

# How do Scratch Users Name Variables and Procedures?

Author1

Uni1

Address1

Email: email1.com

Author2

Uni2

Address2

Email: email2.com

**Abstract**—Research shows the importance of choosing good names to identifiers in software code. More meaningful names improve the comprehension and readability of software code, which leads to increased efficiency in maintenance tasks. In particular, several guidelines encourage long and descriptive variable names. A recent study analyzed the use of variable names in five popular programming languages, with a focus on single-letter variable names because of the apparent contradiction between their frequent use and the fact that these variables violate the aforementioned guidelines.

In this paper, we focus on single-letter variables in Scratch, a popular block-based visual language which focuses on teaching children programming. We start by replicating the previously mentioned study for Scratch. We augment this study by analyzing single-letter procedure names, as well as by investigating the use of Scratch specific naming patterns including spaces in variable names and textual labels in procedure definitions and calls.

The results of our analysis show that Scratch users often use variable and procedure names between 4 and 10 characters in length. For the single letter variables, the most recurring names are x, y and i. While single letter procedures are less popular, the usage of letters is not linked with one character. *I do not understand the meaning of the previous sentence*

Concerning Scratch specific features, 20% of projects have variables that include spaces in the name. The usage of textual string between parameters appears as not so common, however textual patterns used imply an inference from textual languages by using brackets for example. Finally, when compared to the other programming languages, Scratch variable names tend to be longer than of the mainstream programming languages, and the usage of single uppercase letters seems to be similar to the pattern found in Perl, while for the lowercase letters—to the pattern found in Java.

*AS: I'm missing conclusion, some kind of take away message.*

## 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 seems 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

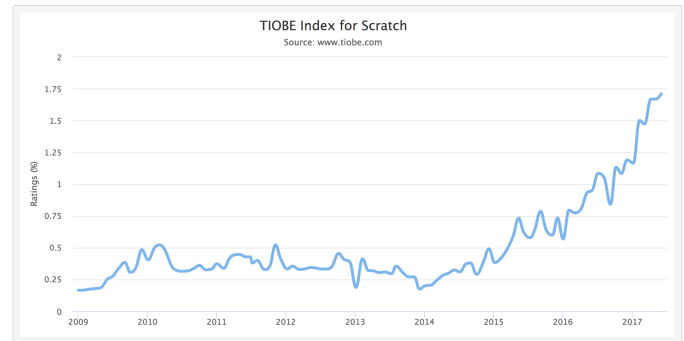


Fig. 1: TIOBE Programming Community index: evolution of the popularity of Scratch: <https://www.tiobe.com/tiobe-index/scratch/>

in Computer Science<sup>1</sup> nor the Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering<sup>2</sup> discuss this topic. In fact, standard metasyntactic variables used in syntax examples are “foo” and “bar” [11]. The names of these identifiers are meaningless, and to some extent, they represent a refusal to name, suggesting the learner that naming is less important, or irrelevant, to the programming task.

In this paper, we analyze the use of variable and procedure names in Scratch. Scratch is a block-based visual language 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 in several countries has been introduced as part of the school curriculum as a means to teach programming [12], [13]. Moreover, the overall popularity of Scratch is witnessed by Scratch being currently rated 19 in the TIOBE index<sup>3</sup>, topping such languages as Lua, Scala and Groovy and since early 2014 exhibiting an increasing trend shown in Fig. 1.

We believe that there are several reasons why understanding the naming practices in the Scratch community is important. First, it is important for the Scratch community itself as bad naming practices can easily propagate from one program to another through ‘remixing’ [14], [15], code sharing practice

<sup>1</sup><http://www.acm.org/education/CS2013-final-report.pdf>

<sup>2</sup><http://www.acm.org/binaries/content/assets/education/se2014.pdf>

<sup>3</sup><https://www.tiobe.com/tiobe-index/>

similar to GitHub forking. Second, it is important for researchers. Software engineering researchers can learn how to support novice programmers, taking their first steps in programming. Researchers on software engineering education can obtain insights in how to define naming guidelines for educational materials, and analyzing the differences between Scratch and textual languages can help in supporting the transition from visual languages to textual ones [16], [17].

We start by a **general discussion of naming practices** in Scratch and analyze the previously published collection of 250000 Scratch projects [18]. We replicate two studies from a recent paper by Beniamini et al. [3]. Similarly to Beniamini et al. we investigate the distribution of the lengths of variable names and study popularity of single-letter variable names such as *i* and *x*. As opposed to Beniamini et al. that focused on variable naming in five mainstream programming languages we focus on Scratch. Furthermore, while Beniamini et al. solely focused on the names of the variables we repeat their study for procedures as well.

Variable and procedure names in Scratch range mostly between 4 and 10 characters. For the single letter variables, the most commonly used names are *x*, *y* and *i*, procedures—*a* and *r*. When compared to the other programming languages, we observe that single-letter variable names are less common in Scratch and that overall Scratch variables have longer names. The usage of single uppercase letters is similar to the pattern found in Perl, for the lowercase—to the pattern found in Java.

Next 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., a variable named 6) and textual labels used between the parameters. For example instead of printing the first *n* letters from a string *s* with a procedure called “`printnof(n,s)`”, in Scratch one can define a procedure called “say *n* characters from text *s*”, as shown in Figure 4. This feature exists in some textual languages too (most notable in SmallTalk) but is not common in most mainstream languages.

Investigating the use of these Scratch specific naming patterns is interesting to understand their role in novice programming. If they are popular among Scratch developers, this might be because they ease novice programming, and that means one could even advocate that these features should be integrated in the mainstream programming languages, if only to ease the transition from block-based languages into textual languages.

Spaces in variable names are common: 20% of projects use this feature. Digits as identifiers are rarely used, and mostly represent constants or parts of the data structure. The usage of textual string between parameters appears as not so common, however textual patterns used imply an

inference from textual languages, e.g., by using brackets.

## II. RELATED WORK

Naming identifiers in software code has been studied extensively in the past decades [1], [19], [2], [3], [4], [20], [5], [21], [6], [7], [22], [23], [8]. In practice, identifiers constitute a major part of the source code: e.g., Deißeböck 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 [24].

For a human to read that code, it is crucial to understand what the identifier means, and deduce what the code does. With no surprise, several studies have investigated the link between identifier naming and code readability/comprehension [2], [5], [21], [22], [23] or identifier naming and externally observable aspects of the software development process that are expected to be affected by comprehension such as change-proneness [1], quality [4], [6] and presence of faults [7], [8]. Caprile and Tonella [25] have explicitly addressed naming of functions, while Caracciolo et al. observed that most method names tend to consist of several words but rarely more than five words [26].

Going beyond the discussion of whether variable names should be shorter or longer, Arnaoudova et al. [27], [28] have studied linguistic anti-patterns, “recurring poor practices in the naming, documentation, and choice of identifiers in the implementation of an entity” such as discrepancies between the behavior implied by the identifier (e.g., `INCLUDE_NAME_DEFAULT` in the Cocoon Apache project) and the corresponding comment (“Configuration default exclude pattern”), or between the identifier and commonly applied guidelines (e.g., getter that does not only access fields). In the educational setting Glassman et al. propose Foobaz, a tool giving semi-automatic feedback on student variable names based on the values the variable can take during the execution and limited input from the teacher [29].

While visual languages such as Scratch, Squeak and Alice recently become a favorable choice for elementary and high schools as an introduction to programming [17], the lion’s share of the previous work on identifier naming has focused on textual languages. Notable exception is the recent work of Moreno and Robles [30]: Moreno and Robles have observed that students often keep names automatically generated by the Scratch environment rather than changing them. van Zyl et al. have observed that one of the interviewed school teachers working with Scratch has taught the students to integrate variable types in their names, e.g., ‘S’ for Strings [12].

## III. RELEVANT SCRATCH CONCEPTS

We briefly introduce several core features of Scratch required for understanding the remainder of the paper. Interested readers are referred to [31] for an extensive overview.

Scratch is a block-based programming language aimed at children, developed by MIT. Scratch can be used to create

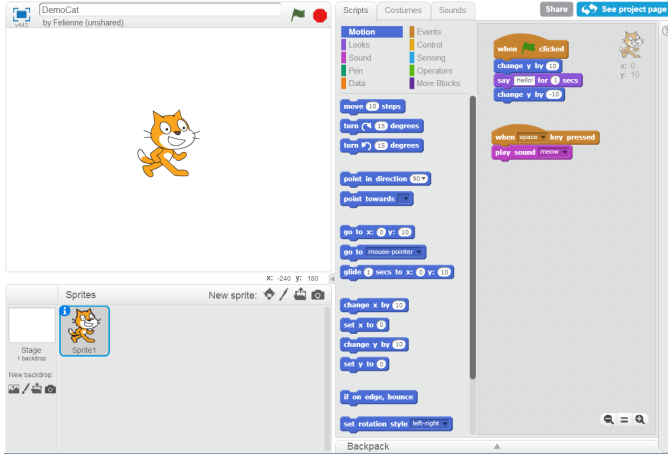


Fig. 2: The Scratch user interface consisting of the ‘cat’ sprite on the left, the toolbox with available blocks in the category ‘motion’ in the middle and the code associated with the sprite on the right.

games and interactive animations, and is available both as a stand-alone application and as a web application. Fig. 2 shows the Scratch user interface in the Chrome browser.

1) *Sprites*: Scratch code is organized into ‘sprites’: two-dimensional pictures that each have their own source 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 Fig. 2 consists of one sprite, the cat, which is Scratch’s default sprite and logo<sup>4</sup>. 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.

2) *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 Fig. 2 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) *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 Fig. 3. This figure also shows that, contrary to most programming languages, variable names in Scratch may contain spaces.

4) *Procedures*: Scratch also allows users to create their own blocks, called procedures. They can have input parameters, and labels in between them. Procedures are created with an interface similar to the one to create variables. Fig. 4 shows the definition and invocation of a procedure.

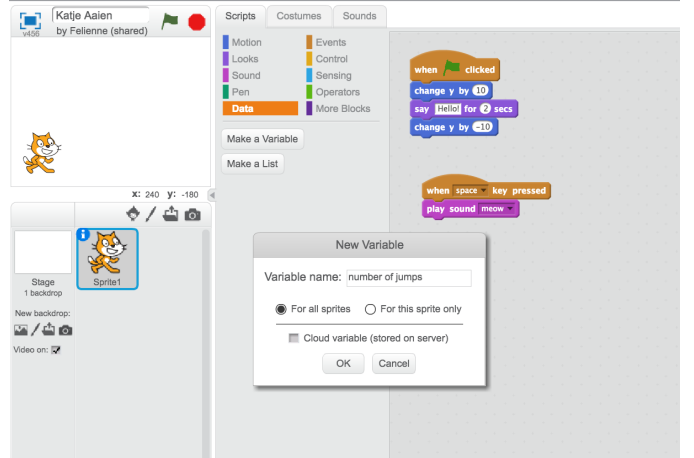


Fig. 3: The Scratch user interface to create a variable

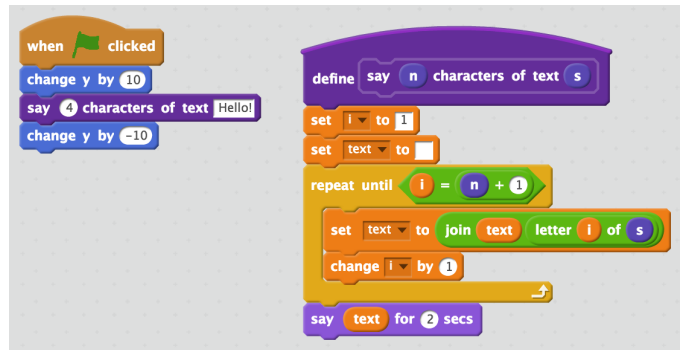


Fig. 4: Scratch code to define and invoke a procedure

## 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 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. [32] we perform a dependent replication of the studies summarized in Fig. 1 and 2 of the original work [3]. Inherently, the programming language is the only factor we vary when compared to the original study. However, Scratch programs are not available on GitHub, so we use the dataset previously scraped and processed by Aivaloglou and Hermans [18]. We report on the results of these replications in Sections V-A1 and V-A2.

Similarly, we perform another dependent replication of the same studies by considering procedure names rather than variable names in Scratch (Section V-A3).

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 to understand the type-related user perceptions (“what type would you consider for a variable called ...?”) we focus on the types as used in

<sup>4</sup><https://Scratch.mit.edu/projects/97086781/>



Fig. 5: Two variables, one of type string and one of type integer

the program. We opt for types as being used as opposed to as being perceived due to limited programming experience of the intended Scratch programmers. Indeed, Scratch is meant for individuals in their first steps of learning how to program, such as school-age students and we do not expect them to have established perceptions on data types of single-letters variables. As opposed to our work, in the original study however, 30% of survey respondents claim a 10-years experience in programming, while 23% have programming knowledge in six different languages or more [3]. Furthermore, we study types as used as opposed to types as defined, since Scratch does not have a concept of an explicit variable type. However, we can deduce the variable types from assignments involving those variables. For example, the two variables in Fig. 5 represent a string and an integer respectively. Results of this conceptual replication are presented in Section V-A4.

Finally, in Section V-B we report on the ways Scratch developers employ Scratch-specific naming practices such as spaces in variable names, numeric values as variables and the use of textual labels in between parameters.

#### B. Dataset

For this paper we use the dataset created by Aivaloglou and Hermans [18], consisting of 250000 Scratch projects scraped from the Scratch website in March 2016. From this dataset, we have selected the projects that use variables or procedures, tantamount to 73473 projects of 29% of the Aivaloglou and Hermans’s original dataset. Variables are used more often than procedures: 69045 projects (27.6%) use variables, while 17605 use procedures (7%). 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

Understanding differences in variable name lengths occurring between different programming languages requires comparison of multiple distributions. Such a comparison is 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 [33].

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 [34]. Therefore, one-step approaches have been sought. We opt for one such approach, the  $\tilde{T}$ -procedure of Konietzschke et al. [35], [36]. This procedure is robust against unequal population variances, respects transitivity, and has been successfully applied in empirical software engineering [37], [38], [39]. In particular, we use the Tukey (all-pairs) contrasts to compare all distributions pairwise.

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’. To eliminate sensitivity in the ways the distance matrix is computed we first calculate the cosine similarity between vectors and compare the mean similarity of a language to the remaining languages in the dataset, and then perform hierarchical clustering based on the Euclidean distance.

When comparing distributions of variable name lengths with the procedure name lengths, the  $\tilde{T}$ -procedure is not applicable. Hence, we perform the Mann-Whitney-Wilcoxon test together with the two-sample test for the nonparametric Behrens-Fisher problem, i.e., test for  $H_0 : p = 1/2$ , where  $p$  denotes the relative effect of the two independent samples [40], [36].

### V. RESULTS

This section presents an overview of our analysis of variable and procedure name use in the previously published Scratch dataset [18].

#### A. Replication studies

We start by reporting on the results of the replication studies.

1) *Variable name length*: The original study of Beniamini et al. [3] 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 9–20% of the names.” Fig. 6 shows the distribution of lengths in the Scratch corpus. A closer look at the data reveals that the single-letter variables constitute ca. 4.9% of all the variable names, i.e., less than the 9–20% observed by Beniamini et al.

Compared to mainstream languages, single-letter variables seem to be less common in Scratch, and the maximum length value for the variable names (250 characters) is the largest by a significant margin. These observations lead us to wonder whether overall the variable names in Scratch tend to be longer than in other programming languages. To this end we apply the  $\tilde{T}$ -procedure described in Section IV-C. Statistical analysis reveals that indeed, variable names in Scratch tend to be longer than in the mainstream languages. Moreover, variable names in Java tend to be longer than those in PHP, variable names in PHP than those in C, variable names in C than those in JavaScript and finally variable names in JavaScript longer than those in Perl. In all cases  $p$ -values have been too small to be computed precisely.

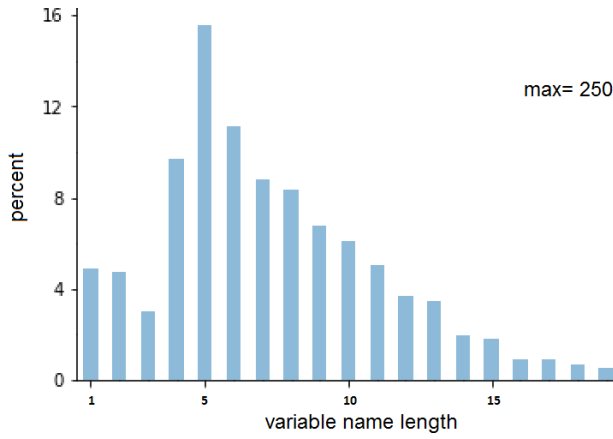


Fig. 6: The distribution of variable's name length in Scratch projects

We conclude that single-letter variable names are less common in Scratch than in other programming languages and that overall Scratch variables have longer names than variables in other programming languages.

2) *Single-letter variable names*: Further we investigate the case of single-letter variable names. For the previously studied programming languages, the authors in [3] highlight the following observations about the single-letter usage:

- The most commonly occurring single-letter variable name is *i*. The authors attribute this to *i* being commonly used as a loop counter. As opposed to the studied mainstream languages, loops are performed in Scratch using predefined blocks. As illustrated on Fig. 7 the majority of these blocks do not require a variable to control the loop iterations. Hence, we expect the usage of the variable name *i* to be less common in Scratch.
- Apart from the popularity of *i* the distribution is language-dependent. Since Scratch is quite different from the mainstream programming languages considered by Beniamini et al., we expect the distribution of the single-letter variable names to be different from the distributions in these languages. Hence, we expect the *similarity* between Scratch and the languages considered by Beniamini et al. to be lower than the *similarity* between the languages considered by Beniamini et al.
- Finally, they observed that the lower case letters are used more frequently than the upper case letters. Since this is also the case for regular text in most natural languages as well, we expect the Scratch programs to follow the same pattern.

Fig. 8 shows the distribution of variables of one letter, in upper and lower case, in the Scratch corpus. Inspecting the data we observe that similarly to the previous study *i* is the most commonly occurring variable. Hence, we conclude that *contradicting our expectations* Observation a) above also holds for Scratch. Furthermore, we observe that *x* and *y* are extremely popular in Scratch. This can be explained by noting



Fig. 7: Scratch blocks that are used to repeat specific actions

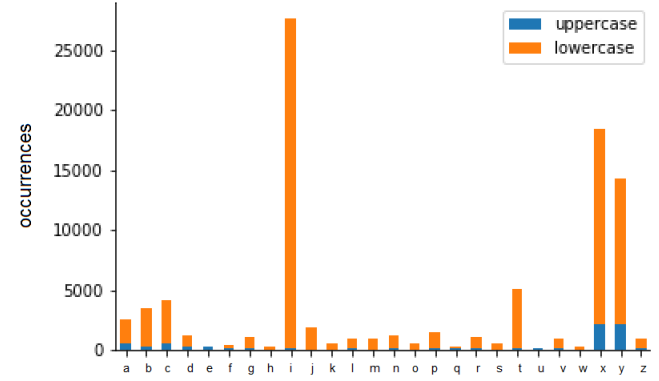


Fig. 8: A histogram of single-letter variables occurrences in Scratch projects

that *x* and *y* represent the coordinates of the sprite object on the stage, which is the area where all the scripts are executed. Hence, they are the basis of moving the object in the 2-D stage. In fact, Scratch default blocks often use *x* and *y* as shown in Fig. 9. It seems that Scratch users are inspired by the Scratch language to name their own variables.

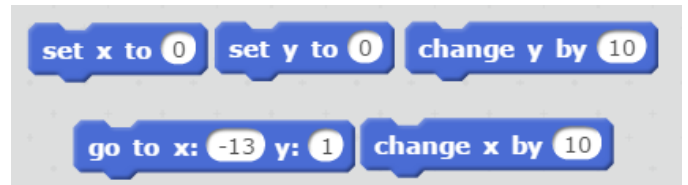


Fig. 9: Some Scratch blocks that use x and y characters

Next we study the similarity of Scratch to the mainstream programming languages in terms of frequency distribution of single-letter variable names. Hence, we compare the mean cosine similarity of the twelve distributions (five mainstream programming languages plus Scratch, considered for the uppercase and the lowercase letters). The mean cosine similarity shows that the Scratch usage of the uppercase letters in the single-letter variable names is the most dissimilar among the twelve distributions (0.39), while the way lowercase letters are used in Scratch quite similarly to the way letters are used in the mainstream programming languages (0.62). Hence, we claim that our expectation based on the Observation b) has been confirmed for the uppercase letters and rejected for the lowercase letters. Closer look at Fig. 10 shows that the usage



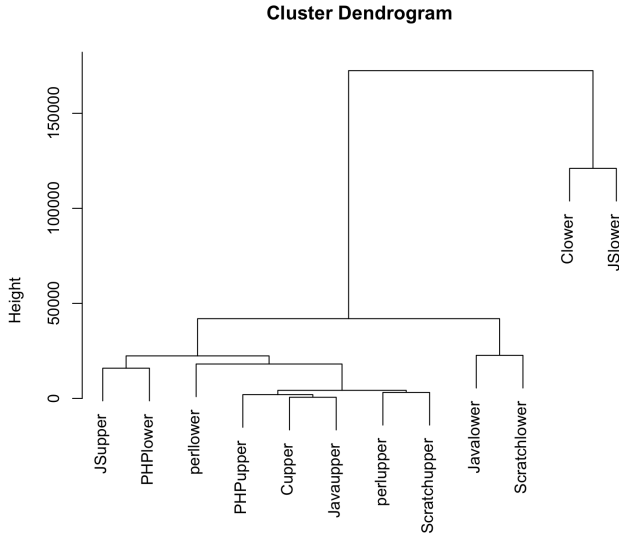


Fig. 10: A cluster dendrogram of Scratch compared to other programming languages for the single-letter pattern

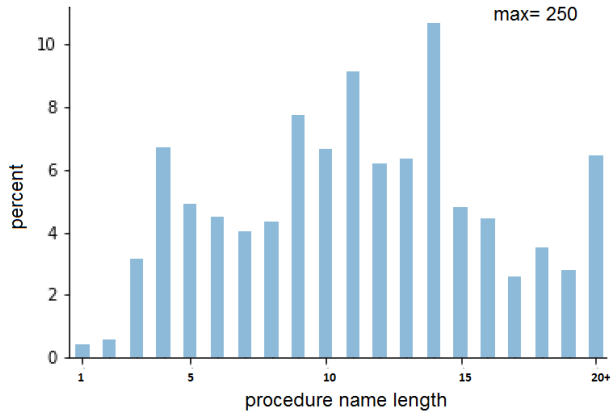


Fig. 11: The distribution of procedure's name length in Scratch projects

of single uppercase letters is similar to the pattern found in Perl, for the lowercase—to the pattern found in Java.

Finally, Fig. 8 clearly shows that the lowercase letters are much more often used as variable names than the uppercase letters, providing support for Observation c).

3) *Procedure names*: Going beyond the study of Beniamini, we additionally consider the naming of procedures in Scratch. For a detailed explanation of procedures in Scratch see Section III-4 and Fig. 4).

Figure 11 shows the distribution of procedures name length in the Scratch dataset. By inspecting this figure we observe that the procedure names tend to be longer compared to Scratch variable names. Indeed, the two-sample test for the nonparametric Behrens-Fisher problem estimates the relative

effect of the two samples (procedure name lengths vs. variable name lengths) as 0.149 (with the  $p$ -value being too small to be computed precisely), i.e., it indicates that the procedure names' lengths tends to be larger than the lengths of the variable names. This observation is confirmed by the Mann-Whitney-Wilcoxon test (the  $p$ -value is too small to be computed precisely). Short names are not common, even less common than short names of variables: single-letter names compose less than 1% of the extracted names, less than 4.9% observed for Scratch variables and 9–20% in C, Java and Perl [3]. The maximum length for a procedure name is 250 characters, which is the same as the maximum length for the variable names. We suspect this exact match is caused by a language constraint that was imposed in a previous versions of Scratch. Current version of Scratch, however, allows for names longer than 250 characters.

Similarly to the variable names, we consider single-letter names for the procedures and revisit Observations a) and c). We could not revisit Observation b) since data on single-letter procedure names in the mainstream languages is not available. Fig. 13 shows the number of occurrences for each alphabetic letter. We see that  $i$  is no longer among the most commonly used letters rejecting Observation a). The top used single-letter name is  $a$ , the first letter in the alphabet, which might explain its popularity. The uppercase letters are used more often than the lowercase letters (in 267 vs. 218 projects), rejecting Observations c). Sometimes the uppercase letter's frequency exceeds the one of the lowercase, e.g., in the case of  $r$ .

4) *Types*: In the original paper of Beniamini et al. [3] the authors observe that some letters are highly associated by developers with the type if its name which starts with these letters: e.g.,  $char$  for  $c$  and  $string$  for  $s$ . Integer data type is a common association for many other letters. Letter names such as  $x$ ,  $y$  and  $z$  are commonly interpreted as coordinates and there seems to be a balance between integer and float associations. As explained in Section IV-A we conduct a conceptual replication by considering types as used as opposed

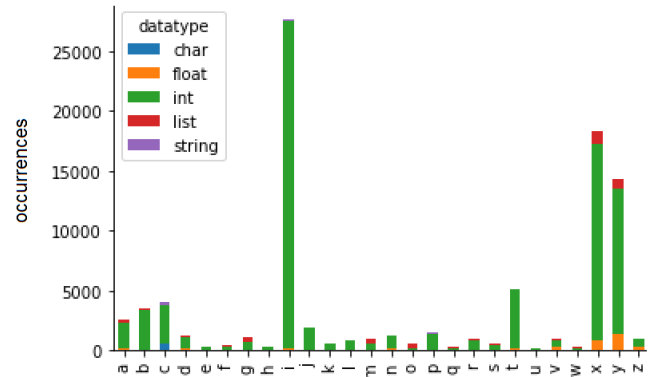


Fig. 12: Inferred types for variables of one letter

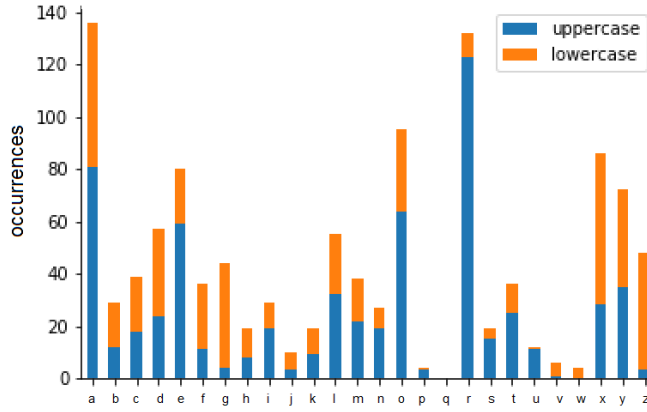


Fig. 13: A histogram of single-letter procedures occurrences in Scratch projects

to types as perceived.

Fig. 12 shows the distribution of single-letter variables with the types inferred. Similarly to the observations of Beniamini et al. every letter can represent an integer variable. As opposed to the observations of Beniamini et al. we see that the types associated with individual letters are much less diverse in Scratch: the integer data type is dominating every single letter. Another disagreement with the original study pertains to the string data type: while in the mainstream programming languages strings are commonly associated with *s* and less frequently with many other letters, the strings are almost completely absent in the Scratch data set. While we can confirm the observation of Beniamini et al. that *c* is used to represent characters, this is not the main usage of *c* and no such association could be observed for floats, lists or strings. Finally, presence of floats as types for *x*, *y* and to lesser extent *z* seems to support their interpretation as coordinates suggested by Beniamini et al. [3].

### B. Scratch-specific constructs

In this section we analyze the occurrence of naming practices that are allowed in Scratch, but are missing from or are not common in most mainstream textual languages.

1) *Use of spaces in variable names:* For reasons of convenience, most textual programming languages do not allow for spaces in variable names. FORTRAN ignored spaces, so technically one could use a space, however that would mean that ‘apples’ and ‘app les’ refer to the same variable. Other languages supporting spaces in variable names are SQL and some Scheme implementations. Even languages targeting data analysts rather than software developers recommend spaces to be avoided [41].

We observe that about 30000 projects use one or more variables with a space in it, as opposed to 60000 that use only space-free variable names. Fig. 15 shows that mostly one space is present in a variable name.

2) *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.

The most popular numeric values used as variable names include small natural numbers (1–12) and 360 likely to represent 360° (cf. Fig. 14).

There seem to be two main uses of numeric variable names. First of all, some variables with numeric names represent constants, as shown in Fig. 16. This seems to indicate the Scratch users prefer to drag in a constant rather than repeatedly typing it.

A second use is the use of integer variables as simple list structures. For example, one of the projects we analyzed is a tic-tac-toe game (See Fig. 18). In that project, the Scratch user defined nine variables named from 1 to 9. Each variable represents one of the nine boxes. Scratch supports lists, so the user here could have also used a list of 9 items, however, they did not. Maybe because they were not familiar with the concept of lists, or maybe they thought this would be easier for the user to memorize the game logic.

3) *Use of Textual Labels between Parameters:* Scratch is influenced by the SmallTalk family of languages and this is visible from the fact that Scratch allows users to insert textual labels in between parameters in order to make procedures more readable, as can be seen in Fig. 4. This practice seems particular idiomatic to Scratch, since normal Scratch blocks use a similar syntax, e.g., in the “say ... for ... seconds” block.

In total 4415 projects use textual labels, so their use is relatively uncommon. We do however find some interesting patterns. Fig. 17 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 ‘:’ at the end of many patterns, which could come from the users being inspired by Scratch

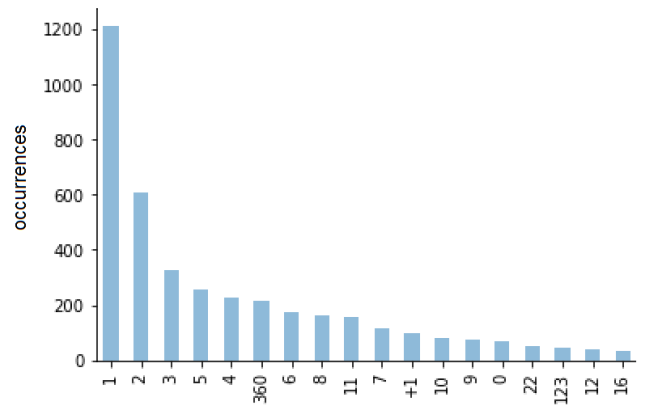


Fig. 14: The most popular numeric values used as variable names

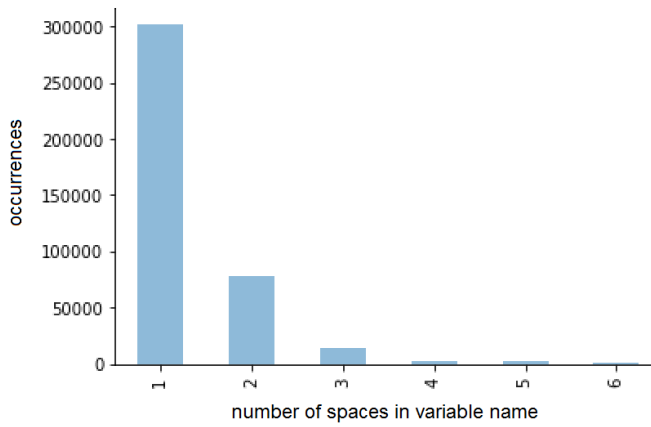


Fig. 15: Number of spaces in variable names

default blocks, which use the colon as shown in Fig. 9. Finally the use of the space (char-space in Fig. 17) 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.

## VI. DISCUSSION

### A. Threats to validity

**Representation of the sample; Statistical analysis chosen; Type inference** As any empirical study our work is subject to series of threats to validity. Construct validity of our study might be threatened by the operationalization of the notion of a type. Due to lack of explicit type declarations in Scratch... **to be continued**

### B. Naming in Scratch

We only considered features of Scratch that enforce the user to input textual values. For example the name of the sprites can also be considered as an identifier. However, Scratch assigns

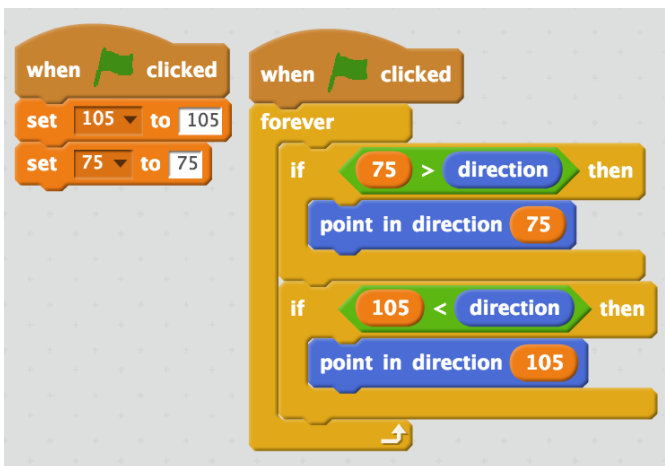


Fig. 16: Numeric variable used as a constant

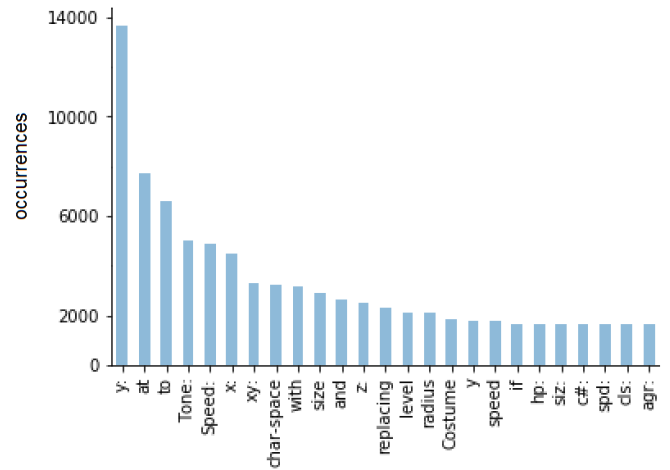


Fig. 17: The most used textual labels in between parameters of procedures

default names to sprites once they are created, and the user may opt not to change it. Default names can be considered as a bad naming practice, but it is out of the scope of this paper. **other ideas for the discussion: for example: what effects would the outcome of this analysis have on Scratch-related material or maybe we skip the discussion?**

## VII. CONCLUSION

In this paper, we study naming patterns for variables and functions in the Scratch programming language, a block-based programming language aimed at novice programmers. We use a previously released dataset consisting of 250.000 Scratch programs.

Our analysis shows that Scratch users most often use variable and procedure names between 4 and 10 characters in length. For the single letter variables, the most commonly used names are x, y and i. Spaces in variable names, a feature relatively unique to Scratch are used in 20% of projects have variables that include spaces in the name. The usage of textual string between parameters appears as not so common, however textual patterns used imply an inference from textual languages by using brackets for example. Finally, when compared to the other programming languages, Scratch variable length distribution, and the usage of single-letter seems to be most similar to Perl.

The paper makes the following contributions:

- A detailed analysis of one letter variable names, replicating [3] on the Scratch programming language
- An analysis of function names in Scratch
- An analysis of naming patterns unique to Scratch, including spaces in variable names, textual labels in procedures and numeric variable names

This paper gives rise to a number of directions for future work. Firstly, Beniamini et al. [3] included a survey in which



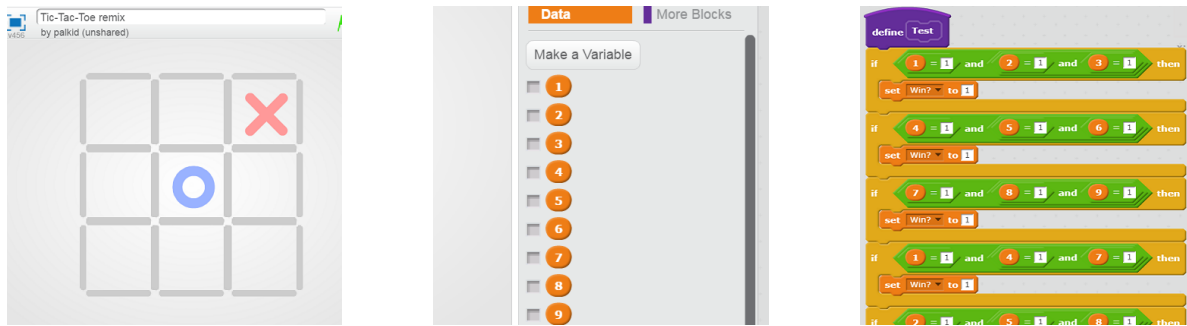


Fig. 18: A Tic-Tac-Toe program in Scratch uses numeric variables to represent the boxes to be checked by the player

they ask developers to predict the type of a (one letter) variable. It could be interesting to ask a similar question of children to for common variable names. Furthermore, a detailed study into the readability of variable names with and without spaces, and procedures with and without labels would help us to create naming guidelines for Scratch.

## 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 *ESEM*. IEEE, 2015, pp. 50–53.
- [2] E. Avidan and D. G. Feitelson, "Effects of variable names on comprehension an empirical study," in *ICPC*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 55–65.
- [3] G. Beniamini, S. Gingichashvili, A. Klein-Orbach, and D. G. Feitelson, "Meaningful identifier names: the case of single-letter variables," in *ICPC*, G. Scanniello, D. Lo, and A. Serebrenik, Eds. IEEE / ACM, 2017, pp. 45–54.
- [4] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the influence of identifier names on code quality: An empirical study," in *CSMR*, R. Capilla, R. Ferenc, and J. C. Dueñas, Eds. IEEE, 2010.
- [5] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *SANER*, M. Pinzger, G. Bavota, and A. Marcus, Eds. IEEE, 2017, pp. 217–227.
- [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 *Workshop on User Evaluations for Software Engineering Researchers*. IEEE, 2013, pp. 13–15.
- [7] G. Scanniello and M. Risi, "Dealing with faults in source code: Abbreviated vs. full-word identifier names," in *ICSM*. IEEE, 2013, pp. 190–199.
- [8] P. Tramontana, M. Risi, and G. Scanniello, "Studying abbreviated vs. full-word identifier names when dealing with faults: an external replication," in *ESEM*, M. Morisio, T. Dybå, and M. Torchiano, Eds. ACM, 2014, p. 64:1.
- [9] T. Kato, Y. Kambayashi, and Y. Kodama, *Data Mining of Students' Behaviors in Programming Exercises*. Cham: Springer, 2016, pp. 121–133.
- [10] K. Rother, *Cleaning Up Code*. Springer, 2017, pp. 195–212.
- [11] E. S. Raymond, *The New Hacker's Dictionary*. MIT Press, 1996. [Online]. Available: [https://books.google.be/books?id=g80P\\\_4v4QbIC](https://books.google.be/books?id=g80P\_4v4QbIC)
- [12] S. van Zyl, E. Mentz, and M. Havenga, "Lessons learned from teaching scratch as an introduction to object-oriented programming in delphi," *African Journal of Research in Mathematics, Science and Technology Education*, vol. 20, no. 2, pp. 131–141, 2016.
- [13] 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.
- [14] B. M. Hill and A. Monroy-Hernández, "The remixing dilemma," *American Behavioral Scientist*, vol. 57, no. 5, pp. 643–663, 2013.
- [15] R. Davis, Y. Kafai, V. Vasudevan, and E. Lee, "The education arcade: Crafting, remixing, and playing with controllers for scratch games," in *International Conference on Interaction Design and Children*. ACM, 2013, pp. 439–442.
- [16] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper, "Mediated transfer: Alice 3 to Java," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 141–146. [Online]. Available: <http://doi.acm.org/10.1145/2157136.2157180>
- [17] Y. Matsuzawa, T. Ohata, M. Sugiura, and S. Sakai, "Language migration in non-cs introductory programming through mutual language translation environment," in *SIGCSE*. ACM, 2015, pp. 185–190.
- [18] E. Aivaloglou and F. Hermans, "How kids code and how we know: An exploratory study on the scratch repository," in *ICER*, 2016.
- [19] N. Anquetil and T. C. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *CASCON*, S. A. MacKay and J. H. Johnson, Eds. IBM, 1998, p. 4.
- [20] B. Caprile and P. Tonella, "Restructuring program identifier names," in *ICSM*. IEEE, 2000, pp. 97–107.
- [21] 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.
- [22] 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.
- [23] B. E. Teasley, "The effects of naming style and expertise on program comprehension," *International Journal of Human-Computer Studies*, vol. 40, no. 5, pp. 757–770, 1994. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S1071581984710366>
- [24] F. Deißeböck and M. Pizka, "Concise and consistent naming [software system identifier naming]," in *IWPC*, May 2005, pp. 97–106.
- [25] B. Caprile and P. Tonella, "Nomen est omen: analyzing the language of function identifiers," in *WCRE*, Oct 1999, pp. 112–122.
- [26] A. Caracciolo, A. Chis, B. Spasojević, and M. Lungu, "Pangea: A workbook for statically analyzing multi-language software corpora," in *SCAM*, Sept 2014, pp. 71–76.
- [27] V. Arnaoudova, M. Di Penta, G. Antoniol, and Y. Guéhéneuc, "A new family of software anti-patterns: Linguistic anti-patterns," in *CSMR*, A. Cleve, F. Ricca, and M. Cerioli, Eds. IEEE, 2013, pp. 187–196.
- [28] V. Arnaoudova, M. Di Penta, and G. Antoniol, "Linguistic antipatterns: what they are and how developers perceive them," *Empirical Software Engineering*, vol. 21, no. 1, pp. 104–158, 2016.
- [29] E. L. Glassman, L. Fischer, J. Scott, and R. C. Miller, "Foobaz: Variable name feedback for student code at scale," in *Annual ACM Symposium on User Interface Software & Technology*. ACM, 2015, pp. 609–617.
- [30] J. Moreno and G. Robles, "Automatic detection of bad programming habits in scratch: A preliminary study," in *Frontiers in Education Conference*, Oct 2014, pp. 1–4.
- [31] K. Brennan, C. Balch, and M. Chung, *CREATIVE COMPUTING*. Harvard Graduate School of Education, 2014.
- [32] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, "The role of replications in empirical software engineering," *Empirical Software Engineering*, vol. 13, no. 2, pp. 211–218, 2008.
- [33] K. R. Gabriel, "Simultaneous test procedures—some theory of multiple

comparisons,” *The Annals Mathematical Statistics*, vol. 40, no. 1, pp. 224–250, 1969.

- [34] 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.
- [35] F. Konietzschke, 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.
- [36] F. Konietzschke, M. Placzek, F. Schaarschmidt, and L. Hothorn, “npar-comp: An r software package for nonparametric multiple comparisons and simultaneous confidence intervals,” *Journal of Statistical Software*, vol. 64, no. 9, pp. 1–17, 2015.
- [37] 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.
- [38] 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.
- [39] 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.
- [40] K. Neubert and E. Brunner, “A studentized permutation test for the non-parametric Behrens-Fisher problem,” *Computational Statistics & Data Analysis*, vol. 51, no. 10, pp. 5192–5204, 2007.
- [41] M. Bochud, *Estimating Heritability from Nuclear Family and Pedigree Data*. Humana Press, 2012, pp. 171–186.
- [42] G. Scanniello, D. Lo, and A. Serebrenik, Eds., *ICPC*. IEEE / ACM, 2017.