How do scratch users name variables and functions? an analysis in scale

Author1
Uni1
Address1
Email: email1.com

Author2
Uni2
Address2
Email: email2.com

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

I. Introduction

The naming of identifiers in software code is considered, by several researchers, as one important aspect of its quality. Previous studies have shown a correlation between the proper naming of identifiers and the readability and comprehension of the code. Comprehension is linked to maintainability tasks and efficient performance of developers while performing them. As one scientist once said *"the code should be considered for human and less occasionally for a computer to execute"*.

In computer science and programming education, there seems a focus on the programming concepts and the syntax of the languages. Programming courses give less attention to naming variables and identifiers. 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, which give the learner the conception that the name is less important, or irrelevant, to the programming task.

Recent studies have focused on the analysis of 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 will be 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 240,000 scratch projects. 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 introduced as part of the curriculum in some countries as a means to teach programming. 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 aim at answering the following research questions:

RQ1 How do scratch users name variables and procedures?
(a) The length distribution of variable and function identifiers across projects.
(b) The usage of single-letter variables.
RQ2 How do Scratch users use language-dependent features in naming identifiers? In particular, to what extent are the following usage patterns popular within scratch projects?
(a) Using spaces within identifiers
(b) Using numeric digits as identifiers
(c) Using the parameter in the middle name for procedures]

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 have been studied extensively in the past decades. in practice, identifiers are a major part of software code as one study found that within the Eclipse code, which has 2 MLoC, 33% of the tokens and 72% of characters are dedicated to identifiers. 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 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. Scratch related concepts

In this section, we highlight some of the basic concepts of Scratch. This paper is by no means about scratch programming. However, clarifying some Scratch concepts is necessary for the paper self-explanation. Scratch is **Complete** .

IV. Research design and dataset

A. Overall design

here we need to tell someth about replications, see the paper by Shull et al on importance of replications in software engineering.

B. Dataset

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. [1], 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 [2]. 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 [3] Therefore, one-step approaches have been sought. We opt for one such approach, the $\widetilde{\mathbf{T}}$-procedure of Konietschke et al. [4]. This procedure is robust against unequal population variances, respects transitivity, and has been successfully applied in empirical software engineering [5], [6], [7]. 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

Figure 1 shows the distribution of lengths in the corpus.

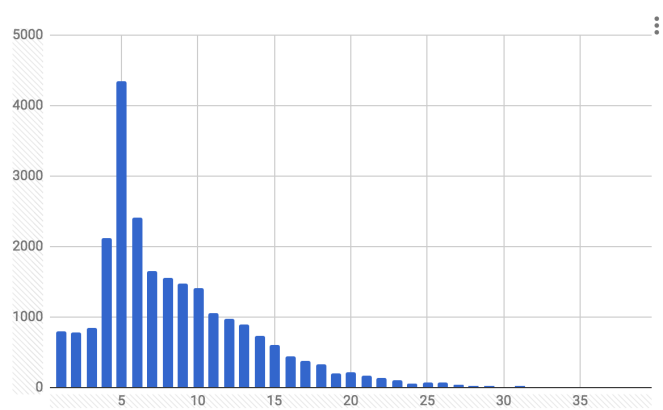

Fig. 1. Total occurrence of variables of different lengths

we could sample some 5 letter identifiers here to see what they usually look like since they are an interesting peak?

in the ICPC paper the diagram goes to 20 and then just has 20+ we could do that too?

A. *One letters*

Figure 2 shows the distribution of variables of one letter in the corpus.

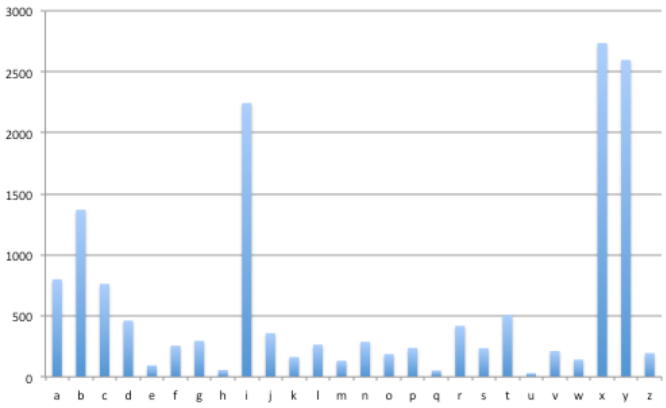

Fig. 2. Total occurrence of variables of one letter

Scratch is not a typed language, variables can contain strings or numbers without declarations or casting. However, we can infer the types by attempting to cast them to a float or an int. As such we did obtain types, allowing us to compare to the ICPC paper.

Figure 3 shows the distribution of variables of one letter in the corpus.

here we can reflect on the differences with "real" languages and the prevalence of ints.

B. *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 4 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'.

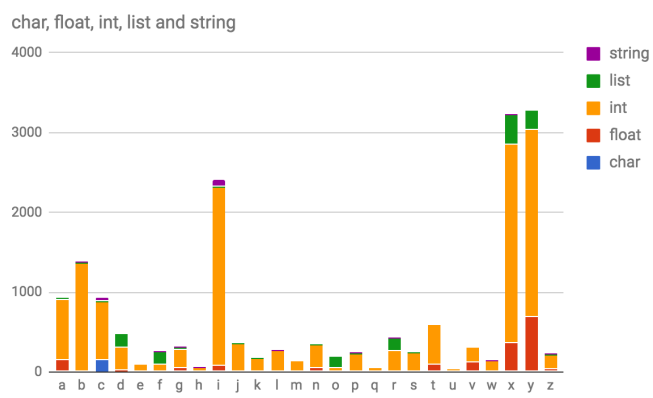

Fig. 3. Inferred types for variables of one letter

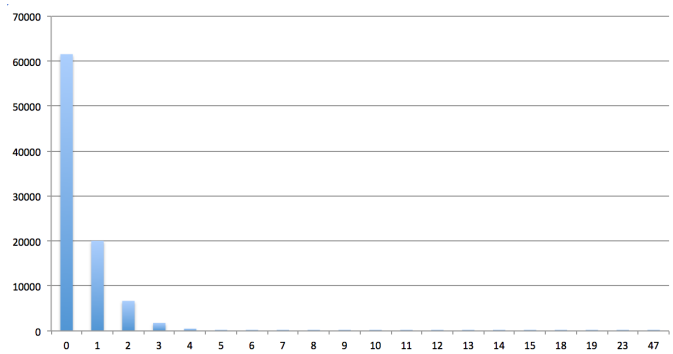

Fig. 4. Number of spaces in variable names

C. *Use of non-letter 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.

here we can show the tic tac toe example

VI. Discussion

VII. Conclusions

VIII. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

References

[1] 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

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

[3] 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.

[4] 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.

[5] 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

[6] 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

[7] 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