Indentation: Simply a Matter of Style or Support for Program Comprehension?

Jennifer Bauer, Janet Siegmund
University of Passau
Passau, Germany

Norman Peitek
Leibniz Institute for Neurobiology
Magdeburg, Germany

Johannes C. Hofmeister
University of Passau
Passau, Germany

Sven Apel
University of Passau
Passau, Germany

***Abstract*—An early study showed that indentation is not a matter of style, but provides actual support for program comprehension. In this paper, we present a non-exact replication of this study. Our aim is to provide empirical evidence for the suggested level of indentation made by many style guides. Following Miara and others, we also included the perceived difficulty, and we extended the original design to gain additional insights into the influence of indentation on visual effort by employing an eye-tracker. In the course of our study, we asked 22 participants to calculate the output of Java code snippets with different levels of indentation, while we recorded their gaze behavior. We did not find any indication that the indentation levels affect program comprehension or visual effort, so we could not replicate the findings of Miara and others. Nevertheless, our modernization of the original experiment design is a promising starting point for future studies in this field.**

***Index Terms*—Code Indentation, Program Comprehension, Visual Effort**

I. Introduction

Program comprehension is an important cognitive process, as programmers spend about half of their time comprehending source code [1], [2]. To support developers in this process, coding conventions have been proposed, which are meant to help them to communicate with other developers, including reviewers, maintainers, and testers of their code. For illustration, we show an example for Pascal in Listing 1 and the same example for Java in Listing 2. The coding conventions for the two languages suggest different ways of formatting the respective code: In Java[1], the guideline suggests to name variables in a lowerCamelCase style. The coding conventions for Pascal[2] recommend to name both variables and constants in a UpperCamelCase style. The two guidelines also contain different rules for indentation: While indentation in Java should consist of four spaces, Pascal code should have an indentation of two spaces. Agreement on a common guideline is supposed to ease the communication between developers and support them in code diffing and merging.

Similar style guides exist for most modern programming languages, sometimes augmented with individual flavors (e.g., Google's JavaScript style guide [3]). A suggested level of indentation is often part of those recommendations for programming languages. However, recommendations are typically not explicitly justified and are only a guess about the most suitable level of indentation by the authors of guidelines. This is surprising, because although indentation is to a large degree a subtle component of style, it is omnipresent and can hardly be escaped when working with code.

```
1  program factorial;
2  const
3    OriginalValue=6;
4  var
5    i,Fact:integer;
6  begin
7    Fact:=1;
8    for i:=1 to OriginalValue do
9      Fact:=Fact * i;
10   writeln(Fact);
11 end.
```

Listing 1. Factorial in Pascal

```
1  class Factorial{
2     public static void main(String[] args) {
3        final int ORIGINAL_VALUE = 6;
4        int i = 0, fact = 1;
5        for (i = 0; i < ORIGINAL_VALUE; i++;) {
6           fact = fact * i;
7        }
8        System.out.println(fact);
9     }
10 }
```

Listing 2. Factorial in Java

Some languages, such as Python [4], explicitly require indentation for indicating the beginning and end of block structures. In such cases, programmers have to commit to the specific language requirements and are limited in the usage of indentations as stylistic device. In languages such as Java, indentation is purely optional. Here, indentation does not bear any meaning, so it can be set arbitrarily and is left to personal preference. For Java, there are several suggestions for code conventions, and the best known might be style guides of Oracle [5] and Google [6], who propose an indentation of four spaces.

Despite guidelines, it is unclear how the level of indentation actually affects the comprehensibility of Java code. One early study was conducted by Miara and others [7] in 1983 ("Program Indentation and Comprehensibility"). It focused on Pascal and showed that an indentation of 2 to 4 spaces is most helpful for comprehending code [7]. Since the results of this study are limited to Pascal and more than 35 years old, we set out to evaluate whether the recommendation of 2 to 4 spaces still holds in modern programming languages.

[1]https://google.github.io/styleguide/javaguide.html#s5-naming
[2]http://www.sourceformat.com/coding-standard-pascal-gnu.htm

To this end, we conduct a non-exact replication of the study of Miara and others, adapted to Java and with students instead of professional programmers. As a further refinement, we are interested in whether different spaces of indentation affect visual effort (i.e., how much effort participants spend to understand a snippet), so we recorded the eye movements of participants with an eye tracker.

Our results indicate that the level of indentation has *no effect* on program comprehension or perceived difficulty of the code snippets, as it neither affected the time the participants needed for solving tasks nor the correctness of the answers nor their rating of difficulty. Furthermore, indentation levels *did not affect* visual effort of participants (a difference for fixation rate vanishes in post hoc tests).

In summary, we make the following contributions:

- We report a non-exact replication of the first study on the effect of indentation levels on program comprehension
- We modernized the experiment design to include typical code fragments in Java
- We added eye tracking to evaluate the effect of indentation levels on visual effort
- We provide a replication package and share all collected data at https://github.com/brains-on-code/indentation/

II. Original Study

In this section, we summarize the original study by Miara and others [7]. The authors of the original study asked 86 programmers with varying experience levels to answer questions about one code snippet with varying indentation levels and to estimate its difficulty. The participants had 20 minutes to complete the tasks and the rating of difficulty. Each participant received *one* of seven code versions.

A. Variables

The original study used a single Pascal code snippet that calculated how often a word occurred in a given input. It used several syntactic structures that are typically indented, such as while loops and if-then-else statements. It was treated with *different levels of indentation*, which was the first independent variable. Specifically, the snippet came in four different configurations, with zero, two, four, and six spaces of indentation. Additionally, the code within Pascal's begin-end blocks was either aligned with the begin and end statements or also indented. Together with the different spaces of indentation, this resulted in seven different versions of the code snippet (see the project's Web site for details).

In their design, Miara and others considered *programming experience* as additional factor. To this end, participants were categorized either as novice or expert, based on the number of years they had been programming. Participants who had been programming less than three years in school or less than two years in practice were categorized as novices, all others as experts.

The first dependent variable was *program comprehension*. To measure it, the authors asked nine multiple choice questions about the program (e.g., "The maximum number of input records is?" or "The output is?" followed by choices) [7]. A higher score in the quiz indicated better program comprehension. Additionally, participants were asked to give a description of the function of the program. The answer was given gradual credit depending on its accuracy. This credit was then added to the overall score.

The second dependent variable was the *perceived difficulty* for comprehending the code snippet—participants should rate the difficulty on a discrete scale from 1 (very easy) to 7 (very hard) with 4 representing a moderate level of difficulty.

B. Results

Miara and others found that two-spaced indentation led to the highest number of correct answers, followed by four spaces. The zero-indented code snippet had the fewest correct answers. As for perceived difficulty, two spaces were rated easier than other indentations, and zero spaces was rated as the most difficult. Independent of the indentation level, experts performed overall better than novices and perceived the snippets as easier.

III. Replication

In this section, we describe our non-exact replication of the study of Miara and others. We highlight the changes that we made in comparison to the original study in each subsection.

A. Material

We decided to convert the Pascal snippets to Java as a modern programming language. The four snippets that we used in the study were selected under the aspects of being similar enough to be comparable to each other (e.g., regarding length, complexity), but at the same time being different enough to avoid learning effects. Each snippet treats a problem involving an array (of numbers or subsequences of a string).

The snippets are similar to standard text book problems and were adapted from the code snippets of previous work [8], [9], [10]. They all contain two block structures (i.e., if-then-else block or loops). To ensure that the snippets are suitable for our study (i.e., such that participants are finished neither too fast nor take too long), we conducted pilot tests. Based on the results of the pilot tests, we replaced one snippet that was too easy (implementing the replacement of elements in an array with a constant value except for one index). In Listings 3 to 6, we show all four snippets that we used in our study.

The first snippet calculates the difference of the respective sums of odd and even numbers in an array (Listing 3). The second snippet adds characters to a resulting string depending on a comparison of a value and an element of a traversed array of numbers (Listing 4). The third snippet sums up the integers contained in intervals, which are given in a string (Listing 5). The last snippet counts the elements of an array that are above and below certain thresholds (Listing 6).

Each snippet has 17 lines of code. This way, we ensured that the snippets are not too trivial and that the eye-tracker was able to differentiate the points of gaze. Each Java class contains a single `main` method, which, in turn, contained at least one

```
1  public class Do {
2     public static void main (String[] args) {
3        int[] array = {5, 6, 11, 0, 2};
4        int integer = 0;
5        int number1 = 0;
6        int numberA = 0;
7        while (integer < array.length){
8           if (array[integer] % 2 == 0){
9              number1 += array[integer];
10          } else {
11             numberA += array[integer];
12          }
13          integer++;
14       }
15       System.out.println(numberA - number1);
16    }
17 }
```

Listing 3. Subtraction of odd and even numbers.

```
1  public class Main {
2     public static void main(String[] args) {
3        int[] values = {3, 0, 1, 0, 2};
4        StringBuilder result = new StringBuilder();
5        int variable = 3;
6        for (int value : values) {
7           if (value == variable) {
8              result.append("x");
9           } else if (value < variable) {
10             result.append("m");
11          }
12          result.append("o");
13          variable --;
14       }
15       System.out.print(result);
16    }
17 }
```

Listing 4. Conversion of numbers into characters.

```
1  public class Program {
2     public static void main(String[] args) {
3        String input = "1-3,10-11";
4        int output = 0;
5        for (String part : input.split(",")) {
6           String[] numbers = part.split("-");
7           int left = Integer.parseInt(numbers[0]);
8           int right = Integer.parseInt(numbers[1]);
9           int number = left;
10          while (number <= right) {
11             output += number;
12             number++;
13          }
14       }
15       System.out.println(output);
16    }
17 }
```

Listing 5. Sum of interval numbers.

```
1  public class Test {
2     public static void main(String[] args) {
3        int variable = 0;
4        String string = "3 21 4 2 55 0 13";
5        int start = 2;
6        int end = 18;
7        String[] keys = string.split(" ");
8        for (int i = 0; i < keys.length; i++) {
9           int key = Integer.parseInt(keys[i]);
10          boolean check = (key >= start && key <= end);
11          if (check) {
12             variable += 1;
13          }
14       }
15       System.out.print(variable);
16    }
17 }
```

Listing 6. Element count of values above and below thresholds.

print statement, determining the output of the program. We chose identifier names such that they did not hint at their function in the code (e.g., `number1`, `variable`), and we named the classes vaguely (e.g., `Program`, `Main`). This way, we ensured that participants could not guess the output based on the identifier names, but instead had to focus on actually comprehending the snippets. This way, we enforced bottom-up comprehension of the code, which reduces the effect of prior knowledge on the way of understanding the snippet [11].

Unlike the original study, the snippets implemented different problems and were shorter (17 compared to 101 lines). This way, we could compensate for our small sample by using a within-subject design: one participant saw all four different levels of indentation on all four different snippets. Furthermore, the snippets needed to fit on one screen without scrolling to ensure a reliable identification of code location to participants' eye gaze. Technically, eye trackers support scrolling, but this adds additional complexity to the setup and analysis to ensure accurate recording of eye movements.

B. Variables

We varied the level of indentation as an independent variable. Similar to the original study, we implemented four levels: zero, two, four, and eight spaces. Note that the fourth level had a depth of eight spaces, instead of six spaces, because we aimed at maximizing the effect of highly indented code, while keeping it at a reasonable level. For indentation of code blocks, we followed Oracle's Java coding conventions [5], so code blocks that belong to the following constructs were indented:

- Class header
- Method header
- Loop header
- `if`, `else if`, and `else`

Unlike in the original study, we did not treat programming experience as independent variable, because we could not recruit a sufficient number of experienced programmers for our study (15 out of 22 participants were students).

As dependent variables, we measured program comprehension, perceived difficulty, and visual effort. We operationalized program comprehension and perceived difficulty different than in the original study:

a) Program Comprehension: To measure program comprehension, we asked the participants to determine the output of a snippet. An answer was counted as correct if it matched exactly the actual output (e.g., "xoxxom" was not given partial credit, when the right answer was "xooxom"). To this end, participants needed to examine the source code line by line and keep track of the values of the variables.

We deviated from the multiple-choice questions and summary of the original study, because our goal was that participants have to go through the code snippets line by line, until having found an answer. This allowed us also to measure visual effort with an eye tracker.

In addition to correctness, we also measured the response time for each task, from the time a snippet was presented until a participant submitted their answer. We decided to include response time (in contrast to the original study), because the duration of the comprehension process is also

an indicator for the effect of different indentation levels on program comprehension.

b) Perceived Difficulty: We asked participants to rate each snippet's difficulty. To this end, we displayed all snippets at once and asked the participants to sort the snippets from easy to difficult, thus enforcing a choice on a relation among the snippets. We chose this method because it encourages the participants to compare the snippets directly with the other snippets they had seen. An alternative would have been to display the individual snippets one after another and asking for a rating on a certain scale. We dismissed this idea, because we anticipated that this method would cue participants to give an absolute rating of how difficult they felt the previous tasks had been and compare each snippet to snippets they had encountered outside of the study. We deviated from the absolute rating of difficulty, because it allowed us to reduce the influence of inter-individual differences, such that each participant sees different levels of indentation and can focus on the rating of difficulty more than on the different indentation levels, rather than on the complexity of the snippet itself. For rating difficulty, we displayed the snippets next to each other, along with the respective answers given in the preceding trials, to indicate that the participant had in fact seen the snippet earlier.

The participants rated twice: First, all snippets were displayed with a normalized indentation depth of four spaces. This way, we aimed at learning about their perception of difficulty without giving hints about the role of indentation in the study. Second, we showed the code with the actual indentation that participants saw in the comprehension tasks, that is, with different indentation levels. This way, participants could take the level of indentation into account for their rating. For analysis, we used both ratings of difficulty. This way, we reduced the influence of different complexities of the individual source-code snippets. Our method of evaluating the participants' estimation of difficulty differs from the one used by Miara and others, who let the participants rate the difficulty on a scale from 1 to 7.

c) Visual Effort: In addition to the original study's setup, we included eye tracking to measure visual effort, which allows us to discover challenges while reading code. We use *fixations* and *saccades* to operationalize visual effort (much like in Sharif and others' eye-tracking study [12]).

A *fixation* takes place when the gaze is resting on a point. It "lasts anywhere from some tens of milliseconds up to several second" [13]. There are numerous measures regarding fixation, such as fixation position, duration, count, and rate. The transition between two fixations is called a *saccade*. It does not need to follow a straight line, but can be curved before ending in a fixation [13]. Measures regarding saccades include saccadic velocity, saccadic amplitude, and the directions of saccades.

Based on fixations and saccades, we can derive different measures that describe visual effort (see Holmqvist and others [13] for a detailed overview). In our study, we selected fixation duration, fixation rate, and saccadic amplitude, because they are the most widely used and thus well-understood measures for visual effort:

Fixation Duration: The duration of a fixation is "likely to be the most used measure in eye tracking research" [13]. In 1980, Just and Carpenter proposed "that there is no appreciable lag between what is being fixated and what is being processed." [14]. This implies that the longer a fixation is, the longer the fixated part is processed, and the larger the visual effort is. This statement has to be handled with some caution, because "some processing trace of a fixated item may continue for a very long time after the eye has left the fixated position" [13]. Nevertheless, fixation duration is a good indicator for processing text or code, because only visually perceived items can be handled by the reader. We expected the level of indentation to affect fixation duration, because participants have to adapt their gaze behavior not only to the different layout, but also to the assumedly differing difficulty of the code. For example, a well-formatted piece of code with the supposedly optimal level of indentation could allow participants to concentrate more on certain code lines, therefore increasing their fixation duration.

Fixation Rate: The number of fixations per second is known as the *fixation rate*. It is related to fixation duration, but "includes saccade and blink duration" [13]. Nakayama and others found that fixation rate (or "gazing time" as they call it) decreases when task difficulty increases [15]. A high fixation rate implies that the reader jumps more from fixation to fixation, so from item to item, resulting in higher visual effort. In contrast, a low fixation rate may either occur when people have a harder time to process some parts of a code snippet, or when they feel no need for looking back for understanding single items. Like with fixation duration, we expected this measurement to be influenced by indentation. For example, non-indented code could increase the number of saccades in the code to figure out corresponding blocks of code, resulting in an increased fixation rate.

Saccadic Amplitude: The spatial length of a saccade is also known as *saccadic amplitude*. It relates to the jumps made by participants and to the difficulty of a task, such as understanding a code snippet. The more difficult the task, the shorter the saccadic amplitude [16]. Higher difficulty in tasks related to counting was also shown to result in decreased saccadic amplitudes [17]. Naturally, the saccadic amplitude can also be influenced by the layout of the stimulus. If visual clues lie farther apart from each other, the saccades between them cover a larger distance, leading to higher visual effort. We therefore expected that the saccadic amplitude is influenced by the level of indentation, as it changes the distance between ending and beginning of new lines. Furthermore, as code becomes more unstructured, the possible resulting jumping between distant lines of code would affect the saccadic amplitude.

Having presented our independent (level of indentation) and dependent (program comprehension, perceived difficulty, visual effort) variables, we now describe our research questions.

C. Research Questions

The overarching question of our study is: Does indentation affect program comprehension? At first glance, indentation by itself does not directly influence program comprehension: The code is still the same, only differently formatted. In many cases, there is no additional information encoded in this choice. The reason why we believe that indentation could have an effect on program comprehension is its structuring influence: Indentation highlights cohesive code parts, making them easier to detect. The advantage that proper indentation offers is therefore a small gain of time, which could be invested for deeper program comprehension. However, this highlighting effect of indentation could be subverted if the number of spaces used is too high. Stretching the code too far to the right could make it more difficult to keep the bigger picture in mind (the role the indented code lines play in the program's overall function). The surrounding code lines could shift out of focus for programmers and they might have trouble to consider the context of the read lines.

We therefore reason that the level of indentation affects program comprehension in terms of correctness and time, as well as perceived difficulty of the program snippets. Furthermore, as indentation changes the layout of code, we assume that visual effort is also influenced by the level of indentation.

To evaluate these considerations, we formulate the following research questions:

$\mathbf{RQ_{Correct}}$	Does indentation affect correctness?
$\mathbf{RQ_{Time}}$	Does indentation affect response time?
$\mathbf{RQ_{Diff}}$	Does indentation affect perceived difficulty?
$\mathbf{RQ_{Fix:Dur}}$	Does indentation affect fixation duration?
$\mathbf{RQ_{Fix:Rate}}$	Does indentation affect fixation rate?
$\mathbf{RQ_{Sacc:Amp}}$	Does indentation affect saccadic amplitude?

D. Experiment Design

Our study has a within-subject design (every participant sees every code snippet and every level of indentation) with a randomized order of code snippets. This way, we reduced the influence of individual differences, such as programming experience and reading speed. Additionally, we reduced the effects of confounding parameters that are inherent to empirical studies, such as learning and ordering effects.

Randomizing the order of the four snippets results in $4! = 24$ different orders. The same counts for the four different levels of indentation, leading to $24 \times 24 = 576$ possible sequences of code snippets and indentations, where no code snippet and no indentation is repeated. For each participant, one sequence was randomly selected.

This design was different than in the original study with one snippet and between subjects. We selected a within-subjects design to reduce the effect of a possible interaction between code snippet and indentation, for example, such that one snippet is the easiest to understand with an indentation of eight spaces (so, using only this version might lead to the false conclusion that eight-space indentation is the most effective). Furthermore, we could compensate for the smaller sample size.

E. Participants

We recruited 39 participants via personal invitation, word-of-mouth recommendation, and an invitation mass e-mail addressing all students at the University of Passau. However, due to a bug in an auxiliary code script, the order was not randomized for all participants, so we needed to omit the data of 17 participants from the analysis. Hence, all following statements refer to the remaining 22 participants (see Section VI and project's Web site for more details). As compensation, we offered participants sweets and the chance to win an Amazon gift card.

The participants were mainly students of Computer Science and Internet Computing at the University of Passau. Most participants were undergraduates in the fourth or fifth semester (Mean = 4.7, SD = 2.1), who spent, on average, seven hours per week on programming (Mean = 7.4, SD = 7.3).

Another group of participants was working at our department of Computer Science (Ph.D. students, post docs, university staff). Three other participants were employees at the *msg systems ag*, a software company located in Passau, Germany.

F. Procedure

We conducted the study in an office with two desks facing each other. On one desk, the monitor and keyboard for the participants were assembled, the other one was empty. The experimenter took place on this desk. We closed the curtains and roller shutters as far as possible and turned on the light in the room to obtain optimal results from the eye tracker. During the study, a construction site outside the building repeatedly caused a higher noise level, but the participants stated that they have not been disturbed during the tasks.

We used a Tobii EyeX tracker with the 'Tobii Eye Tracking Core Software' in version 2.9.0. The tracker has a sampling rate of 60 Hz and an operating range of 50–90 cm. It is suitable for a display size up to 27" [18]. We used a monitor with a diagonal of 24" and a FHD resolution (1920×1080 pixels / 16:9 aspect ratio) with a refresh rate of 60 Hz. We placed it ~50 cm away from the desk's edge and ~19 cm above the desk's level. We provided a standard keyboard and a wireless mouse.

Before the actual study started, we asked participants to take place in front of the screen and to sit down as if they would do when they had to use the keyboard and mouse. To obtain good gaze data, we asked to them to change their position until they felt comfortable and until the Tobii Tracking Software could reliably detect their eye movements. The experimenter stayed in the room, and we instructed participants to ignore her. Then, the study software was executed. It was a self-written .NET program, which uses the WPF-Framework and handles the logging of the gaze data and the participants' input. The language used in the instructions was German.

The study started with a welcome screen, which was also where the participants gave their informed consent. We asked demographic questions and about their programming experience. When participants declared to be students, we directed them to answer questions about their studies, before getting to the questions regarding programming experience, which all participants answered. Afterwards, the eye tracker was calibrated. We asked the participants to remain in their current position as far as possible to keep the calibration valid. Subsequently, we gave them the instructions for the tasks, including a warm-up snippet to get accustomed to the study. Then, the actual tasks started. The snippets were preceded by a fixation cross of 1.5 seconds. The following task screen showed the stimulus code snippet in the main part of the window. Separated by a thin line, the input field for the output answer was at the bottom part of the screen. The question to be answered ("What output does this code produce?") was written to the left side of this field. With a click on 'Done', participants moved on to the next screen. Each task had a time limit of 5 minutes (which we introduced based on the pilot study, so that the experiment would not last too long for slower readers). No participant exceeded the time limit.

After each snippet, participants had the chance to decide for themselves when to continue. After comprehending all four snippets, participants rated the difficulty of the code snippets. Via drag-and-drop, participants ordered the snippets according to difficulty. Then, we asked participants whether they participated seriously and whether they got distracted during the experiment. Finally, participants could state whether they wanted to take part in the lottery and be informed about the study results.

Each trial took, at most, 30 minutes (up to 20 minutes for the code tasks plus ~10 minutes to answer the questionnaire, read the instructions and the warm-up task.)

IV. Results

In this section, we present the results of our study, followed by the analysis of the influence of the level of indentation on the depending variables.

A. Descriptive Statistics

This section gives an overview of the values of the dependent variables and their distributions. In tables, we show mean and standard deviation of the values, if the data are normally distributed (as the Shapiro-Wilk test [19] indicates), and the median and the interquartile range (IQR), otherwise.

Since *Response time* and *Saccadic Amplitude* are not normally distributed, we calculated the natural logarithm of the values (i.e., a log transformation), on which we base the analysis. The figures and tables show the raw, untransformed values of the data. Note that the graphs show groupings and do not reflect that the data were collected using a repeated-measures design, that is, our statistics account for inter-individual effects, but our graphs do not as they are not able to show the data points collected from the same participant.

TABLE I
Summary of Response Times and Correctness

	Time in Seconds		
Indentation	Median	IQR	% correct answers
0	95.00	34.77	73 %
2	97.55	41.07	68 %
4	93.42	47.79	77 %
8	85.08	59.02	55 %

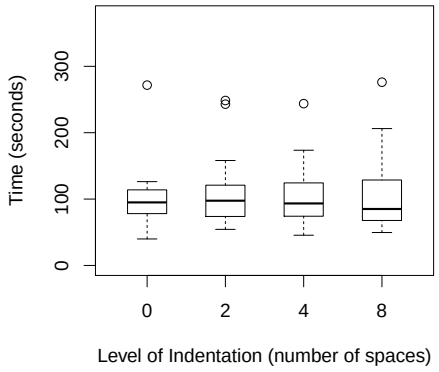

Fig. 1. Distributions of Response Times (untransformed)

1) Program Comprehension: The results for program comprehension (i.e., response time and correctness) are shown in Table I. The time values are not normally distributed but skewed to the left (shown by a p-value of below 0.001 for the Shapiro-Wilk test), which is typical for response times. An indentation of eight spaces resulted in the fastest response time, but in the smallest number of correct answers. Four spaces of indentations led to the highest correctness of answers. Overall, the participants needed the longest time for completing the tasks with an indentation of two spaces.

To minimize the effect of the skewed response times, they can be transformed, for example, by inversion or applying a logarithmic function [20]. Here, we log-transformed the response times [20], leading to normally distributed data (p-value of 0.1 for the Shapiro-Wilk test). With normally distributed data, we can conduct an ANOVA to evaluate our hypotheses (see Section IV-B).

TABLE II
Summary of Position in Rating with Equal and Real Indentations

	Rating Equal		Rating Real	
Indentation	Median	IQR	Median	IQR
0	2	1	2	2
2	2	1.75	2	1
4	1	0.75	1	2
8	1	2	1	2

2) Rating: Table II shows a summary of the position in the rating for the respective level of indentation that a rated code snippet had, when the rating took place with equal indentations (i.e., four spaces) and with the actual indentation of how participants saw them in the study. The medians of the two conditions are equal, only the distributions differ for the indentation levels. In both cases, small levels of indentation (zero and two spaces) were rated as more difficult than larger levels of indentation (four and eight spaces).

3) Visual Effort:

a) Fixation Duration: The median fixation duration of each trial is approximately normally distributed (Shapiro-Wilk test: $W = 0,98$, $p = 0.18$). The results are summarized in Table III. On average, the fixations of participants lasted longest with an indentation of four spaces (i.e., 157 milliseconds), and shortest with non-indented code (146 milliseconds), indicating the highest visual effort for 4 spaces.

b) Fixation Rate: The fixation rate is not normally distributed but skewed to the right with a long tail on the left side ($W = 0.76$, $p < 0.001$). The results are summarized in Table III. Here, participants had the highest fixation rate of 3.45 fixations per second with non-indented code and the lowest with four spaces (3.23 fixations per second), indicating the highest visual effort for 0 spaces.

c) Saccadic Amplitude: The average saccadic amplitudes per trial are not normally distributed ($W = 0.95$, $p = 0.001$). They are skewed to the left and thus were log-transformed for later analysis (similar to response times). The transformed values are normally distributed according to a Shapiro-Wilk test ($W = 0.98$, $p = 0.11$). The untransformed values are summarized in Table III. Overall, the participants' saccades were the longest when they handled non-indented code (i.e., 155 pixels) and the shortest when handling code with an indentation of four spaces (139 pixels), so visual effort appears to be highest for 0 spaces.

B. Statistical Analysis

Next, we answer our research questions. We used two tests for statistical significance of the differences in our data regarding our research questions:

One-way repeated-measures ANOVA: This test compares means of groups that are differentiated by one factor (here: level of indentation), and that were collected from the same source [21]. It requires that the dependent variable is interval-scaled and normally distributed (either directly or after transformation) and that sphericity can be assumed (for example, by insignificance of the Mauchly test [22]).

Friedman test: This is an alternative test for ANOVA when the assumptions are violated, so when data are not normally distributed and/or sphericity cannot be assumed [23]. By ranking the values, the test analyzes the variance of repeated measures derived from one independent variable. It requires that the values between samples are paired and independent within a sample.

Next, we report the results of the significance tests for each research question, followed by an explanation for test selection.

$RQ_{Correct}$: Does indentation affect correctness? → No.
The Friedman test showed that the number of correct answers was not significantly affected by the levels of indentations ($\chi^2(3) = 3.32$, $p = 0.36$).
We used this test, because the dependent variable ('correct answer given') is not interval-scaled.

RQ_{Time}: Does indentation affect response time? → No.
A One-way ANOVA with repeated measures showed that the effect of the level of indentation on the log-transformed response times was not significant ($F(3, 63) = 0.44$, $p = 0.72$, $\eta_p^2 = 0.006$).
We selected this test, because the dependent variable ('Time') is normally distributed after a log-transformation. Sphericity can be assumed, as a Mauchly test indicates ($Mauchly - W(3) = 0.85$, $p = 0.65$).

RQ_{Diff}: Does indentation affect perceived difficulty? → No.
For determining whether the rating positions of the code snippets differed depending on the level of indentation, we applied the Friedman test, once for the rating with equal indentations, and once for the one with the actual indentations. For both, the differences are not significant ($\chi^2(3) = 4.64$, $p = 0.20$ and $\chi^2(3) = 5.35$, $p = 0.15$).
Since the dependent variable ('Rating Position') is ordinal-scaled, we chose the Friedman test.

$RQ_{Fix:Dur}$: Does indentation affect fixation duration? → No.
A One-way ANOVA with repeated measures showed that the effect of the level of indentation on the median fixation duration per trial was significant ($F(3, 63) = 2.85, p = 0.045, \eta_p^2 = 0.028$). However, we could not confirm this difference with post hoc tests, for which we used t tests for dependent samples and false discovery rate (FDR) correction to adjust the p value for multiple testing.
We chose ANOVA and the t test, because the dependent variable ('Median Fixation Duration Per Trial') is interval-scaled and normally distributed. Sphericity can be assumed ($Mauchly - W(3) = 0.77$, $p = 0.39$).

$RQ_{Fix:Rate}$: Does indentation affect fixation rate? → No.
There was no significant difference among the distributions of the fixation rates for the four levels of indentation according to the Friedman test ($\chi^2(3) = 7.36$, $p = 0.06$).
As the dependent variable ('Fixation Rate') is not normally distributed, also not after transformation, we chose the Friedman test.

TABLE III
SUMMARY OF MEDIAN FIXATION DURATION, FIXATION RATE, AND SACCADIC AMPLITUDE

	Fixation Duration per Trial (ms)		Fixation Rate (fixations/second)		Saccadic Amplitude (pixels)	
Indentation	Median	IQR	Median	IQR	Median	IQR
0	146.09	25.94	3.45	0.81	155.53	41.36
2	153.75	24.45	3.38	0.39	144.11	33.23
4	157.41	25.83	3.23	0.67	139.48	22.58
8	153.50	22.04	3.42	0.38	149.09	40.42
Total	152.69	24.54	3.35	0.69	145.24	32.91

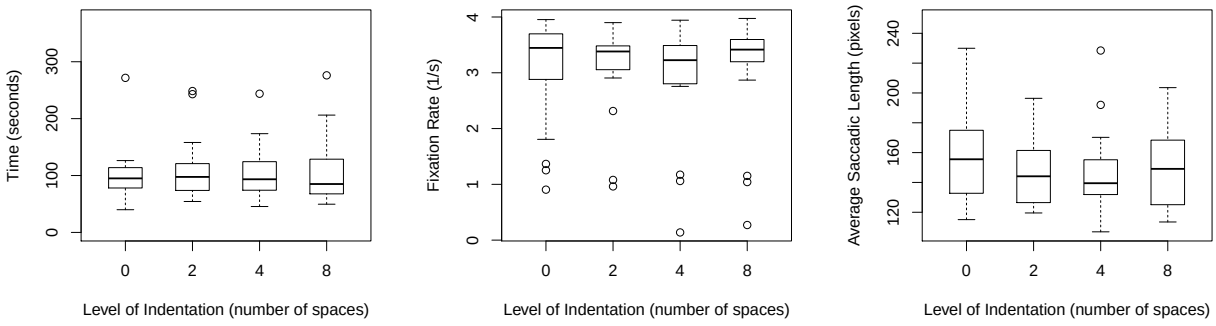

Fig. 2. Distribution of Median Fixation Duration, Fixation Rate, and Saccadic Amplitude (untransformed)

$RQ_{Sacc:Amp}$: Does indentation affect saccadic amplitude? → No.

A One-way ANOVA with repeated measures showed that the effect of the level of indentation on the log-transformed average saccadic amplitude was not significant ($F(3, 63) = 1.69$, $p = 0.18$, $\eta_p^2 = 0.029$).

We selected this test, because the data are normally distributed after a log-transformation and the dependent variable ('Saccadic Amplitude') is interval-scaled. Sphericity can be assumed ($Mauchly - W(3) = 0.87$, $p = 0.74$).

We further discuss the strictness of our statistical analysis in Section VI-C.

V. DISCUSSION

Although we observed subtle differences for correctness, response time, perceived difficulty, and visual effort, none of these differences was statistically significant. Thus, we obtained a *negative result*, showing that guidelines for indentation do seem to be a matter of style.

There are several possible explanations for the negative result. First, the actual effect of the original study was possibly spurious. Maybe comprehensibility was not influenced by indentation but by hidden variables that were not present in our study. Maybe the obtained results only apply to Pascal, but not to Java, or maybe programming education played a more important role. However, without further studies, we cannot know why Miara and others observed an effect and we did not.

Another possible explanation could be the small effect size. Regarding response time, the maximum difference was only 12 seconds (between 2 and 8 spaces, cf. Table I), and the maximum difference for correctness was 22% (i.e., there were 5 more right answers given for snippets with 4 spaces of indentation than for those with 8 spaces). For subjective rating of difficulty, Cliff's delta is also small, that is, -0.17 for equal indentations and 0.27 for the actual indentations. Regarding visual effort, the effect sizes are below 0.03 in terms of eta^2 for fixation duration, fixation rate, and saccadic amplitude. For fixation duration, this translates into a maximum difference of 11 ms (between 0 and 4 indentation spaces, cf. Table III). In other words, participants needed about 11 ms longer to process information when working with 4 spaces of indentation, compared to 0 spaces. Similarly small is the maximum difference in fixation rate (0.22 fixations per second; between 0 and 4 spaces) and saccadic amplitude (16 pixels; between 0 and 4 spaces).

Additionally, the sample size may have played a role: While Miara and others recruited 86 participants, we obtained only 22 records. However, where Miara and others used a between-subjects design with small groups, we used a within-subjects design and each participant saw 4 trials, resulting in 88 valid data points that we used to calculate our statistics. Thus, our sample size was effectively comparable. Wherever possible, we used parametric tests, which have more statistical power than non-parametric tests. However, comparing the power and sample sizes between the studies is difficult, because we were unable to pinpoint the effect sizes of the original study, as the authors did not report all relevant values to calculate an effect size precisely, so our discussion is based on a rough estimation

of these values. Lehman [24] suggests to estimate the sample size for non-parametric tests by adding approximately 15% to an estimation of a similar parametric test. Assuming that the effect of indentations on comprehension is small (between 0.1 and 0.2 [25]), we estimated a required sample size for our study, using G-Power [26]. Aiming at a desirable power of $1 - \beta = 0.8$, we estimated a total sample size of $n = 304$ participants to find a small effect ($f = 0.15$) using a repeated-measures ANOVA at a type-I-error probability $\alpha = 0.05$, with four measurements of four groups. Thus, to find an effect of this magnitude with acceptable power, our sample might simply be too small. It is likely that our design was not sensitive enough to detect such a small effect with the configuration given. However, we believe that a shorter response time in the order of seconds, as we found, does not justify as evidence for an effect of indentation on program comprehension.

Besides the statistical aspects, other code-related factors could have masked an effect, such as the computation performed in the code snippets, the size and complexity of a snippet, or syntax highlighting. It might well be that, the effect of indentation comes more into play when the code is longer and more complex. For example, with a higher degree of nesting (e.g., 4 or 5 nested loops), the depth of indentation may indeed affect comprehensibility of a code snippet. Looking at the study by Miara and others, their snippet was longer (101 lines, compared to 17). Taking the results of both studies into account, it might well be that for snippets similar to ours, indentation depth might not affect comprehensibility, which instead might come only into play with longer and more complex code.

Last, experience may have affected the results. Different to the original study, we mostly recruited novice programmers (i.e., undergraduate students) for our sample. It might well be that they are not yet accustomed to layout guidelines. Soloway and Ehrlich found a similar effect, showing that the performance of experts degraded compared to the performance of novice programmers when their expectations were violated [27]. We might observe a similar effect here, such that programming experience interacts with indentation depth.

In a nutshell, we could not find any evidence that a certain level of indentation is optimal for program comprehension. Thus, especially when working with code that has similar properties as the snippets in our setting (e.g., in terms of complexity and length), personal preferences and experience of developers might be the best recommendation that we can give. Nevertheless, it might be worth to dig deeper into why the personal preferences of indentation emerge in the first place, and how indentation interacts with other properties of source code, such as complexity, nesting depth, or length.

VI. Threats to Validity

A. Internal Validity

While all participants declared that they were not distracted during the tasks, the noise of the construction site might have had an influence on the concentration of participants. Of course, the sheer presence of the eye tracker may have affected the participants' behavior, too. To minimize this threat, we asked participants afterwards how much they were distracted, and none said that it disturbed their work.

Rating the code snippets' difficulty by asking participants to order them does not depict absolute values. Participants were not able to adequately express their perception of difficulty, when they thought all snippets were equally easy or difficult. However, forcing participants to make a decision for each snippet allowed us to compare the relative difficulty of the snippets to each other and not to other code snippets that participants had worked with before.

The eye-gaze attributes examined in this study are only a subset of possible aspects for measuring visual effort. Other measures, such as blink rate or pupil dilation, might have been more informative. We did not measure these because the aim of this study was to get a first insight into how visual effort is affected by indentation, whereas for measuring other factors, a more elaborate eye tracker setup is needed.

The absence of evidence of the effect of indentation and visual effort could originate from the high subjective factor of eye-tracking data, "meaning that one person's parameters are different from another person's, irrespective of task" [13]. The small number of participants might thus hinder obtaining significant results for differences in visual effort. Furthermore, the data obtained by the eye tracker may be too inaccurate to find significant effects in gaze behavior with the materials and setting of this study.

Due to an error in the script for conducting the experiment, the order of the snippets was randomized only for 22 of the 39 participants. To evaluate whether the lack of a randomized order affected the results, we separately analyzed the data of the 22 participants that received a randomized order and compared the results to the analysis of all 39 participants. We found that the effect sizes as well as the means or medians differed between the 22 and 39 participants, indicating that the order of presentation also affected the results. To reduce the effect of order, we only included the data of the 22 participants who received a randomized order in our analysis (see project's Web site[3] for a comparison).

B. External Validity

The number of participants was comparatively small, and they were also mostly students. Thus, the lack of effects of indentation depth can only be very carefully transferred to other settings, such as longer code snippets or more experienced programmers. Nevertheless, our setting is especially relevant for novice students, which comprise a significant population in computer science.

C. Statistical Conclusion Validity

Given our small sample size in comparison with the original study, we need to be careful not to be too conservative in our hypotheses testing. To this end, we used a liberal approach to correct for multiple comparison (i.e., FDR correction, not a Bonferroni correction). Furthermore, we also considered defining a less conservative significance level of 0.1 (as often

used in exploratory studies), but the effects still vanish after FDR correction.

VII. Related Work

There are related studies that explicitly manipulate a layout property as independent variable. Jbara and Feitelson evaluated how regularity of code affects program comprehension [28]. Regularity refers to structural similarity of code fragments, so the more the structural patterns (e.g., several nested for loops) repeat, the more regular code is. They found that with more regular code, participants tend to thoroughly read code in the beginning only, and switch to a scanning gaze pattern once they have grasped the structure. Binkley and others evaluated the effect of identifier naming styles, that is, camelCase and under_score on program comprehension, and found that camelCase styles lead to more accuracy and affected response times, which is modulated by familiarity with the style, such that participants familiar with camelCase are faster, but otherwise under_score style is faster [29]. Sharif and Maletic replicated this study and added eye tracking, finding that participants could spot under_score identifier styles significantly faster, independent of participants familiarity with the styles [12]. Furthermore, identifiers with the under_score style lead to lower visual effort for some of the measures they used. Busjahn and others manipulated the similarity of the execution order of code and the actual location in the source code file [30]. They recorded the eye movements of participants and found that the presentation of code affected the reading order, which is modulated by expertise, such that experts follow the execution order and novices follow the presentation order. All these papers evaluate how code layout affects program comprehension, and in that line they contribute empirical evidence for code style guidelines. Hofmeister and others found that shorter identifier names take longer to comprehend, whereas longer names with more semantic information reduce response times [8].

Different to these four studies, there are several studies that aim at building a model of how a set of layout properties affect the comprehensibility of code. Jørgensen conducted a study that dates before the study by Miara and others [31]. He let human raters create a ground truth of readable and non-readable source code, on which he based a regression model containing several properties of code, which results in a readability score. He found that the average number of `goto`-statements per label has the highest influence on the readability score. In a modernized setting, Buse and Weimer conducted a similar study [32], [33]. Instead of a regression model, they build a classifier and used properties of code as features to learn whether a code snippet is readable or not. They found that the average number of identifiers in a line of code is the best feature in predicting the readability. Lee and others conducted a similar study, with the difference that the ground truth was automatically determined based on readability metrics by Posnett and others [34], [35]. In contrast to Jørgensen and Buse & Weimer, they found javadoc comments to be the best predictor for readability (i.e., when they fail to follow their convention, readability decreases). In contrast to our work, these authors looked for a set of layout properties of code that influence readability, while we explicitly modify one property to evaluate its effect on readability.

In another line of research, indentation has been studied as a measure for complexity and maintainability of code. For example, Munson and Khoshgoftaar considered mean and maximum indentation level as a complexity measure [36]. Hindle and others found that indentation correlates with traditional complexity and maintainability measures [37], such as McCabe's cyclomatic complexity [38] and Halstead's complexity metrics [39]. Finally, Gong and Schmidt used indentation to refine McCabe's complexity measure [40]. In contrast to our focus on the effect of indentation on program comprehension, this line of research uses indentation as a modification to objective complexity measures.

VIII. Conclusion

Suggestions on indentation depth of code blocks are part of most style guides on code layout, yet there is little empirical evidence on optimal indentation depth. Inspired by a study by Miara and others [7], who found that 2-spaced indentation has the best effect on program comprehension, we conducted a non-exact replication to provide empirical evidence for style guides. To this end, we modernized the study design by adapting it to Java snippets and by including eye tracking to measure visual effort.

Our results did not show any effect of indentation depth on program comprehension, perceived difficulty, or visual effort, indicating that indentation is indeed simply a matter of task and style, and do not provide support for program comprehension.

There are several ways to continue this line of research, especially by increasing the complexity of the code snippets, such that scrolling is also necessary. This way, keeping track of the control flow could become more challenging for the participants, so the size of the effect of indentation could increase, simply by its increased occurrence. However, this would also require a more advanced eye tracker that is more robust when measuring gaze behavior. Furthermore, exploring how the visual effort of more experienced programmers is affected by different indentation depths is also on our agenda.

Acknowledgments

Bauer's, Siegmunds, and Peitek's work is supported by DFG grant SI 2045/2-1. Siegmunds work is further funded by the Bavarian State Ministry of Education, Science and the Arts in the framework of the Centre Digitisation.Bavaria (ZD.B).

References

[1] T. D. LaToza, G. Venolia, and R. DeLine, "Maintaining mental models: a study of developer work habits," in *Proceedings of the International Conference on Software Engineering (ICSE)*. ACM, 2006, pp. 492–501.

[2] R. Tiarks, "What programmers really do: An observational study," *Softwaretechnik-Trends*, vol. 31, no. 2, pp. 36–37, 2011.

[3] Google, *Google JavaScript Style Guide*, available online at https://google.github.io/styleguide/jsguide.html; visited on October 9th, 2017.

[4] P. S. Foundation, *The Python Language Reference*, available online at https://docs.python.org/3/reference/lexical_analysis.html; visited on October 8th, 2017.

[5] Sun Microsystems, Inc., *Code Conventions for the Java Programming Language*, available online at http://www.oracle.com/technetwork/java/codeconvtoc-136057.html; visited on September 18th, 2017.

[6] Google, *Google Java Style Guide*, available online at https://google.github.io/styleguide/javaguide.html; visited on September 18th, 2017.

[7] R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman, "Program indentation and comprehensibility," *Communications of the ACM*, vol. 26, no. 11, pp. 861–867, 1983.

[8] J. Hofmeister, J. Siegmund, and D. V. Holt, "Shorter identifier names take longer to comprehend," in *Proceedings of IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2017, pp. 217–227.

[9] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proc. Int. Conf. Software Engineering (ICSE)*. ACM, 2014, pp. 378–389. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568252

[10] J. Siegmund, N. Peitek, C. Parnin, S. Apel, J. Hofmeister, C. Kästner, A. Begel, A. Bethmann, and A. Brechmann, "Measuring neural efficiency of program comprehension," in *Proc. Europ. Software Engineering Conf./Foundations of Software Engineering (ESEC/FSE)*. ACM, 2017, pp. 140–150. [Online]. Available: http://doi.acm.org/10.1145/3106237.3106268

[11] N. Pennington, "Stimulus Structures and Mental Representations in Expert Comprehension of Computer Programs," *Cognitive Psychology*, vol. 19, no. 3, pp. 295–341, 1987.

[12] B. Sharif and J. I. Maletic, "An eye tracking study on camelCase and under_score identifier styles," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2010, pp. 196–205.

[13] K. Holmqvist, M. Nystrm, R. Andersson, R. Dewhurst, H. Jarodzka, and J. Van de Weijer, *Eye tracking: A comprehensive guide to methods and measures*. OUP Oxford, 2011.

[14] M. A. Just and P. A. Carpenter, "A theory of reading: From eye fixations to comprehension." *Psychological Review*, vol. 87, no. 4, p. 329, 1980.

[15] M. Nakayama, K. Takahashi, and Y. Shimizu, "The act of task difficulty and eye-movement frequency for the'oculo-motor indices'," in *Proceedings of the Symposium on Eye Tracking Research & Applications*. ACM, 2002, pp. 37–42.

[16] M. H. Phillips and J. A. Edelman, "The dependence of visual scanning performance on search direction and difficulty," *Vision Research*, vol. 48, no. 21, pp. 2184–2192, 2008.

[17] J. G. May, R. S. Kennedy, M. C. Williams, W. P. Dunlap, and J. R. Brannan, "Eye movement indices of mental workload," *Acta Psychologica*, vol. 75, no. 1, pp. 75–89, 1990.

[18] Tobii, *Tobii Eyex Tracker*, http://tobiigaming.com/product/tobii-eyex; visited on Sep. 12th, 2017.

[19] S. S. Shapiro and M. B. Wilk, "An analysis of variance test for normality (complete samples)," *Biometrika*, vol. 52, no. 3/4, pp. 591–611, 1965.

[20] R. Ratcliff, "Methods for dealing with reaction time outliers." *Psychological Bulletin*, vol. 114, no. 3, p. 510, 1993.

[21] T. Anderson and J. Finn, *The New Statistical Analysis of Data*. Springer, 1996.

[22] J. W. Mauchly, "Significance test for sphericity of a normal n-variate distribution," *The Annals of Mathematical Statistics*, vol. 11, no. 2, pp. 204–209, 1940.

[23] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.

[24] E. L. Lehmann, *Nonparametrics: Statistical Methods Based on Ranks*. Springer, 2006.

[25] J. Cohen, *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, 1988.

[26] F. Faul, E. Erdfelder, A. Buchner, and A.-G. Lang, "Statistical power analyses using g*power 3.1: Tests for correlation and regression analyses," *Behavior Research Methods*, vol. 41, pp. 1149–1160, 2009.

[27] E. Soloway and K. Ehrlich, "Empirical studies of programming knowledge," *IEEE Transactions of Software Engineering*, vol. 10, no. 5, pp. 595–609, 1984.

[28] A. Jbara and D. G. Feitelson, "How programmers read regular code: A controlled experiment using eye tracking," *Empirical Software Engineering*, vol. 22, no. 3, pp. 1440–1477, 2017.

[29] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To camelCase or under_score," in *International Conference on Program Comprehension (ICPC)*. IEEE, 2009, pp. 158–167.

[30] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye movements in code reading: Relaxing the linear order," in *Proceedings of International Conference on Program Comprehension (ICPC)*. IEEE, May 2015, pp. 255–265.

[31] A. H. Jørgensen, "A methodology for measuring the readability and modifiability of computer programs," *BIT Numerical Mathematics*, vol. 20, no. 4, pp. 393–405, 1980.

[32] R. Buse and W. Weimer, "A metric for software readability," in *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis*. ACM, 2008, pp. 121–130.

[33] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, 2010.

[34] T. Lee, J.-B. Lee, and H. Peter, "Effect analysis of coding convention violations on readability of post-delivered code," *IEICE Transactions on Information and Systems*, vol. E98-D, no. 7, pp. 1–11, 2015.

[35] D. Posnett, A. Hindle, and P. Devanbu, "A simpler model of software readability," in *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 2011, pp. 73–82.

[36] J. C. Munson and T. M. Khoshgoftaar, "The dimensionality of program complexity," in *Proceedings of the 11th International Conference on Software Engineering (ICSE)*. ACM, 1989, pp. 245–253.

[37] A. Hindle, M. W. Godfrey, and R. C. Holt, "Reading beside the lines: Using indentation to rank revisions by complexity," *Sci. Comput. Program.*, vol. 74, pp. 414–429, 2009.

[38] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, no. 4, pp. 308–320, 1976.

[39] M. H. Halstead *et al.*, *Elements of software science*. Elsevier New York, 1977, vol. 7.

[40] H. Gong and M. Schmidt, "A complexity measure based on selection and nesting," *SIGMETRICS Perform. Eval. Rev.*, vol. 13, no. 1, pp. 14–19, Jun. 1985.