D) compare codes, and (E) locate a mistake. We developed, ran, and analyzed a one-hour workshop, that we present here and share our experiences, leading to recommendations for teachers. Our contribution is twofold: (1) we provide a set of (adaptable) activities and exercises for supporting deliberate naming practices, thereby assisting teachers interested in adopting naming practices into their curriculum; (2) we provide insights regarding the student perspective on naming practices, derived from the activities, revealing potential issues and opportunities in teaching the topic.

CCS CONCEPTS

• **Social and professional topics → Computing education**.

KEYWORDS

programming education; naming practices; course design; dialogic teaching; reflection; critical thinking; student perceptions

ACM Reference Format:
Vivian van der Werf, Felienne Hermans, Marcus Specht, and Efthimia Aivaloglou. 2024. Promoting Deliberate Naming Practices in Programming Education: A Set of Interactive Educational Activities. In *Proceedings of the 2024 ACM Virtual Global Computing Education Conference V. 1 (SIGCSE Virtual 2024), December 5–8, 2024, Virtual Event, NC, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3649165.3690115

SIGCSE Virtual 2024, December 5–8, 2024, Virtual Event, NC, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0598-4/24/12
https://doi.org/10.1145/3649165.3690115

1 INTRODUCTION

From prior work, we know that variable naming practices are considered important by teachers, yet findings from introductory programming Massive Open Online Courses (MOOCs) and interviews with teachers [23, 24] indicate that many inconsistencies remain in teaching the subject matter. These inconsistencies are often due to variations in beliefs, goals, and intentions among teachers and designers of educational materials [23]. Therefore it seems critical that the Computer Science Education community provides guidelines or approaches to teachers on handling naming practices.

However, any teaching approaches also need to consider the perspectives and attitudes of students regarding the topic. Such student perspectives largely remain unexplored and teachers seem to handle the topic based on personal experience: prior work showed teachers indicating that their students do not find naming a problem because they never receive questions about it [23]. However, this could be a result of course design decisions or reflect factors such as the teacher's own beliefs or possible disinterest in the topic.

Since naming practices influence a programmer's code comprehension both positively and negatively [4, 8, 14, 18–20], we believe that software developers must have a thorough understanding of what makes a good name and be able to reflect critically on different naming practices. Our work presents a *dialogic teaching approach* to teaching a critical reflection on naming practices. We developed five types of activities that we ran and analyzed during a one-hour workshop given to a specialist vocational education program on software development. Through this workshop we were also able to explore students' perspectives and experiences on any barriers to adopting 'good naming practices', in this research denoted as names that carry the content or intent of the named object.

After presenting background on the topic of naming practices, we present our activities and their design (section 3), the workshop and its settings (section 4), and our experiences with the different activities (section 5). Finally, we reflect on our experiences and provide practical implications for the activity types.

2 BACKGROUND

That (variable) naming is important for comprehension and code quality is indisputable from the existing literature focusing on the effect of naming on program comprehension, code quality, and coding skills. Most importantly, programmers rely on names for their understanding of code [3, 14, 18, 19, 21, 22], and names often serve as beacons during code comprehension [10]. Moreover, bugs are easier to find when words are used [14]. Additionally, names that are not descriptive enough, for example, single letters or abbreviations from which meaning is not directly clear, interfere with code comprehension [4, 14, 18, 19]. The same holds true for too long names that can be difficult to remember [5]. Additionally, names can be unintentionally misleading and should therefore be chosen cautiously [2, 3, 8, 9]. Especially general, non-specific names, such as 'length' [8] or 'result' [20], appear problematic. Finally, novices can wrongly believe that computers interpret or assign values based on the semantic meaning of variables' names, and thus incorrectly apply semantic assumptions to syntax [15].

Consequently, thinking about teaching variable naming in introductory programming courses becomes relevant. Thirty years ago, Keller [16] indicated that variable naming was rarely included in programming textbooks. Since then, little research observed teaching practices on this topic. Recently, Van der Werf et al. [23, 24] found that teachers addressed naming practices in their learning materials, but inconsistently: variable naming practices are not always taught explicitly, taught practices are sometimes conflicting, and given examples codes do not always match the provided rules and recommendations. About a decade ago, Glassman et al. [11] developed a tool and a quiz for their online course (MOOC) to assess naming on length and vagueness. By evaluating the tool, they found that feedback on naming practices, as well as both good *and* bad examples, was highly valued by students. Unfortunately, no follow-up has been published since. Research investigating code quality perceptions among students and teachers [6] confirmed students' desire for 'more and more specific feedback about what was good and bad in their code'. Other studies on variable naming in education found that novice programmers often fail to name variables correctly [12] and that Scratch students are misled by variables named with a letter, probably because of prior knowledge from their mathematics education [13].

3 ACTIVITIES - DESIGN & EXPECTATIONS

Since good naming practices depend on several factors, such as the context, programming language, purpose, and naming conventions, we argue that practitioners should focus on fostering a critical but adaptive attitude towards naming. Rather than teaching specific naming styles, our activities are designed to (1) strengthen students' reasoning about 'good' and 'bad' naming practices by encouraging them to reflect on names and (2) support an understanding of how names can influence code comprehension. To further support these objectives, we also focus on (3) creating awareness through personal experience by letting students explore their perceptions on the topic and making them experience various advantages, drawbacks, and limitations of different names for themselves. This, in turn, highlights the effects of naming choices. Finally, we aim to (4) train deliberate naming choices by building critical thinking skills applied to naming. This is crucial for in-depth reflection, especially knowing that students are expected to figure out naming 'by themselves' while feedback on naming is often missing [23].

Critical thinking, defined as 'reasonable reflective thinking focused on deciding what to believe or do' [7], is often most effectively taught by combining *(critical) dialogue* with *authentic instruction* [1]. Dialogue in this context covers learning through discussion, specifically including teacher-posed questions and teacher-led whole-class discussion. Authentic instruction covers genuine and engaging problems such as applied problem-solving, case studies, simulations, games, and role-play. We maintain open-minded dialogue by applying the pedagogy of *dialogic teaching*, which is defined as 'a *general* pedagogical approach that embodies the strategic use of different types of talk, ranging from rote repetition to discussion, to achieve certain pedagogical goals' [17]. Our activities therefore also centralize whole-class discussions and authentic examples.

In particular, we developed five different activity types, each with an opportunity to reflect usually through whole-class discussion and comparison of answers supported by an online polling system: (A) develop and express **perceptions**, experiences, and opinions, (B) **create** appropriate names for given variables within a code, (C) **rank** a set of given names based on (perceived) appropriateness or deceptiveness, (D) read and **compare** two identical codes with different names, and (E) **locate** a naming mistake in a code containing one misleading name. To stimulate reflection and discussion on naming, we facilitated program comprehension by *always* accompanying our code examples with a description of what the code does, its output, and the contents of each variable, both through the presented materials and the teacher. Below we discuss the activities separately before showing how we adapted them to develop a one-hour interactive workshop on naming and presenting our experiences per activity.

3.1 Activity Type A: Perceptions

This activity type stimulates students' reasoning and opinions on naming, encouraging them to develop and express their perceptions and own experiences. We designed two variants, one to "warm-up" (A1), focusing on activating and motivating students to explore the topic based on their prior experiences, and one to "wrap-up" (A2), aiming to consolidate opinions and establish students' viewpoints.

Variant A1 includes questions such as "when writing code, do you pay attention to naming?", "do you find naming an issue for software developers?", and "in your opinion, is naming worth the effort?" that students answer on a scale from 1 (never/not at all) to 10 (always/absolutely) through the online polling tool. This tool generates a summary of opinions to show the class as input for discussion. The teacher facilitates the discussion by prompting for more in-depth reasoning, and students are expected to participate by reacting to one another. Variant A2 asks students to individually write down their reasons for paying or not paying attention to naming practices and what prevents them from paying (more) attention to it. This can be implemented right after discussion, or at the end of the lesson. Alternatively, variant A2 could be given as preparation before class in a *flipped classroom* style with the intended interaction during class, serving the same purpose as A1.

3.2 Activity Type B: Create Names

This activity uses student input on code snippets to lead the discussion, ensuring authentic instruction. The activity not only stimulates students to reason about appropriate names but also stimulates interest in the examples as the discussed names are their own. Additionally, the discussion allows for developing a common understanding of ‘good’ naming practices among the students.

Students are given a small code with redacted variable names. They individually name the redacted variables on paper and submit them anonymously through the online polling tool, which creates an overview of given names in the form of a list or word cloud. The teacher presents this overview, using it as the basis for discussion and prompting students to indicate what they notice about the set of names, which names they prefer and why, and what elements from these names they consider a part of “good” or “bad” naming.

3.3 Activity Type C: Evaluate Through Ranking

This activity encourages reflection by asking students to rank names for specific code snippets based on which they find most to least appropriate, or most to least misleading alternatively. By doing so, they rely on their perceptions and opinions to evaluate what they consider appropriate. Moreover, the activity provides an opportunity to experience that naming needs not be as straightforward as it seems at first sight. We expect different or opposing preferences to provide an ideal situation for discussion that is essential to reveal students’ reasoning, show them how and why a name can be misleading to some, and help them understand the effect of names.

We designed two variants: (C1) ranking names from a given set per a single variable from most to least appropriate, and (C2) ranking names from a complete code snippet (each name representing a different variable) from most to least misleading. Again, a whole-class, teacher-led discussion is facilitated by submitting individual rankings to the online polling tool.

3.4 Activity Type D: Compare Two Codes

This activity type provides an opportunity to reflect on the effect of different naming styles on code comprehension by reading and comparing two codes only differing in the names representing the variables. By prompting students to compare the two codes and evaluate which they find easier to understand or more efficient, students further develop their perceptions. Moreover, by prompting students to reflect on which code looks more like those of other people and those written by themselves, they are stimulated to put the naming styles, including their own, in context.

We opted for two identical programs representing opposite naming styles: (1) letters and abbreviations, and (2) full word names. Students write down what they notice while comparing the codes and then select the program most fitting to four questions (which is easier, more efficient, looks like their programs, looks like other people’s programs). They also explain their reasoning on paper.

3.5 Activity Type E: Locate the Mistake

This activity aims for students to understand the effect of names on their understanding of code, showing them that names can be (unintentionally) deceiving and that choosing a good name might not be as straightforward as they might assume. We expect that this activity might serve as an ‘eye-opener’ to students when they struggle to identify the naming error. Discussion afterward is essential to reveal students’ reasoning and to show them how and why a name can (sometimes) be misleading.

To mirror a real-life situation, we offer students code containing a (single) misleading name. To aid them, the explanation of the program, its output, and the contents of the variables are stressed (again). Students need to read and analyze the code to evaluate its names and are asked which name is wrong. Again, a whole-class, teacher-led discussion is facilitated by the online polling tool.

4 WORKSHOP - DESIGN, SETTING & DATA

To test our activities, we developed a one-hour workshop covering all activity types (see Figure 1), and implemented it in March 2024 in two first-year classes within a three-year vocational program Software Development in an urban area in The Netherlands. We reached a total of 27 (male) students, aged 16-17. Twenty-one students gave consent to use their data. To accommodate the course, the first author visited the classes to do classroom observations, gaining a feel for the classroom interactions. We then developed the assignments in collaboration with the students’ usual teacher, and presented the example codes in C#, as this is the language the students were learning, thereby eliminating possible confusion due to encountering an unfamiliar language. The first author led the workshop with the students’ usual teacher present. The workshop was given in Dutch, including all the variable names used. Quotes and names presented in this paper are all translated into English.

To collect data, we video recorded the front of the classroom, with only the whiteboard and the first author on tape, supported by additional audio recordings to capture students’ verbal input observational notes of any events taken by a student assistant. The video recordings were transcribed and complemented with transcription from the audiotapes when necessary. Furthermore, we collected students’ submissions in the online polling tool and their written contributions on paper hand-outs. These were digitized and added to the data from the polling tool using MS Excel. Finally, we collected students’ experiences of the workshop through a questionnaire, part of the paper hand-out. The Ethics Review Committee of Leiden University approved this research.

It should be noted that students appreciated the online polling tool and it worked as intended. However, paper writing proved more challenging as students remarked it had been “ages” since they had written anything with pen and paper. We also observed that the positioning of tables in the classroom influenced the workshop. There was a flexible seating arrangement with up to seven students per table. According to the teachers, this was the common setup, however, students were easily distracted by group dynamics within and between tables.

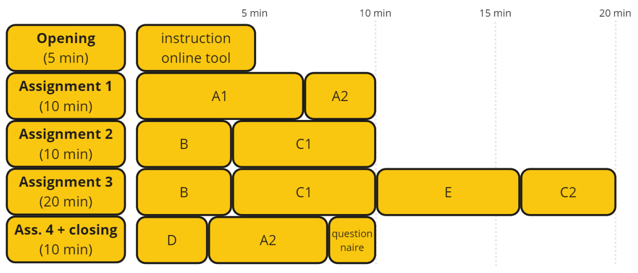

Figure 1: Overview of the workshop with time indications.

Table 1: Students' reasons for and against paying attention to naming and mentioned limitations

Reasons for	Code clarity (11)
	Understanding (7): self (4), others (5), both (2)
	Code readability (4)
	Eases programming/debugging (3)
	Preventing errors (2)
Reasons against	Costs (too much) time (9)
	It is a non-issue (5): i.e., *"I already know"*, … *"it is not necessary"*, *"it does not matter"*
	Competes with writing/performance/accuracy (3)
	Costs too much effort (2)
Obstacles	Time (8)
	It is a non-issue (6): i.e., *"nonsense"*, *"unnecessary"*
	Possible confusion (3)
	Nothing (3)
	Too repetitive (1)

5 WORKSHOP - EXPERIENCES & RESULTS

5.1 Assignment 1: Activity Type A1 & A2

The 'warming-up' discussions (**Activity Type A1**) were successful in exploring students' perspectives. Students expressed they did not find naming an issue for software developers since "it is not that difficult at all." When asked whether they pay attention to naming, students responded diversely across the scale, giving ample room for discussion. Some students indicated that they gave specific attention because "it is important to keep track of your code," whereas others said they did not because "it is easy."

The written assignment directly afterward (**Activity Type A2**) showed more depth to students' reasoning: when prompted to name reasons for paying or not paying attention to naming, we found several themes, addressed in **Table 1**. Many students acknowledge that naming improves the code and benefits both understanding and writing, yet at the same time, they also express several issues. Most importantly, students indicate that paying attention to naming costs too much time and students fail to see its relevance. These results reveal that students have mixed experiences and opinions about whether or not naming deserves their attention, some of which hold them back from embracing the topic.

5.2 Assignment 2: Activity Type B and C1

To illustrate different naming options and their effects, we used the same code snippet for both activities, which converts temperature from Celsius to Fahrenheit (Figure 2a). After each activity, a whole-class discussion facilitated the understanding of how, why, and when the use of certain names can be counterproductive.

When asked to name variables [A] and [B] (**Activity Type B**), students predominantly write *celsius-fahrenheit* (8) or *tempCelsius-tempFahrenheit* (7), where "temp" could also be replaced by "degrees". Less popular were constructions like *temperature-fahrenheit* (3), *temperature-result* (1) and *number1-number2* (2). This demonstrates a preference for clarity, and perhaps already internalized conventions or community guidelines, but also reveals lazy and less informative attempts.

```
double A,B;
Console.WriteLine("Give temperature in Celsius");
A = Console.Read();
B = 1.8 * A + 32;
Console.WriteLine("Temperature in Fahrenheit:" + B);
```

(a) Converts temperatures from Celsius [A] to Fahrenheit [B].

```
double A,B,D;
int C;
Console.WriteLine("give [A]");
A = double.Parse(Console.ReadLine());
Console.WriteLine("give [B]");
B = double.Parse(Console.ReadLine());
Console.WriteLine("give [C]");
C = double.Parse(Console.ReadLine());
D = (A * B * C) / 100.0;
Console.WriteLine("Euro " + D.ToString("N2"));
```

(b) Calculates profit [D] from a savings account with [A] amount of money, [B] interest rate in %, for [C] number of years.

Figure 2: C# Code snippets for assignments 2 (a) and 3 (b).

(n=18)	[A] celsius [B] fahrenheit	[A] degree [B] result	[A] temperature1 [B] temperature2	[A] temp1 [B] temp2	[A] input [B] output	[A] c [B] f	[A] a [B] b	[A] x [B] y
1st *[7 points]*	17	0	1	0	0	0	0	0
2nd	0	7	6	4	0	1	0	0
3rd	1	5	8	2	1	0	0	1
4th	0	4	2	7	2	3	0	0
5th	0	1	1	3	8	2	3	0
6th	0	1	0	0	4	3	7	3
7th	0	0	0	1	2	2	7	6
8th *[0 points]*	0	0	0	1	1	7	1	8
average rank	1,1	3,1	2,8	4,0	5,4	6,2	6,3	7,1
% of max points	98,4%	69,8%	74,6%	57,1%	37,3%	25,4%	23,8%	13,5%

Figure 3: Assignment 2, C1: Name-pair ranking

When ranking (best-worst) a set of name pairs for this code (**Activity Type C1**), we see a similar pattern (see Figure 3). However, students disagreed on the name pairs *temp1-temp2*, *input-output*, and *c-f*, which provided room for discussion in class: upon seeing the results of the ranking, students showed surprise, commenting, for example, that the name pair *c-f* was much better and more practical than *temperature1* or *temp1*, as these are too long and could be confusing. Although no student expressed it out loud, the disagreement for *temp1-temp2* could result from the common use of *temp* as an abbreviation for a temporary variable.

5.3 Assignment 3: Activity Type B, C1, E, and C2

Again, all activities use a single code snippet (see Figure 2b), allowing for experiencing and discussing the effect of different naming choices for a single code. The snippet presents a simplified calculation of the profit after saving a given amount for a given interest rate and a given time. Due to prior courses on the subject, students should be familiar with the economic context and terminology.

After explaining the program and each variable's contents, the students were asked to name the variables [A] to [D] (**Activity Type B**). Their answers revealed a preference for using a name that combines two words, such as *startingAmount*, *interestRate* or *numberOfYears* ("aantalJaar"). Also popular were single words such as *amount*, *balance*, or for variable [D], *result*, *outcome*, and *money*.

Interestingly, even though the variables' contents were discussed and provided, several students still made mistakes, writing *totalAmount* for variable [C] or *interest* for variable [D]. Moreover, while

Table 2: Students' 1st choice from a given list of names per variable (assignment 3: C1). The order shows the average ranking when a full ranking was provided (n=9).

[A] (n=18)	[B] (n=18)	[C] (n=21)	[D] (n=21)
startingAmount (17)	**interestRate (17)**	**numberOfYears (15)**	profit (5)
amount (0)	interest (1)	years (2)	result (5)
account (0)	percentage (0)	term (1)	outcome (2)
start (1)	perc (0)	time (0)	**endAmount (6)**
			calculation (0)

the names given to variables [A], [B], and [C] were mostly specific, the names given to variable [D] varied widely and showed little creativity. In fact, no student provided a name that included *profit*, which accurately describes the content. This could indicate a lack of domain knowledge. Alternatively, it could indicate an inability or unwillingness to translate the contents into a suitable name, or a lack of vocabulary or creativity. After seeing their classmates' answers, some students commented on the length of names by critiquing the combined names. This demonstrates a preference for shorter and more compact names.

For each variable, we presented students a set of names and asked them to rank from best to worst (**Activity Type C1**). Even though just before this activity, students clearly expressed they regarded combined names as "too long", the names students chose as "best name" overwhelmingly disregard that sentiment (see Table 2). Looking at the second, third, and fourth/last choice we mostly see clear 'winners' and shared 'losers'. For example, *amount*, *account*, and *start* received mixed positions, as did *years* and *term*, indicating they are considered equally bad, receiving very mixed positions.

However, while patterns are more or less similar for variables [A], [B], and [C], students demonstrate different preferences for variable [D]: some students move toward *profit* as the best name, but *endAmount* and *result* remain equally popular. During the whole-class discussion, we witnessed students strongly defending their choices, although without clear and convincing arguments. When prompted what makes *calculation* so much worse than *outcome* or *result* the following discussion took place (translated): *S: "Calculation of what?" // S: "It's too vague." // S: "'Result' is more specific." // T: "But how about result/outcome 'of what'?" // S: "Yes, but 'result' is much clearer." // S: "It's the outcome of the calculation."*

One explanation for the preference for *result* might be that students are influenced by the "meta-program", where they prioritize the result of the function or program over a better reflection of the content. After all, the name *result* or *outcome* gives little information on what that result is composed of.

By locating a naming mistake (**Activity Type E**), students experience first-hand how certain naming practices can be unintentionally misleading. To illustrate this, we presented again the same code, but now with the names *startingAmount*, *interestRate*, *endAmount*, and *termInYears*, asking the students to find the mistake (endAmount).

Despite these efforts, this activity proved very difficult for the students. Almost half of the students indicated they did not know, and only three answered correctly, while four students pointed to *interestRate* and another four to *termInYears*. Since none of the students could explain why *endAmount* was misleading, the teacher attempted to make students get there by asking questions such as: "What does the variable represent?", "What does the name *endAmount* represent?", "What do you expect the program to deliver as output when the variable is called *endAmount*?", and "Does *endAmount* mean the same as *profit*?". Only with the last question, some students started to realize the mistake, but the majority still needed an explicit example. While the activity served as an eye-opener to many students, some continued to resist, commenting that "endAmount" still accurately represents the amount at the 'end' of the calculation. This further indicates that these students consider a certain 'meta-level' when choosing names, in a similar fashion as the names *result* and *outcome*, rather than choosing a name more indicative of its actual contents.

Table 3: Names chosen as most misleading (assignment 3: C2)

Variable	**Version 1** (n=19)	**Version 2** (n=11)
A	amount (3)	account (3)
B	percentage (5)	interestRate (1)
C	years (3)	**time** (7)
D	**calculation** (8)	result (0)

By having students rank all names from a code snippet from most to least misleading (**Activity Type C2**), we create an authentic context in which students are stimulated to further explore how names can have negative effects. We prepared two different versions, each having a different set of (misleading) names. The most misleading names per version are presented in Table 3 and show consistency with previous activities.

5.4 Assignment 4: Activity Type D and A2

By comparing identical codes with opposite naming styles (**Activity Type D**) students get another opportunity to experience the effect of naming choices. Students were unanimous in their opinion on which code they found easier (words) and most also indicated that this version looks more like their own programs. However, students were split in half on which approach was more efficient, showing that some prefer letters and abbreviations for efficiency, which is consistent with the opinions we found during Assignment 1 when students noted that paying attention to naming is too time-consuming and can compete with other objectives.

When prompted again with more general questions on naming practices (**Activity Type A2**), we see that, compared to the start of the lesson, students are slightly more concerned. In more detail, we see four types of responses to the question *do you find naming a problem for software developers?* (scale 1-10). Those answering on the lowest end of the scale (1, never) say "it's not difficult" and "if naming would be a problem we have a big issue". Those rating 2-3 *and* those rating 7 and up note that naming "is no effort at all and helps immensely", whereas those rating 4-6 note that naming "costs a lot of time" and is a "big effort". Reactions to the question *do you find naming worth the effort?* can be found in Figure 4, which shows conflicting perspectives, especially among the middle group, and room for further dialogue.

5.5 Questionnaire (Workshop Evaluation)

Students rated the workshop with an average score of 7.6 out of 10 (n=21). Assignment Three was found most informative (n=8), followed by Assignment One (n=6) and Two (n=4). Half of the

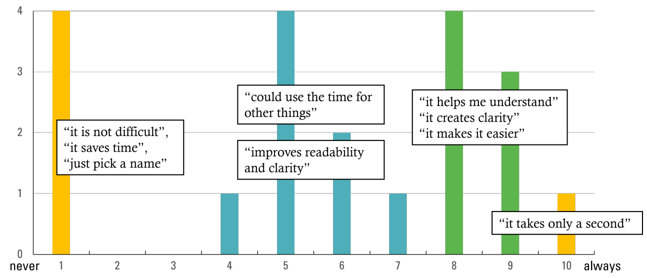

Figure 4: Results (n=19): is naming worth the effort?

students (n=11) indicated that they plan to pay more attention to naming. These students also noted the following takeaways from the workshop: "naming is important," "the hows and whys of naming," and "to make deliberate decisions." On the other hand, eight students indicated that they do not plan to pay more attention to naming, even though their takeaways also included "names should be readable for another person too" and "always use correct names." These students may feel they already pay enough attention to naming, as one student also indicated "I didn't learn anything I didn't already know." Three students concluded the questionnaire by noting that they found the lesson enjoyable and very informative.

6 PRACTICAL IMPLICATIONS

Since naming practices influence code comprehension in various ways, but are currently taught inconsistently in introductory programming education [23, 24], we experimented with an adaptive and interactive teaching approach to naming practices, supporting critical thinking, reflection, and deliberate naming choices. We presented five Activity Types and their design, discussed how they can be adapted for implementation, and showed our experiences with them. Below we highlight our insights and recommendations regarding the adoption of naming practices in a curriculum.

Use versatile activities. Although we used C# code, the various activities support any programming language that uses labels for variables. Moreover, they support selecting and adapting existing programs from student code or examples from a running or previous course. This versatility helps create authentic experiences while limiting the time needed to develop new materials.

Encourage whole-class discussion. Whole-class discussions proved extremely valuable in revealing potential issues (see below) and creating opportunities for reflection, nudging students to re-evaluate naming practices. We found the use of a third-party online polling tool beneficial in supporting the dialogue, but expect that other ways of promoting dialogue would have a similar effect, as long as students can share their choices and opinions with others and reflect upon those of others as well. Additionally, we have seen that reflection is deepened by also including activities that focus on individual consolidation of opinions after discussion.

Address and counter obstacles by offering various experiences. Perception-focused activities (Type A) are successful in revealing potential issues for paying attention to naming practices, as students reported conflicting opinions and experiences. At least half of the students perceive paying attention to naming as (too) time-consuming, inefficient, and irrelevant, while also reporting finding the act of naming easy. If left unattended, students could take a long time before experiencing the benefits of good naming choices. Considering that naming affects comprehension and heavily depends on many factors while teachers expect students to 'figure out naming by themselves' [23], a nonchalant attitude is unhelpful. Including a diverse set of activities provides (guided) opportunities for students to 'figure out naming' by practicing, experiencing, and reflecting on different naming styles, hence increasing the chances to gain new perspectives. In this way, we can 'prime' students to adapt to a wide range of names and encourage the adoption of deliberate naming choices, without teaching specific naming styles.

Highlight the pitfalls surrounding naming. At the same time, students already recognize reasons for paying attention, most often to improve code clarity and support code understanding for themselves and others. However, reflection based on ranking names (Activity Type C) and locating a naming mistake (Activity Type E) revealed that students are unaware of the limitations of certain naming choices and in particular of how naming choices can (unintentionally) deceive a reader. Through the activities, students showed increased awareness of how names are interpreted differently by different people, or when names, with the best intentions, do not accurately reflect the variable's contents. The teacher-led whole-class discussion was vital in increasing this awareness, highlighting both the strengths and weaknesses of naming choices. Our results show that using a single code snippet for various activities (Assignment Three) supports 'aha-moments' while using a variety of codes creates repeated practice with a wider range of examples.

7 CONCLUDING REMARKS

In this work, we implemented a set of educational activities on variable naming in the single context of a vocational program with two small-size workshops totaling 27 (male) students. We stress that student participation, group dynamics, and teacher involvement all influence the outcomes of the discussions. However, we encourage practitioners to experiment with the activities also in other contexts, such as higher education or primary/secondary education. It would be especially interesting to compare these results with a similar workshop given to more experienced students, specifically related to our encountered (explicit) preference for names such as 'result', 'endAmount', and 'outcome'. We find this preference intriguing, as such names seem to be tailored to the 'meta-program' rather than the contents of the variable, and are known to be problematic for code comprehension [8, 20]. Finally, it could be interesting to use the activities with other programming languages, or even try out naming activities in a language the students are *not* familiar with, to see the effects on comprehension. Regarding comprehension, we also suggest experimenting with presenting code *without* an explanation of its purpose, its output, and/or a description of the contents of the variables. This likely increases difficulty and completion time, but would also train reading and program comprehension skills.

REFERENCES

[1] Philip C. Abrami, Robert M. Bernard, Eugene Borokhovski, David I. Waddington, C. Anne Wade, and Tonje Persson. 2015. Strategies for Teaching

Students to Think Critically: A Meta-Analysis. *Review of Educational Research* 85, 2 (2015), 275–314. https://doi.org/10.3102/0034654314551063 arXiv:https://doi.org/10.3102/0034654314551063

[2] Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. 2016. Linguistic antipatterns: what they are and how developers perceive them. *Empirical Software Engineering* 21, 1 (Feb. 2016), 104–158. https://doi.org/10.1007/s10664-014-9350-8

[3] Eran Avidan and Dror G. Feitelson. 2017. Effects of Variable Names on Comprehension: An Empirical Study. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 55–65. https://doi.org/10.1109/ICPC.2017.27

[4] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G. Feitelson. 2017. Meaningful Identifier Names: The Case of Single-Letter Variables. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 45–54. https://doi.org/10.1109/ICPC.2017.18

[5] Dave Binkley, Dawn Lawrie, Steve Maex, and Christopher Morrell. 2009. Identifier length and limited programmer memory. *Science of Computer Programming* 74, 7 (2009), 430–445. https://doi.org/10.1016/j.scico.2009.02.006

[6] Jürgen Börstler, Harald Störrle, Daniel Toll, Jelle van Assema, Rodrigo Duran, Sara Hooshangi, Johan Jeuring, Hieke Keuning, Carsten Kleiner, and Bonnie MacKellar. 2017. "I Know It When I See It": Perceptions of Code Quality. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education* (Bologna, Italy) *(ITiCSE '17)*. Association for Computing Machinery, New York, NY, USA, 389. https://doi.org/10.1145/3059009.3081328

[7] Robert H. Ennis. 2018. Critical Thinking Across the Curriculum: A Vision. *Topoi* 37, 1 (March 2018), 165–184. https://doi.org/10.1007/s11245-016-9401-4

[8] Dror G. Feitelson. 2023. From Code Complexity Metrics to Program Comprehension. *Commun. ACM* 66, 5 (apr 2023), 52–61. https://doi.org/10.1145/3546576

[9] Dror G. Feitelson, Ayelet Mizrahi, Nofar Noy, Aviad Ben Shabat, Or Eliyahu, and Roy Sheffer. 2022. How Developers Choose Names. *IEEE Transactions on Software Engineering* 48, 01 (jan 2022), 37–52. https://doi.org/10.1109/TSE.2020.2976920

[10] Edward M. Gellenbeck and Curtis R. Cook. 1991. *An Investigation of Procedure and Variable Names as Beacons During Program Comprehension.* Technical Report. USA. https://doi.org/10.5555/891020

[11] Elena L. Glassman, Lyla Fischer, Jeremy Scott, and Robert C. Miller. 2015. Foobaz: Variable Name Feedback for Student Code at Scale. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 609–617. https://doi.org/10.1145/2807442.2807495

[12] Abdul R.M. Gobil, Zarina Shukor, and Itaza A. Mohtar. 2009. Novice difficulties in selection structure. In *2009 International Conference on Electrical Engineering and Informatics*, Vol. 02. 351–356. https://doi.org/10.1109/ICEEI.2009.5254715

[13] Shuchi Grover and Satabdi Basu. 2017. Measuring Student Learning in Introductory Block-Based Programming: Examining Misconceptions of Loops, Variables, and Boolean Logic. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 267–272. https://doi.org/10.1145/3017680.3017723

[14] Johannes Hofmeister, Janet Siegmund, and Daniel V. Holt. 2017. Shorter identifier names take longer to comprehend. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 217–227. https://doi.org/10.1109/SANER.2017.7884623

[15] Lisa C. Kaczmarczyk, Elizabeth R. Petrick, J. Philip East, and Geoffrey L. Herman. 2010. Identifying Student Misconceptions of Programming. In *Proceedings of the 41st ACM Technical Symposium on Computer Science Education* (Milwaukee, Wisconsin, USA) *(SIGCSE '10)*. Association for Computing Machinery, New York, NY, USA, 107–111. https://doi.org/10.1145/1734263.1734299

[16] Daniel Keller. 1990. A guide to natural naming. *ACM SIGPLAN Notices* 25 (1990), 95–102.

[17] Min-Young Kim and Ian A.G. Wilkinson. 2019. What is dialogic teaching? Constructing, deconstructing, and reconstructing a pedagogy of classroom talk. *Learning, Culture and Social Interaction* 21 (2019), 70–86. https://doi.org/10.1016/j.lcsi.2019.02.003

[18] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2006. What's in a Name? A Study of Identifiers. In *14th IEEE International Conference on Program Comprehension (ICPC'06)*. 3–12. https://doi.org/10.1109/ICPC.2006.51

[19] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2007. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering* 3, 4 (Dec. 2007), 303–318. https://doi.org/10.1007/s11334-007-0031-2

[20] Andrea Schankin, Annika Berger, Daniel V. Holt, Johannes C. Hofmeister, Till Riedel, and Michael Beigl. 2018. Descriptive Compound Identifier Names Improve Source Code Comprehension. In *Proceedings of the 26th Conference on Program Comprehension* (Gothenburg, Sweden) *(ICPC '18)*. Association for Computing Machinery, New York, NY, USA, 31–40. https://doi.org/10.1145/3196321.3196332

[21] Armstrong A. Takang, Penny A. Grubb, and Robert D. Macredie. 1996. The effects of comments and identifier names on program comprehensibility: an experimental investigation. *J. Program. Lang.* 4 (1996), 143–167.

[22] Barbee E. Teasley. 1994. The effects of naming style and expertise on program comprehension. *International Journal of Human-Computer Studies* 40, 5 (1994), 757–770. https://doi.org/10.1006/ijhc.1994.1036

[23] Vivian van der Werf, Alaaeddin Swidan, Felienne Hermans, Marcus Specht, and Efthimia Aivaloglou. 2024. Teachers' Beliefs and Practices on the Naming of Variables in Introductory Python Programming Courses. In *Proceedings of the 46th International Conference on Software Engineering: Software Engineering Education and Training* (Lisbon, Portugal) *(ICSE-SEET '24)*. Association for Computing Machinery, New York, NY, USA, 368–379. https://doi.org/10.1145/3639474.3640069

[24] Vivian Van Der Werf, Min Yi Zhang, Efthimia Aivaloglou, Felienne Hermans, and Marcus Specht. 2023. Variables in Practice. An Observation of Teaching Variables in Introductory Programming MOOCs. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1* (Turku, Finland) *(ITiCSE 2023)*. Association for Computing Machinery, New York, NY, USA, 208–214. https://doi.org/10.1145/3587102.3588857