How do Scratch Programmers Name Variables and Procedures?

Alaaeddin Swidan
Delft University of Technology
The Netherlands
Email: alaaeddin.swidan@tudelft.nl

Alexander Serebrenik
Eindhoven University of Technology
The Netherlands
Email: a.serebrenik@tue.nl

Felienne Hermans
Delft University of Technology
The Netherlands
Email: f.f.j.hermans@tudelft.nl

***Abstract*—Research shows the importance of selecting good names to identifiers in software code: more meaningful names improve readability. In particular, several guidelines encourage long and descriptive variable names. A recent study analyzed the use of variable names in five programming languages, focusing 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 analyze variables in Scratch, a popular block-based language aimed at children. We start by replicating the above single-letter study for Scratch. We augment this study by analyzing single-letter procedure names, and by investigating the use of Scratch specific naming patterns: spaces in variable names, numerics as variables and textual labels in procedure names.

The results of our analysis show that Scratch programmers often prefer longer identifier names than developers in other languages, while Scratch procedure names have even longer names than Scratch variables. For the single-letter variables, the most frequent names are x, y, and i. Single-letter procedures are less popular, but show more tendency to be in upper case. When compared to the other programming languages, the usage of single uppercase letters in Scratch variables seems to be similar to the pattern found in Perl, while for the lowercase letters—to the pattern found in Java. Concerning Scratch specific features, 44% of the unique variable names and 34% of the projects in the dataset include at least one space. The usage of textual labels between parameters in procedure names appears as not common, however textual patterns used imply an influence from textual languages, for example by using brackets.

Alaaeddin: modified based on rev.2: Previous research indicate the identifier names as one significant issue in transitioning from visual block-based to textual programming languages. The naming patterns we found support this claim for Scratch programmers who may incur difficulties when transitioning to the use of mainstream textual programming languages. Those languages restrict the use of spaces in identifiers and more often divert into short and single-letter names—tendencies opposite to the naming preferences in Scratch.

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, many computer science and programming curricula 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], [10] and practitioners [11] 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, as discussed by Raymond [12] standard metasyntactic variables used in syntax examples are *"foo"* and *"bar"*. The names of these identifiers are meaningless, and to some extent, they represent a refusal to name, suggesting to the learner that naming is less important to the programming task.

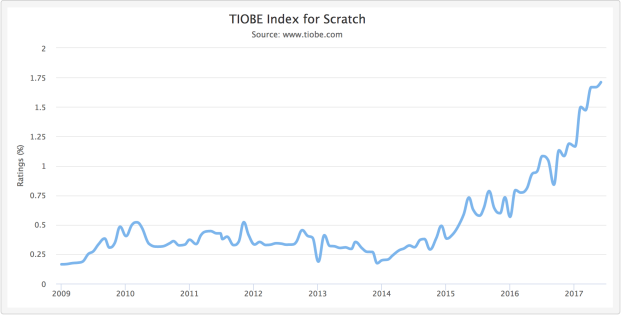

Fig. 1: TIOBE Programming Community index: evolution of the popularity of Scratch

The goal of this paper is to **understand the patterns in variable and procedure naming 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 [13], [14]. Moreover, the overall popularity of Scratch is witnessed by Scratch being currently rated 19 in the TIOBE index[3], topping such languages as Lua, Scala and Groovy and since early 2014 exhibiting an increasing trend shown in Fig. 1.

Scratch programming materials too do not focus on naming. For example, the Creative Computing Learner Workbook

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

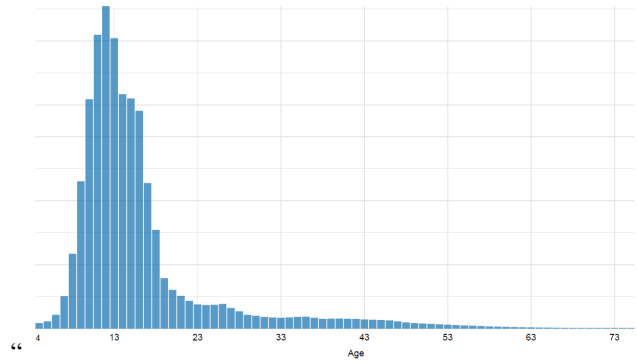

Fig. 2: Age distribution of Scratch users at the time of registration according to Scratch statistics web-page

created by the ScratchEd group at Harvard does not explain how to select good names for procedures and variables [4]. It is therefore interesting to explore the naming habits of Scratch programmers. And there are other 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' [15], [16], a code sharing practice 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. As shown in Fig. 2, 51% of Scratch programmers are aged between 10 and 15 at the time of registration according to the statistics provided by Scratch[5]. Therefore, researchers of 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 [17], [18].

We start by a **general discussion of naming practices** in Scratch and analyze the previously published collection of 250,000 Scratch projects [19]. 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. who 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 procedure names as well.

> Variable names in Scratch range mostly between 4 and 10 characters. While procedure names tend to be longer. For the single-letter variables, the most commonly used names are x, y and i, procedures—a, y 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**. Scratch supports a number of less commonly used naming features, for example spaces may be used in names e.g., a variable *max i* and integers and even floating point numbers can be used as variable names e.g., a variable named *6* or *3.14159*. Finally, Scratch allows for textual labels in between parameters. For example a method for printing the first n letters from a string s could be called "printnof(n,s)" in a textual language. Scratch allows for this too, however, one can also define a procedure called "say n characters from text s", as shown in Fig. 5. This feature does exist in textual languages too—most notable in SmallTalk—but is not commonly found 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 programmers, this might be because they ease novice programming, and that means one could even advocate that these features should be integrated in the textual programming languages, if only to ease the transition for the block-based languages programmers.

> Spaces in variable names are common: 34% of projects use this feature. Numeric values as identifiers are rarely used, and mostly represent constants or parts of the data structure. The usage of textual string between parameters is not as 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], [20], [2], [3], [4], [21], [5], [22], [6], [7], [23], [24], [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 [25].

For a human reading code, it is crucial to the understanding of code to also understand what the identifiers mean. As such, several studies have investigated the link between identifier naming and code readability and comprehension [2], [5], [22], [23], [24] 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 [26] have observed that names of C functions consist of on average of 2.04–3.36 words, often verbs expressing action, while Caracciolo et al. observed that most method names in Java and Smalltalk also tend to consist of several words but rarely more than five words [27].

[4]http://scratched.gse.harvard.edu/guide/files/CreativeComputing20140820_LearnerWorkbook.pdf

[5]https://Scratch.mit.edu/statistics/

Going beyond the discussion of whether variable names should be shorter or longer, Arnaoudova et al. [28], [29] 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 [30]. While visual languages such as Scratch, Sqeak, and Alice have recently become a favorable choice for schools as an introduction to programming [18], 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 [31]: they observed that Scratch programmers often do not change the sprite names that were automatically generated by the environment. 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 [13]. Finally, Hermans and Aivaloglou encouraged the students in Scratch MOOC to choose meaningful names and to avoid keeping default ones [10]. **Alaaeddin:I added this to link with abstract and conclusion remarks.** The work on naming patterns and preferences for students learning how to program with visual languages is important to understand, and then act as a community, the difficulties faced by students when transitioning to textual programming languages. According to Klling et. al. [43], for these learners, dealing with user-defined identifiers is one of the *"fundamental challenges"*. It involves an extra cognitive load to remember identifiers, with case sensitivity in some cases, instead of selecting a variable from a drop-down list in Scratch. It also relates to the broader challenges of spelling and writing for these students.

III. Relevant Scratch Concepts

We briefly introduce several core features of Scratch required for understanding the reminder of the paper. Interested readers are referred to the book *"Creative Computing"* [32] for an extensive overview.

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 3 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.

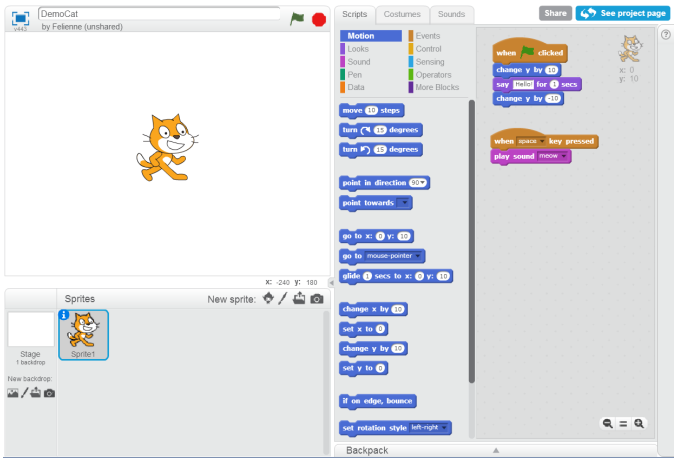

Fig. 3: The Scratch UI consisting of the 'cat' sprite on the left, the toolbox with available blocks in the category 'motion' in the middle and the code associated with the sprite on the right.

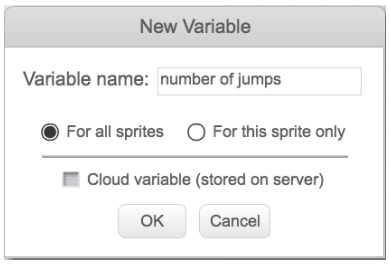

Fig. 4: The Scratch user interface to create a variable

The Scratch program in Fig. 3 consists of one sprite, the cat, which is Scratch's default sprite and logo[6]. 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. 3 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 programmers can use variables. Variables are untyped, but have to be 'declared' through the Scratch user interface, shown in Fig. 4. This figure also shows that, contrary to most programming languages, variable names in Scratch may contain spaces.

4) Procedures: Scratch allows programmers to create their own blocks, called procedures. They can have input parameters, and labels in between. Procedures are created with an interface similar to the one to create variables. Figure 5 shows the definition and the invocation of a procedure in Scratch.

[6]https://Scratch.mit.edu/projects/97086781/

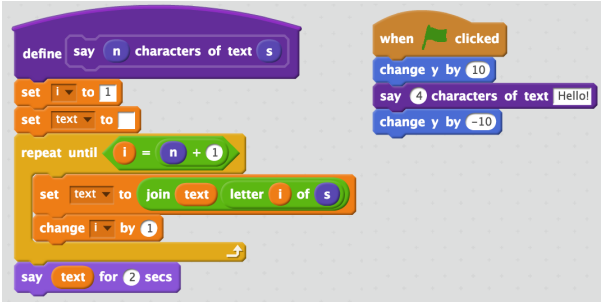

Fig. 5: Scratch code to define and invoke a procedure

IV. Research design and dataset

A. Overall design

The goal of this paper is to compare naming practices of the Scratch programmers to those of the developers in mainstream programming languages. To that end, we start by partially replicating the recent work of Beniamini et al. on single-letter variables in Java, C, PHP, Perl, and JavaScript [3]. In terms of the classification of Shull et al. [33], we perform a dependent replication of the studies summarized in Fig. 1 and 2 of the original work. Inherently, the programming language is the only factor we vary when compared to the original study. However, as opposed to the data in the original study, Scratch programs are not available on GitHub. Hence, we use the dataset previously scraped and processed by Aivaloglou and Hermans [19]. We report on the results of these replications in Sections V-A1 and V-A2. Furthermore, we perform another dependent replication of the same studies by considering Scratch procedures rather than variables (Section V-A3).

Next, we perform a conceptual replication of the study of the single-letter variable types of Beniamini et al. [3]. The original study conducted a survey to understand the type-related user perceptions, with questions such as *"what type would you consider for a variable called ...?"*. We however focus on the types as used in the program. We investigate types *"as-being-used"*, as opposed to *"as-being-perceived"* in the original study due to the limited programming experience of the intended Scratch programmers. Scratch is meant for people taking their first steps in programming, 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 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. 6 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.

Fig. 6: Two variables, one of type string and one of type integer

B. Dataset

For this paper we use the dataset created by Aivaloglou and Hermans [19], 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 procedures, which amount to 73,473 projects, 29% of the Aivaloglou and Hermans's original dataset. Variables are used more often than procedures: 69,045 projects (27.6%) use variables, while 17,605 use procedures (7%). We used Python to process the original dataset and generate the graphs in this paper.

C. Identifier Extraction

To follow the steps of the replicated study, we collect the unique variable names used in the projects' scripts. Within the scripts we identify Scratch blocks that are used to assign a value to the variables, : e.g., "Change *variable* by *value*" or "Set *variable* to *value*". Variable names in Scratch are unique: once a variable is declared in a project, its name cannot be used to create another variable in the same project, even in a different scope, e.g., in a different sprite.

To determine the type of single-letter variables, we perform type inference on the parameter value assigned to each variable. The inference is performed via standard data type conversion of the value. If the variable is accessed multiple times in a project with different data types, for example first as a string and then as an integer, both data types are counted.

For the procedure names, we follow a similar approach to the extraction of variable names. We extract the blocks used to call a particular procedure and then extract the name of the procedure, and count the number of projects in which a procedure name occurs.

We note here that Scratch allows the user to create multiple procedures with the same exact name in the same sprite. It is not clear to us why the language would support such a feature. We argue, however, that counting the procedure names once per project is an indication of the naming patterns used and fits the needs of this study. For the presence of spaces in and numbers as variables we simply analyze the previously extracted variable names, while textual patterns in procedure names are detected from the extracted procedures' names. We provide the Python code, input and output files for verification and replication purposes on a GitHub repository[7].

[7]https://github.com/Felienne/ScratchVars

D. 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 [34]. 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 [35]. Therefore, one-step approaches have been sought. We opt for one such approach, the $\widetilde{\mathbf{T}}$-procedure of Konietschke et al. [36], [37]. This procedure is robust against unequal population variances, respects transitivity, and has been successfully applied in empirical software engineering [38], [39], [40]. 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 $\widetilde{\mathbf{T}}$-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 [41], [37].

For spaces use in variables' names we do two things: (i) To understand how much popular spaces are among all the names, we extract the space count per unique variable name. (ii) To understand the trend of using spaces across the dataset (multiple projects and multiple users), we extract per project the maximum space count found in the project's variables. For the textual patterns in procedure names, we count the occurrence of each of the extracted token.

V. Results

This section presents an overview of our analysis of variable and procedure name used in the previously published Scratch dataset [19]. We start by replicating studies of Beniamini et al. [3] and proceed with investigating Scratch-specific features.

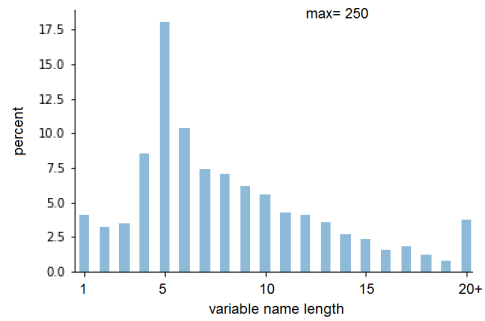

Fig. 7: The distribution of variable's name length in Scratch projects

A. Replication Studies

Our first analyses are the replication studies regarding one letter variables.

1) Variable Name Length: The original study of Beniamini et al. [3] 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"*. Figure 7 shows the distribution of lengths in the Scratch dataset. A closer look at the data reveals that the single-letter variables constitute ca. 4% of all the variable names, i.e., less than the 9–20% observed by Beniamini et al. Compared to the five programming languages in the study of Beniamin et. al., single-letter variables seem to be less common in Scratch, while the maximum length of a variable's name –250 characters– is significantly larger. These *Observation*s lead us to wonder whether overall the variable names in Scratch are longer than in other programming languages. To this end, we apply the $\widetilde{\mathbf{T}}$-procedure described in Section IV-D. Statistical analysis reveals that variable names in Scratch indeed are longer than in the textual languages studied in Beniamini et. al. Moreover, variable names in Java tend to be longer than those in PHP, variable names in PHP longer than those in C, variable names in C longer 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.

2) Single-letter Variable Names: Further we investigate the case of single-letter variable names. For the previously studied programming languages, Beniamini et. al. [3] highlight the following *Observation*s about the single-letter usage:

a) 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. 8, the two left-most blocks –forever and repeat 10– do not require a variable to control the loop iterations. However, inside the loop the user will have no access to the built-in loop's iterator. When that is needed by the user, then the third block in Fig. 8 –repeat until– can be used. To understand its use

see Fig. 5. In summary, because of the built-in language support to variable-free loops, we expect the usage of the variable name i to be less common in Scratch.

b) Apart from the popularity of i the distribution is language-dependent. Since Scratch is quite different from the programming languages considered by Beniamini et al., we expect the distribution of the single-letter variable names in Scratch to be different to the distributions in those languages. Hence, we expect the *similarity* between Scratch and the languages considered by Beniamini et al. to be lower than the *similarity* between those five languages in Beniamini et. al. study.

c) Finally, the authors observe 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: Scratch blocks that are used to repeat specific actions

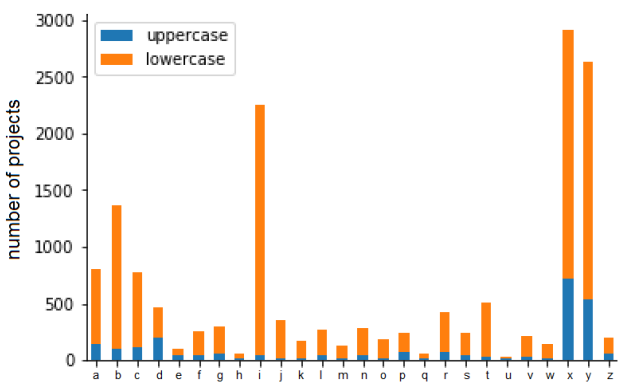

Fig. 9: A histogram of single-letter variables occurrences in Scratch projects

Figure 9 shows the distribution of variables of one letter, in upper and lower case, in the Scratch dataset. 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 the fact that x and y represent the coordinates of the sprites on the stage. They thus form the basis of moving sprites in the 2-D stage. There seems to be an agreement on this use among the developers of textual languages studied in Beniamini et. al. When they were surveyed about variable name interpretations, x and y were commonly interpreted as "coordinates". In addition, built-in Scratch blocks often use x and y as shown in Fig. 10. We conclude that Scratch programmers seem to be inspired by the Scratch language in naming their own variables.

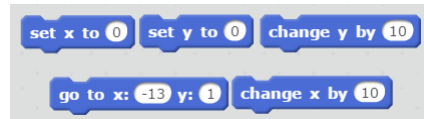

Fig. 10: Some Scratch blocks that use built-in variables x and y

Next we study the similarity of Scratch to the five programming languages in terms of frequency distribution of single-letter variable names. Hence, we compare the mean cosine similarity of the twelve distributions (the five 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 similar to the way letters are used in the five programming languages (0.62). Hence, we claim that our expectation based on *Observation b*) has been confirmed for the uppercase letters and rejected for the lowercase letters. Closer look at Fig. 11 shows that the usage of single uppercase letters is similar to the pattern found in Perl, for the lowercase—to the pattern found in Java.

Finally, Fig. 9 clearly shows that the lowercase letters are much more frequently used as variable names than the uppercase letters, providing support for *Observation c*).

> Single-letter variable names are less common in Scratch than in other programming languages, and Scratch variables have longer names than variables in other programming languages.

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. 5. Figure 12 shows the distribution of the procedure names 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 **Alaaeddin To Alex: rev. 2 suggests putting p¡0.0001 for example.**), i.e., it indicates that the procedure names' lengths tends to be larger than the lengths of the variable names. This *Observation* is additionally confirmed by the Mann-Whitney-Wilcoxon test (the p-value is too small to be computed precisely). Short procedure names are not common, they are even less common than variable names: single-letter procedure

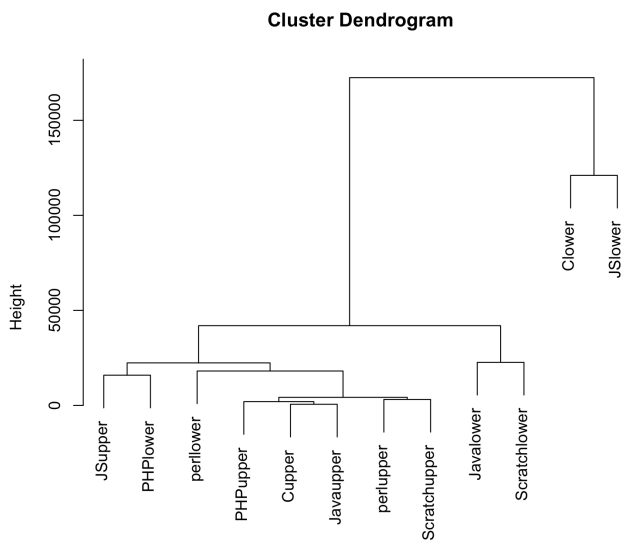

Fig. 11: A cluster dendrogram of Scratch compared to other programming languages for the single-letter pattern

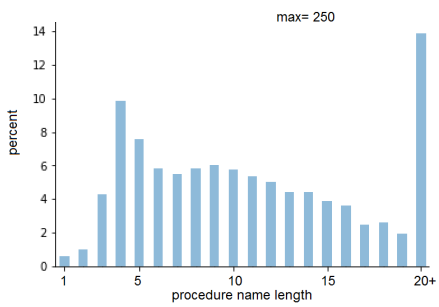

Fig. 12: The distribution of procedure's name length in Scratch projects

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 variables [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 previous versions of Scratch. The 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*) with respect to procedures since data on single-letter procedure names was not collected for the five programming languages in the study of Beniamini et. al. Figure 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

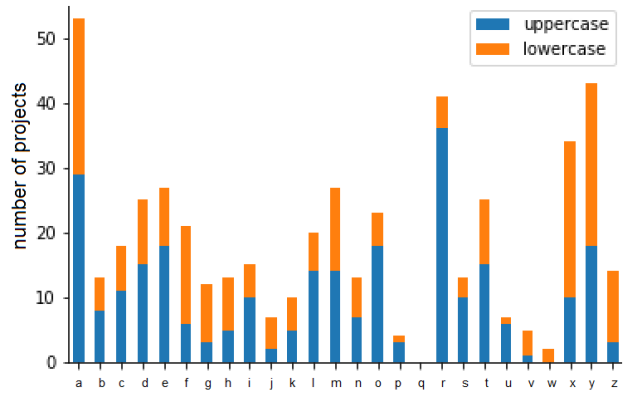

Fig. 13: A histogram of single-letter procedures occurrences in Scratch projects

more often than the lowercase letters (in 267 vs. 218 projects), rejecting *Observation c*).

4) Types: In the paper of Beniamini et al. [3] the authors observe that some letters are highly associated with the data type starting with the same letter: e.g., char for c and string for s. They also highlight that the integer data type is a common association for many other letters. Furthermore, a survey for developers showed that variable names x, y and z are commonly interpreted as coordinates, and for these letter it seems a balance exists between integer and float associations.

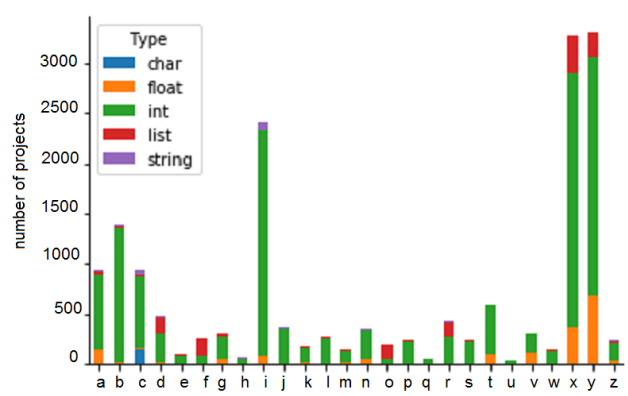

Fig. 14: Inferred types for variables of one letter

As explained in Section IV-A we conduct a conceptual replication by considering the types of variables as they are used, as opposed to types as guesses by programmers. Figure 14 shows the distribution of single-letter variables with the types inferred. The majority of single-letter variables are encountered as integers in the Scratch dataset. This partially agrees with their *Observation* of integers being common for many letters, however, the data types are less diverse in Scratch compared to the five programming languages considered. One *Observation* that contradicts *Observations* in the original study is related to the string data type. While string data type in the

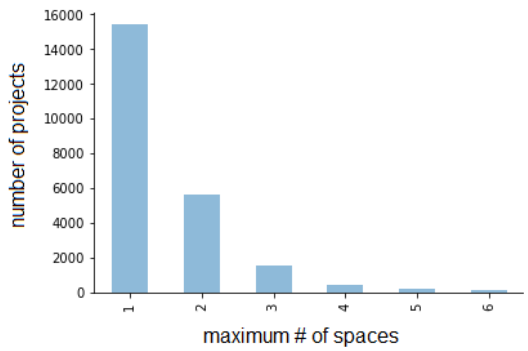

Fig. 15: Number of spaces in variable names

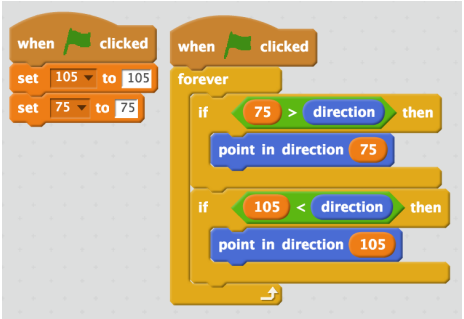

Fig. 16: Numeric variable used as a constant

five programming languages studied is commonly associated with s and less frequently with many other letters, the strings are almost completely absent in the Scratch dataset apart from i. This is also observed in the other data types where no noticeable usage could be observed for floats, lists or strings. The only exception to some extent is the variable c where we observe some usage linked with the $char$ data type. Finally, the presence of floats as types for x, y and to lesser extent z seems to support the *Observation* that these single-letters variables are perceived 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) Spaces in Variable Names: Most textual programming languages do not allow spaces in variable names. FORTRAN ignored spaces, so 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 be avoided [42]. To understand the usage of spaces in variable names, we measure their presence across the unique variable names, and across projects. Out of 67,286 unique variable names in the dataset, we find that 44.05% have at least one space –30.95% have one space, 9.91% have two and 2.15% have three space characters. For the usage of space character per project, we count the maximum count of a space character in the project's variable names. Out of 69,045 projects which include variables, we find 34% include variables names with at least one space character. As Fig. 15 shows, variable names with one space character is the most prevalent. We conclude that Scratch programmers prefer natural language naming of a variable: the use of a space character in variable names is common to some extent for many users, indicated by the project count, and for many used variable names.

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 using 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

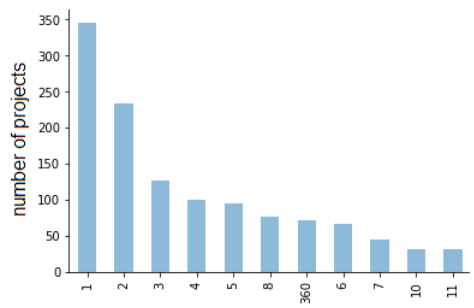

Fig. 17: The most popular numeric values used as variables

names include small natural numbers (1–12) and 360 likely to represent 360° (cf. Fig. 17).

There seem to be two main uses of numeric variable names. First, some variables with numeric names represent constants, as shown in Fig. 16. This seems to indicate the Scratch programmers prefer to drag in a constant rather than repeatedly type 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. In that project, the programmer defined nine variables named 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 which is visible from the fact that Scratch allows users to insert textual labels in between parameters in order to make procedures more readable, as shown in Fig. 5. This practice seems particularly idiomatic to Scratch, since built-in Scratch blocks use a similar syntax, e.g., in the "say ... for ... seconds" block. In total 4,414

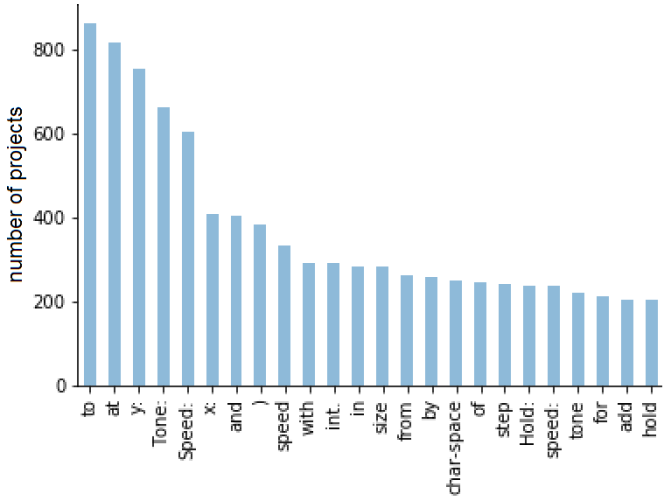

Fig. 18: The most used textual labels in between parameters of procedures

projects use textual labels accounting for 25% of projects with procedures, and 1.77% of all projects in the dataset. Their use is relatively uncommon, however, we do find some interesting patterns. Figure 18 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:', and the use of a closing bracket. We suspect the lack of the opening bracket (to be caused by the fact that it would be included in the procedure name. Furthermore we see the use of ':' at the end of many patterns, which could come from the users being inspired by Scratch default blocks, which use the colon as shown in Fig. 10. Finally, the use of the space (char-space in Fig. 18) is interesting, since Scratch already leaves some room between the parameters, also when a space is not used.

VI. Threats to Validity

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 the lack of explicit type declaration in Scratch our approach inferred the type by inspecting the value assigned to the variable. To be representative for a project, our method counted the different types encountered for the same variable within a project. For example if the variable "score" was set to "empty" at first, and then set to 0 in a different script, we count two types for the variable: one string and one integer. The same can be applied in textual languages with less strict type systems such as PHP. While the method is simple, it assures that data types used by Scratch programmers are represented in the data, and forms a good proxy to compare the results to the perceptions of professional developers. It is possible that these data types do not reflect the perception of users precisely. This is why this threat will be addressed in future work by surveying students in our running Scratch MOOC.

Another threat to the construct validity is our decision of what is a variable name in Scratch. We choose to include the names of variables created specifically through "Make a Variable" button in Scratch UI. A possible threat is that this approach does not represent other types of variables in a Scratch programs. Sprites, for example, can be named, and thus considered as variables. However, we believe they should be excluded from our study, since we perform a dependent replication study to variable names in textual languages. Therefore, what we consider as a Scratch variable must hold a major property similar to variables in the textual languages: its name must be entered by the user. In Scratch, however, sprites are assigned default names automatically, and they often remain unchanged by the users [31].

One threat to the internal validity of our study comes from the use of non-Latin characters in identifiers. These characters are encoded as series of question marks, which is a limitation we inherit from the original dataset. Since one non-Latin character can be encoded as several question marks, the variable/procedure names length reported would overestimate the actual length. However, the impact of these identifiers is limited as the majority of Scratch programmers are from countries that use Latin alphabet[8]. To validate this, we manually analyzed the variable names extracted and found that merely 450 variables have non-Latin alphabetic characters, less than 0.67% of the total 67,287 unique variable names. Therefore, we did not exclude names with non-Latin characters from the analysis.

Finally, a threat to the external validity concerns the generalization of the study results. We argue that we use a large dataset which comprises around 1% of 23 million currently shared Scratch projects. It could be that the dataset does not reflect the users trend for identifier naming. However, the Scratch community represents novice programmers with younger age. For these users, it is difficult to imaging they have an established trend in naming without prior education.

VII. Conclusion

In this paper, we study naming patterns for variables and procedures 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 programmers most often use variable names between 4 and 10 characters in length, while procedure names in Scratch tend to have longer names than the variables. For the single-letter variables, the most commonly used names are x, y and i. When compared to the other programming languages, Scratch variable length distribution, 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. Spaces in variable names, a feature relatively unique to Scratch, are used in 34% of projects in which variables can be found. The usage of textual

[8]https://scratch.mit.edu/statistics/

string between parameters appears as not so common, however textual patterns used imply an inference from textual languages by using brackets for example.

The paper makes the following contributions:

- A detailed analysis of variables name length and single-letter variables in particular, replicating [3] on the Scratch programming language
- An analysis of procedure names in Scratch
- An analysis of naming patterns unique to Scratch, including spaces in variable names, numeric variable names and textual labels in procedures.

Studies concerning the difficulties novice programmers have when transitioning to textual programming languages highlight that handling identifier naming is one of the fundamental challenges faced by these learners, as part of a broader spelling challenge [43]. We believe the naming patterns found in our study support the claim that challenges will be faced by Scratch programmers when transitioning to mainstream textual programming languages. Those languages restrict the use of spaces in identifiers and more often divert into short and single-letter names—tendencies opposite to the naming preferences in Scratch.

This paper gives rise to a number of directions for future work. Firstly, Beniamini et al. [3] included a survey in which they ask developers to predict the type of a (one letter) variable. It could be interesting to ask a similar question to Scratch programmers 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. Finally, one additional area to explore is the relation between name length of variables with the level of computational thinking of the Scratch programmer in addition to other demographic factors like gender, age and language.

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*, 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] F. Hermans and E. Aivaloglou, "Teaching software engineering principles to K-12 students: a MOOC on Scratch," in *ICSE-SEET '17 Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, 2017, pp. 13–22.

[11] K. Rother, *Cleaning Up Code*. Springer, 2017, pp. 195–212.

[12] E. S. Raymond, *The New Hacker's Dictionary*. MIT Press, 1996.

[13] 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.

[14] 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.

[15] B. M. Hill and A. Monroy-Hernández, "The remixing dilemma," *American Behavioral Scientist*, vol. 57, no. 5, pp. 643–663, 2013.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] B. Caprile and P. Tonella, "Restructuring program identifier names," in *ICSM*. IEEE, 2000, pp. 97–107.

[22] 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.

[23] 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.

[24] 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.

[25] F. Deißenböck and M. Pizka, "Concise and consistent naming [software system identifier naming]," in *IWPC*, May 2005, pp. 97–106.

[26] B. Caprile and P. Tonella, "Nomen est omen: analyzing the language of function identifiers," in *WCRE*, Oct 1999, pp. 112–122.

[27] A. Caracciolo, A. Chis, B. Spasojević, and M. Lungu, "Pangea: A workbench for statically analyzing multi-language software corpora," in *SCAM*, Sept 2014, pp. 71–76.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] K. Brennan, C. Balch, and M. Chung, *Creative Computing*. Harvard Graduate School of Education, 2014.

[33] 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.

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

[35] 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.

[36] 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.

[37] F. Konietschke, M. Placzek, F. Schaarschmidt, and L. Hothorn, "nparcomp: An R software package for nonparametric multiple comparisons and simultaneous confidence intervals," *Journal of Statistical Software*, vol. 64, no. 9, pp. 1–17, 2015.

[38] 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.

[39] 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.

[40] 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.

[41] K. Neubert and E. Brunner, "A studentized permutation test for the nonparametric Behrens-Fisher problem," *Computational Statistics & Data Analysis*, vol. 51, no. 10, pp. 5192–5204, 2007.

[42] M. Bochud, *Estimating Heritability from Nuclear Family and Pedigree Data*. Humana Press, 2012, pp. 171–186.

[43] M. Klling, N. C. C. Brown, and A. Altadmri, "Frame-based editing," *Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE '15*, 2015.