How do Scratch Users Name Variables and Functions?

Author1
Uni1
Address1
Email: email1.com

Author2
Uni2
Address2
Email: email2.com

***Abstract*—The abstract goes here.**

I. Introduction

The naming of identifiers in the source code has been extensively studied (see, e.g., recent studies of this subject [1], [2], [3], [4], [5], [6], [7], [8]). Still, the impact of the variable name choice on code readability and maintainability is controversial, as witnessed, e.g., by recent studies of Beniamini et al. [3] and Hofmeister et al. [5] reaching contradictory conclusions. Furthermore, computer science and programming education seem to focus on the programming concepts and the syntax of the languages as opposed to practices in naming variables and identifiers. Indeed, while "meaningful variable names" are advocated by some teachers [9] and practitioners [10] neither the ACM Curriculum Guidelines for Undergraduate Programs in Computer Science[1] nor the Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering[2] discuss this topic. In fact, one of the most common examples in many programming languages is the use of *"foo"* and *"bar"* naming for variables and functions. These two identifiers have meaningless names, and to some extent, they represent a refusal to name, suggesting the learner that the name is less important, or irrelevant, to the programming task.

Recent studies have focused on the analysis of identifier naming in software code repositories. In this paper, we give attention to novice programmers with the Scratch visual language. When it comes to novice programmer and learners, it is more important for the software community to understand conceptions in naming that are prevalent. These users are the future developers, and at some point, they need to follow an implicit or explicit guideline of naming and collaborating in larger software repositories. To this end, we have analyzed 250,000 Scratch projects previously published by [11]. Scratch is a block-based visual language that was developed by MIT with the aim of helping young people learn the basic concepts of programming and collaboration. Scratch has recently become very popular among school-age children and even in introduced as part of the school curriculum as a means to teach programming [12]. Analyzing Scratch programs will give researchers and software engineering community a perspective on naming from learners making their first step in programming and might turn to be the future programmers. In this paper we therefore aim at understanding *how do Scratch users name variables and procedures?*

To answer this question we first replicate two studies from a recent paper by Beniamini et al. [3]. We start by investigating the distribution of the lengths of variable and function names across Scratch projects. Next, we focus on popularity of single-letter names such as i and x as variable and function names.

To augment the previous study of Beniamini et al. [3] we focus on Scratch-specific features in naming identifiers. In particular, we aim at understanding to what extent spaces within identifiers (e.g., variable *max i*), digits as identifiers (e.g., variable 6) and parameters inserted the middle of the procedure name (**Give an example**). Those features are missing from the main-stream programming languages: if they will prove to be popular among the Scratch developers, one could advocate that they should be integrated in the main-stream programming languages as well.

More questions: analysis of the peak value of 5, what is the most recurrent name? Is it expected or justified? More questions: variable length based on category of the Scratch program: game animation, ... More questions: variable length in relation to Dr. Scratch CT

II. Background and motivation

Naming identifiers in software code has been studied extensively in the past decades [1], [13], [2], [3], [4], [14], [5], [15], [6], [7], [16], [8]. In practice, identifiers constitute a major part of the source code: e.g., Deißenböck and Pizka found that in Eclipse 3.0M7 which is tantamount to 2 MLoC, 33% of the tokens and 72% of characters correspond to identifiers [17]. **AS falling asleep, sorry** For a human to read that code, it is crucial to understand what the identifier means, and then can deduce what the code does. With no surprise, several studies have confirmed the link between good identifier naming and code's readability and comprehension. The comprehension is not a target in its own; the true reason is that better comprehension lets the developers perform maintenance tasks more effectively and efficiently. But what is a good naming approach? It is out of this paper's scope to explore the various recommendations of good identifier naming. However, it is worth to mention that there are different perspectives on what a good name is. Some researchers emphasize the usage of actual

[1] http://www.acm.org/education/CS2013-final-report.pdf
[2] http://www.acm.org/binaries/content/assets/education/se2014.pdf

and complete words from a dictionary, or known abbreviations, which reflect the context of the program's purpose. Others argue that consistency in naming style is the most important. The usage of single-letter identifiers attracts much attention in research. For programmers, it is tempting to choose single-letter identifiers for quicker code writing that involves less mental load on the choice of a name. Additionally, single-letter named variables such as *"i"* or *"j"* have become almost a standard choice for index values when coding loops. Research has shown however that shorter identifier names are longer to comprehend, and the length of a variable should reflect its scope.

Despite the agreement on the importance of identifier names, and efforts to introduce naming guidelines and additional tools to help the programmer choose better names, developers find giving appropriate names to identifiers as a difficult task. In the end, it is the decision of the individual writing the code, and many factors may contribute to that decision. One area to consider here is the computer science and in particular programming education. There seems to be a great focus on the programming concepts, while less to zero attention is given to beautifying the code. For example, the usage of the identifier names *"foo"* and *"bar"* is prevalent in code examples. These names have no real meaning, and the student cannot link them to what the code does.

We are inspired by the work done by **ref** where they explored the single-letter naming for multiple software repositories from different programming languages. It is important for the software community to understand the patterns in which software developers apply naming to identifiers. We argue that this understanding can improve the quality of the guidelines and introduce research-supported tools that meet the needs of the programmers. For this sake, we believe it is even more important to focus on the novice programmers, students, and learners. For these future developers, the usage of a particular naming pattern in this level forms a (mis)conception that could move along with them to other programming languages and future careers. The work was done by **add ref** excludes the visual languages which have recently become a favorable choice for elementary and high schools as an introduction to programming, for example, Scratch and Alice. These block-based languages allow the user to create and assign their identifiers for variables and blocks. By studying these identifiers, we can understand better how novice programmers apply standard and language-specific naming, and how it compares to other textual based languages previously explored in the literature.

III. Relevant Scratch Concepts

This paper is by no means an introduction into Scratch programming, we refer the reader to [18] for an extensive overview. To make this paper self-contained, however, we explain a number of relevant concepts in this section.

Scratch is a block-based programming language aimed at children, developed by MIT. Scratch can be used to create games and interactive animations, and is available both as a stand-alone application and as a web application. Figure 1 shows the Scratch user interface in the Chrome browser.

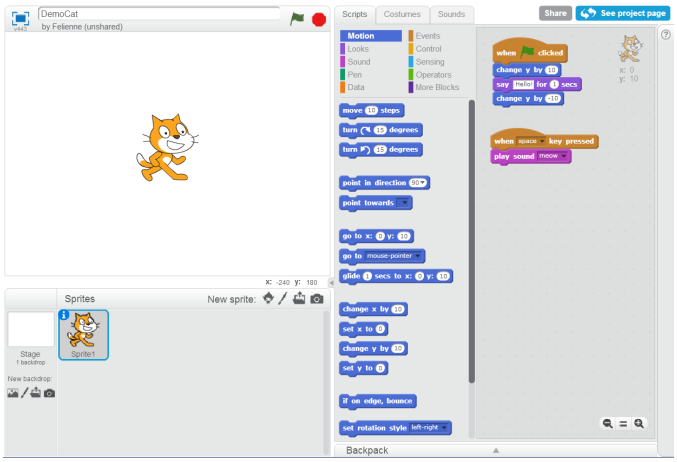

Fig. 1. The Scratch user interface consisting of the 'cat' sprite on the left, the toolbox with available blocks in the category 'motion' in the middle and the code associated with the sprite on the right. The upper right corner shows the actual location of the sprite.

A. Sprites

Scratch code is organized by 'sprites': two-dimensional pictures each having their own associated code. Scratch allows users to bring their sprites to life in various ways, for example by moving them in the plane, having them say or think words or sentences via text balloons, but also by having them make sounds, grow, shrink and switch costumes. The Scratch program in Figure 1[3] consists of one sprite, the cat, which is Scratch's default sprite and logo. The code in the sprite will cause the cat to jump up, say "hello", and come back down, when the green flag is clicked, and to make the 'meow' sound when the space bar is pressed.

B. Events

Scratch is *event-driven*: all motions, sounds and changes in the looks of sprites are initiated by events. The canonical event is the 'when Green Flag clicked', activated by clicking the green flag at the top of the user interface. In addition to the green flag, there are a number of other events possible, including key presses, mouse clicks and input from a computer's microphone or webcam. In the Scratch code in Figure 1 there are two events: 'when Green Flag clicked' and 'when space key pressed'

C. Scripts

Source code within sprites is organized in scripts: a script always starts with an event, followed by a number of blocks. The Scratch code in Figure 1 has two distinct scripts, one started by clicking on the green flag and one by pressing the space bar. It is possible for a single sprite to have multiple scripts initiated by the same event. In that case, all scripts will be executed simultaneously.

[3]https://scratch.mit.edu/projects/97086781/

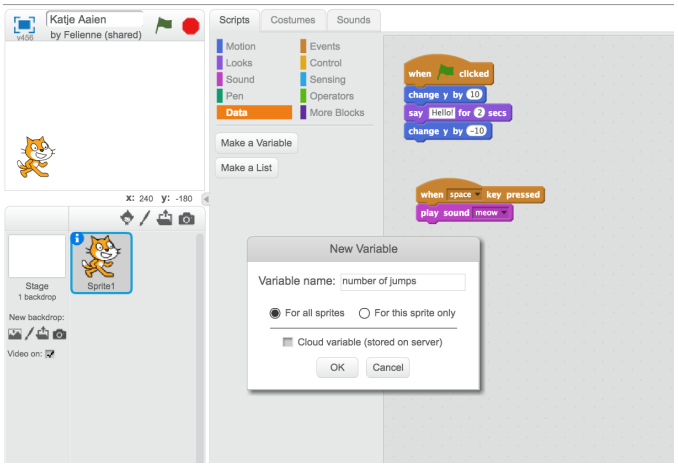

Fig. 2. The Scratch user interface to create a variable

D. Remixing

Scratch programs can be shared by their creators in the global Scratch repository[4]. Shared Scratch programs can be 'remixed' by other Scratch users, which means that a copy of this program is placed in the user's own project collection, and can be then further changed. The 'remix tree' of projects is public, so users can track which users remix their programs, a bit similar forking in GitHub. Contrary to forking however, changes upstream cannot be integrated back into the original project. Because Scratch users can remix each others programs, maintainability of programs is important. A given Scratch program can be read and adapted by many children, there are projects on the Scratch home page which are remixed hundreds of times. **maybe this part about importance should be move somewhere else? this section might be skipped.**

E. Variables

Like most textual languages, Scratch users can use variables. Variables are untyped, but have to be 'declared' through the Scratch user interface, shown in Figure 2. This figure also shows that, contrary to most programming languages, variable names in Scratch may contain spaces.

F. Procedures

Scratch also allows users to create their own blocks, called procedures. They can have input parameters, and labels in between them. Functions are created with an interface similar to the one to create variables. Figure 3 shows the definition and invocation of a procedure.

IV. Research design and dataset

A. Overall design

As our goal is to compare the naming practices among the Scratch-developers with those of the developers in the mainstream programming languages we start by partially

[4]https://scratch.mit.edu/explore/projects/all/

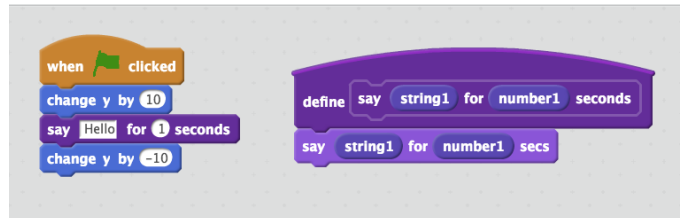

Fig. 3. Scratch code to define and invoke a procedure

replicating the recent work of Beniamini et al. on the use of single-letter variables in Java, C, PHP, Perl and JavaScript [3]. In terms of the classification of Shull et al. [**?**] we perform a dependent replication of the studies summarized in Figures 1 and 2 of the original work by Beniamini et al. [3]. Inherently, the programming language is the only factor we vary when compared to the original study. However, as Scratch programs are not available on GitHub that has been used in the study of Beniamini et al., **Felienne, is this true?** we also had to change the source of the data.

Next, we perform a conceptual replication of the study of the single-letter variable types of Beniamini et al. [3]. While the original study has conducted a survey

Finally, to augment this study we also investigate the ways Scratch developers can benefit from Scratch-specific naming practices such as spaces in variable names, digits as variables and use of textual labels in between parameters.

B. Dataset

For this paper we use the dataset created by Aivaloglou and Hermans [11], consisting of 250.000 Scratch projects scraped from the Scratch website in March 2016. From this dataset, we have selected the projects that use variables or functions, together this are 73.473 projects (29%) of Aivaloglou and Hermans's original dataset. Variable use is more common that procedure use. In total 69.045 projects (27.6%) use variables, while 17.605 use procedures (7.0%). We used Python to process the original dataset and generate the graphs in this paper. The code we used is available at https://github.com/Felienne/ScratchVars.

C. Data analysis

To augment the visual comparison of the variable name data derived from Scratch as well as the programming languages considered by Beniamini et al. [3], we conduct statistical analysis.

Understanding differences in variable name lengths requires comparison of multiple distributions, traditionally performed as a two-step process consisting of (1) testing a global null hypothesis, that can be intuitively formulated as "all distributions are the same", using ANOVA or its non-parametric counterpart, the Kruskal-Wallis test, and (2) performing multiple pairwise comparisons of different distributions, testing specific subhypotheses such as "distributions 2 and 4 are the same". However, it has been observed that such a two-step approach

can result in inconsistencies when either the global null hypothesis is rejected but none of the pairwise subhypotheses is rejected or vice versa [19]. Moreover, it has been suggested that the Wilcoxon-Mann-Whitney test, commonly used for subhypothesis testing, is not robust to unequal population variances, especially in the unequal sample size case [20] Therefore, one-step approaches have been sought. We opt for one such approach, the $\widetilde{\mathrm{T}}$-procedure of Konietschke et al. [21]. This procedure is robust against unequal population variances, respects transitivity, and has been successfully applied in empirical software engineering [22], [23], [24]. In particular, we use the Tukey (all-pairs) contrasts to compare all distributions pairwise.

Furthermore, to understand differences and similarities between the distributions of single-letter variable names in different languages we represent each programming language as a 26-dimensional vector with the dimensions corresponding to 'a', ..., 'z', and apply hierarchical clustering. **why hierarchical?**

V. Results

This section presents an overview of our analysis of variable and function name use in our previously published Scratch dataset [11].

Unless indicated otherwise, all graphs show the number of projects a certain type of variable is used in, rather then the number of occurrences of the variable. Some projects access and update variables hundreds of times, so we felt that counting use would skew the graphs towards bigger programs. **maybe this paragraph fits with dataset more than with results? Not sure.**

Figure 4 shows the distribution of lengths in the corpus. The original study of Beniamini et al. has concluded that the single-letter variable names "are approximately as common as other short lengths except in PHP" and that "in C, Java, and Perl they make up 920% of the names."

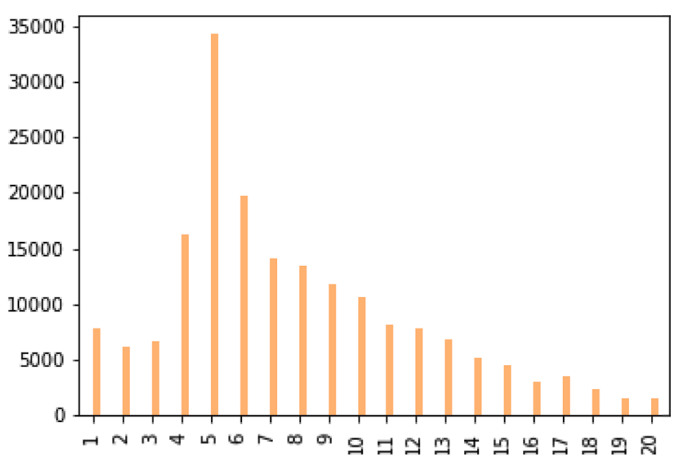

Fig. 4. Number of projects using variables of different lengths

still need to add the 20+ bucket if we want to have that

A. One letters

Figure 5 shows the distribution of variables of one letter in the corpus.

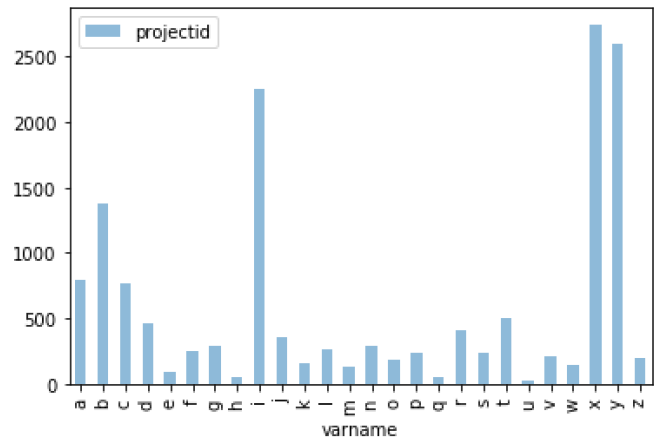

Fig. 5. Number of projects using variables of each of the one letter variables

Figure 6 shows the distribution of *functions* of one letter in the corpus.

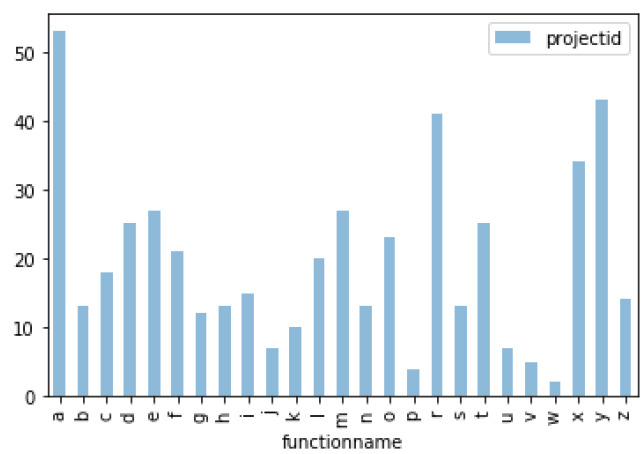

Fig. 6. Total occurrence of functions of one letter

Beniamini et al. [3] also explored the types of one letter variables, by performing a questionnaire among **summary of study here** .

This gave us the idea to also explore types of one letter variables in the context of our dataset. While Scratch variables have no types, we can deduce their use from our dataset, by exploring what assignments are made to variables and deducing the types from those. For example, the two variables in Figure 7 represent a string and an integer respectively. With this process we can compare our results to the types of Beniamini et al. [3]. Figure 8 shows the distribution of variables of one letter in the corpus.

Fig. 7. Two variables, one of type string and one of type integer

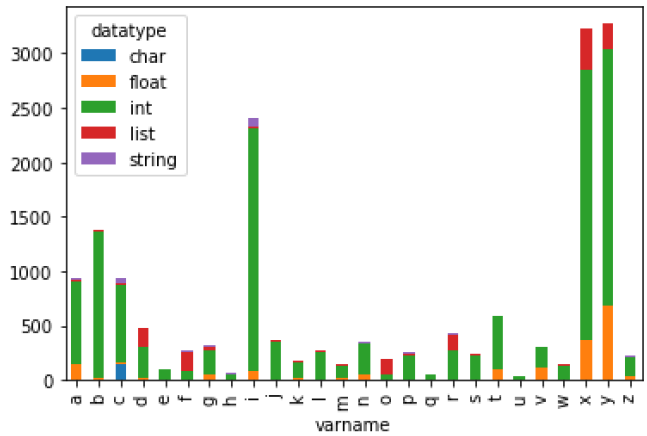

Fig. 8. Inferred types for variables of one letter

here we need to write some reflection on the results. Alaaeddin....? :-) Comparing the results with the other languages and between functions and variables.

VI. Scratch specific contructs

A. Use of spaces in variable names

Contrary to most textual programming languages, Scratch allows users to use spaces in variable names. This is quite commonly used, about 30.000 projects use one or more variables with a space in it, versus 60.000 that use only space-free variable names. Figure 9 shows the distribution of spaces in variable names. We have found that many introductory Scratch programming materials demonstrate the use of space free variables, and that children—and adults—that already have programming experience deem the use of spaces in variables as non-natural, even though arguable 'number of apples' is more natural that 'nApples'.

B. Use of numeric variable names

In addition to spaces in variable names, Scratch even allows the use of numbers and even floating point numbers as variables. We found 718 projects with integer variable names and 19 with floating point names. While their use is rare, we manually examined some projects and numbers are used in interesting and clever ways.

it seems these things are mainly used as constants. We could explore the dataset a bit to find out!

here we can show the tic tac toe example

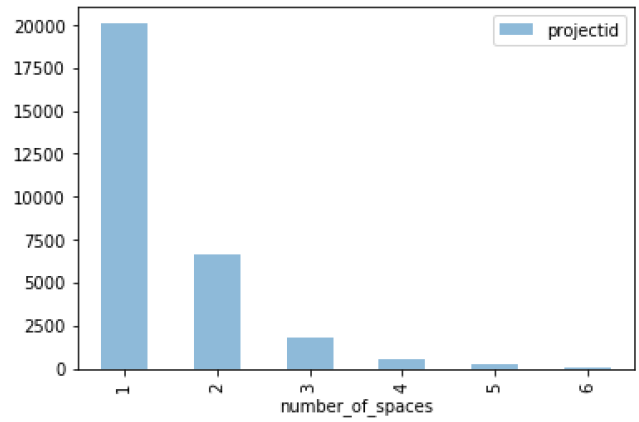

Fig. 9. Number of spaces in variable names

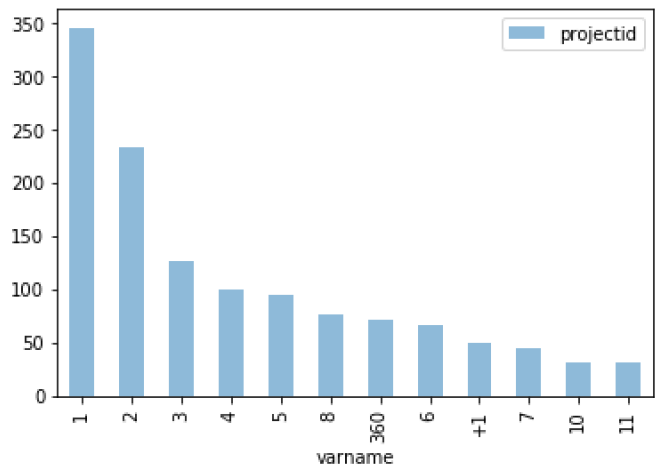

Fig. 10. Number of projects using the most popular numeric variables

C. Use of textual labels in between parameters

As can be seen in Figure 3, Scratch allows users to insert textual labels in between parameters in order to make functions more readable. Scratch built-in blocks use a similar syntax, for example in the "say ... for ... seconds" block. In total 4415 projects use textual labels, so their use if not that common. We do however find some interesting patterns. Figure 11 shows the most commonly used labels. Here we see some patterns common in textual languages, like the use of labels for the names of the parameters 'x:' and 'y:'. Furthermore we see the use of ')', which could come from textual languages too. Finally the use of the space is interesting, since Scratch already leaves some room between the parameters, also when a space is not used. The use of space as a separator could indicate that Scratch users feel room between variables is currently too small.

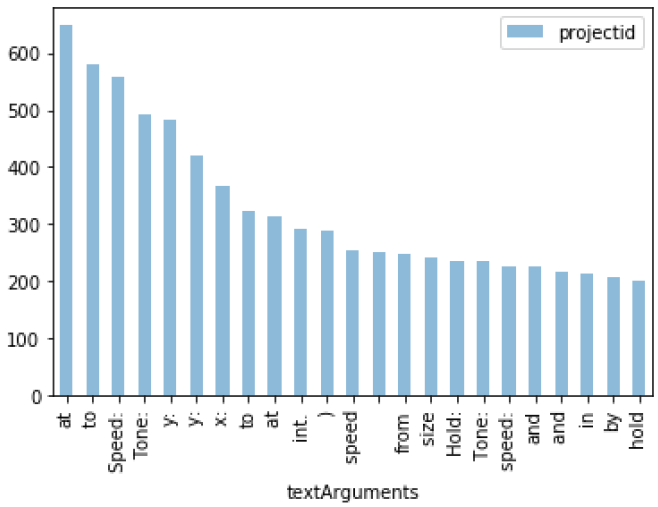

Fig. 11. Number of projects using the most popular numeric variables

VII. Discussion

VIII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] H. Aman, S. Amasaki, T. Sasaki, and M. Kawahara, "Empirical analysis of change-proneness in methods having local variables with long names and comments," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM 2015, Beijing, China, October 22-23, 2015*. IEEE, 2015, pp. 50–53. [Online]. Available: https://doi.org/10.1109/ESEM.2015.7321197

[2] E. Avidan and D. G. Feitelson, "Effects of variable names on comprehension an empirical study," in *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 55–65. [Online]. Available: http://dl.acm.org/citation.cfm?id=3101422

[3] G. Beniamini, S. Gingichashvili, A. Klein-Orbach, and D. G. Feitelson, "Meaningful identifier names: the case of single-letter variables," in *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017, Buenos Aires, Argentina, May 22-23, 2017*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 45–54. [Online]. Available: http://dl.acm.org/citation.cfm?id=3101421

[4] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *14th European Conference on Software Maintenance and Reengineering, CSMR 2010, 15-18 March 2010, Madrid, Spain*, R. Capilla, R. Ferenc, and J. C. Dueñas, Eds. IEEE Computer Society, 2010, pp. 156–165. [Online]. Available: https://doi.org/10.1109/CSMR.2010.27

[5] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE Computer Society, 2017, pp. 217–227. [Online]. Available: https://doi.org/10.1109/SANER.2017.7884623

[6] M. Lungu and J. Kurs, "On planning an evaluation of the impact of identifier names on the readability and quality of smalltalk programs," in *2nd International Workshop on User Evaluations for Software Engineering Researchers, USER 2013, San Francisco, CA, USA, May 26, 2013*. IEEE Computer Society, 2013, pp. 13–15. [Online]. Available: https://doi.org/10.1109/USER.2013.6603079

[7] G. Scanniello and M. Risi, "Dealing with faults in source code: Abbreviated vs. full-word identifier names," in *2013 IEEE International Conference on Software Maintenance, Eindhoven, The Netherlands, September 22-28, 2013*. IEEE Computer Society, 2013, pp. 190–199. [Online]. Available: https://doi.org/10.1109/ICSM.2013.30

[8] P. Tramontana, M. Risi, and G. Scanniello, "Studying abbreviated vs. full-word identifier names when dealing with faults: an external replication," in *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, M. Morisio, T. Dybå, and M. Torchiano, Eds. ACM, 2014, p. 64:1. [Online]. Available: http://doi.acm.org/10.1145/2652524.2652593

[9] T. Kato, Y. Kambayashi, and Y. Kodama, *Data Mining of Students' Behaviors in Programming Exercises*. Cham: Springer International Publishing, 2016, pp. 121–133. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39690-3_11

[10] K. Rother, *Cleaning Up Code*. Springer, 2017, pp. 195–212. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-39690-3_11

[11] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the scratch repository," in *ICER*, 2016.

[12] J.-M. Sáez-López, M. Román-González, and E. Vázquez-Cano, "Visual programming languages integrated across the curriculum in elementary school: A two year case study using scratch in five schools," *Computers & Education*, vol. 97, pp. 129–141, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131516300549

[13] N. Anquetil and T. C. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research, November 30 - December 3, 1998, Toronto, Ontario, Canada*, S. A. MacKay and J. H. Johnson, Eds. IBM, 1998, p. 4. [Online]. Available: http://doi.acm.org/10.1145/783160.783164

[14] B. Caprile and P. Tonella, "Restructuring program identifier names," in *2000 International Conference on Software Maintenance, ICSM 2000, San Jose, California, USA, October 11-14, 2000*. IEEE Computer Society, 2000, pp. 97–107. [Online]. Available: https://doi.org/10.1109/ICSM.2000.883022

[15] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "Effective identifier names for comprehension and memory," *ISSE*, vol. 3, no. 4, pp. 303–318, 2007. [Online]. Available: https://doi.org/10.1007/s11334-007-0031-2

[16] A. A. Takang, P. A. Grubb, and R. D. Macredie, "The effects of comments and identifier names on program comprehensibility: an experimental investigation," *J. Prog. Lang.*, vol. 4, no. 3, pp. 143–167, 1996. [Online]. Available: http://compscinet.dcs.kcl.ac.uk/JP/jp040302.abs.html

[17] F. Deißenböck and M. Pizka, "Concise and consistent naming [software system identifier naming]," in *International Workshop on Program Comprehension*, May 2005, pp. 97–106.

[18] K. Brennan, C. Balch, and M. Chung, *CREATIVE COMPUTING*. Harvard Graduate School of Education, 2014.

[19] K. R. Gabriel, "Simultaneous test procedures—some theory of multiple comparisons," *The Annals Mathematical Statistics*, vol. 40, no. 1, pp. 224–250, 1969.

[20] D. W. Zimmerman and B. D. Zumbo, "Parametric alternatives to the Student t test under violation of normality and homogeneity of variance," *Perceptual and Motor Skills*, vol. 74, no. 3(1), pp. 835–844, 1992.

[21] F. Konietschke, L. A. Hothorn, and E. Brunner, "Rank-based multiple test procedures and simultaneous confidence intervals," *Electronic Journal of Statistics*, vol. 6, pp. 738–759, 2012.

[22] B. Vasilescu, A. Serebrenik, M. Goeminne, and T. Mens, "On the variation and specialisation of workload - A case study of the gnome ecosystem community," *Empirical Software Engineering*, vol. 19, no. 4, pp. 955–1008, 2014. [Online]. Available: https://doi.org/10.1007/s10664-013-9244-1

[23] B. Vasilescu, A. Capiluppi, and A. Serebrenik, "Gender, representation and online participation: A quantitative study," *Interacting with Computers*, vol. 26, no. 5, pp. 488–511, 2014. [Online]. Available: https://doi.org/10.1093/iwc/iwt047

[24] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information & Software Technology*, vol. 74, pp. 204–218, 2016. [Online]. Available: https://doi.org/10.1016/j.infsof.2016.01.004

[25] G. Scanniello, D. Lo, and A. Serebrenik, Eds., *Proceedings of the 25th International Conference on Program Comprehension, ICPC 2017,*

Buenos Aires, Argentina, May 22-23, 2017. IEEE / ACM, 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3101414