Impacts of Coding Practices on Readability

Rodrigo Magalhães dos Santos
Instituto de Pesquisas Tecnológicas de São Paulo – IPT-SP
rodrigo.santos@ipt.br

Marco Aurélio Gerosa
Northern Arizona University – NAU
Flagstaff, Arizona
Marco.Gerosa@nau.edu

ABSTRACT

Several conventions and standards aim to improve maintainability of software code. However, low levels of code readability perceived by developers still represent a barrier to their daily work. In this paper, we describe a survey that assessed the impact of a set of Java coding practices on the readability perceived by software developers. While some practices promoted an enhancement of readability, others did not show statistically significant effects. Interestingly, one of the practices worsened the readability. Our results may help to identify coding conventions with a positive impact on readability and, thus, guide the creation of coding standards.

KEYWORDS

Code Readability, Code Comprehension, Programming Style, Coding Best Practices, Software Developers' Opinions Survey

ACM Reference Format:
Rodrigo Magalhães dos Santos and Marco Aurélio Gerosa. 2018. Impacts of Coding Practices on Readability. In *ICPC '18: ICPC '18: 26th IEEE/ACM International Conference on Program Comprehension , May 27–28, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3196321.3196342

1 INTRODUCTION

Coding standards and conventions are common in the software development industry. Most standards aim at improving maintainability, assuring that a shared vision among all team members will be kept across the entire – or most of – code base [1, 7, 13, 14, 19, 23]. On the other hand, there are standards focused on quality attributes other than maintainability, e.g. safety on mission critical software [2, 15].

Despite many available coding conventions, several reports [3, 24] on problems related to low readability can be found in the literature, with significant negative impacts on software projects. For instance, there is evidence of a correlation between high code complexity and a decline in contributions received by open source projects [24] and a correlation between high code complexity and delays in the first contributions made by developers [3].

In this paper, we describe a survey aiming at finding details about how coding practices influence – positive or negatively – on levels of code readability as perceived by developers. We analyzed opinions from two groups of developers: computer science bachelor students and professional programmers of a large Brazilian software company. Given some peculiarities of this survey, we developed a web application tailored to the task. A set of 11 coding practices were derived from code readability models [6, 27]. For each coding practice, a pair of code snippets were defined, so that one snippet adhered to the practice and the other violated it. A random sample of 10 pairs was presented to participants asking them to tell which code snippet they considered the most readable.

This survey aims to answer the following research questions:

RQ.1 How coding practices influence the readability levels as perceived by readers?
RQ.2 What is the influence of readers' characteristics over their perceptions of readability?

Among 11 coding practices assessed, 8 showed statistically significant results: 7 of them increased the readability and 1 decreased it. The remaining 3 practices did not present statistically significant effects. Moreover, further analysis looking for relations between participants' profiles and their opinions did not present statistically significant correlations, suggesting that there is no relation between developers' profiles and code readability as perceived by them.

This article is organized as follows: Section 2 presents a literature review of previous related work. Section 3 outlines details about the web application built for the survey, the coding practices assessed, the process of building the pairs of code snippets and the statistical tests employed. Section 4 summarizes participant profiles, their votes and their comments, followed by the results of statistical tests and a discussion about them. Section 5 discusses threats to validity and, finally, in Section 6, we summarize results and present proposals for further investigations.

2 RELATED WORK

In the early XX century, efforts towards modeling readability began, intending to guide the production of content at degrees of difficulty well suited to target audiences [10]. As programming languages evolved and programs increased in

ICPC '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5714-2/18/05…$15.00
https://doi.org/10.1145/3196321.3196342

complexity, the development of methods aiming at the evaluation of code readability began.

Coding standards and conventions, two elements commonly related to code complexity management [16], have also been discussed in the literature. Although there is no previous work discussing general properties of coding standards, there are studies discussing the effects of adopting coding standards in software development projects.

2.1 Text Readability Models

One of the first studies on the readability of texts in English was published by Edward L. Thorndike in 1921 [29]. Analyzing a large number of works focused on children, he found out that the more often a word is used, easier it is assimilated. Such result suggests that the level of difficulty associated to a word is derived from how frequent it is used, what, by its turn, points to a relation between readability and the conventional use of a language. The word frequency table developed by Thorndike was later adopted for building predictive models of text readability [20].

It is noteworthy that statistical readability models are not intended to *explain* how readability is affected by text features. They are built to explore correlations between text features and readability levels perceived by readers. Moreover, due to their statistical nature, readability models are prone to errors in specific cases. For instance, the expression *"To be or not to be"* is short and only contains frequently used words, what qualifies it to be classified as a highly readable sentence. However, it is known that that sentence bears a deep, philosophical meaning, which no statistical model would be able to detect [8].

Such characteristic is not restricted to text readability models, having also been described in models of source code readability [9].

2.2 Code Readability Models

More recently, the development of code readability models based on logistic models began. Following a process similar to the one adopted in the development of text readability models like the *Flesch-Kincaid readability tests* [12], developers' opinions about several code snippets were collected and used for training models based on static code attributes. These models have been showing results successively closer to developers' opinions in comparison to their predecessors [6, 25, 27], also highlighting the role of code spatial distribution throughout the screen [9].

In this work, we used two of the aforementioned models to derive the coding practices assessed: the models of Buse and Weimer and Scalabrino et al. [6, 27]. To the best of our knowledge, Buse and Weimer were the first to build a code readability model based on logistic regression – logistic models provide outputs within the interval of $]0, 1[$, thus working as classifiers with outputs closer to 1 in the case of inputs classified as readable and 0 otherwise.

The development of Buse and Weimer's model began by electing static code attributes like, for instance, average line lengths and average number of identifiers per line. In order to obtain data to train the model, they surveyed opinions of 120 software developers about 100 code snippets. The answers were used to establish weights to different code attributes measured by their model.

Scalabrino et al. [27] follow a similar process. However, their model focuses solely on textual features, like number of dictionary words used to name identifiers, degree of specificity or generality of meanings, and number of different words within the same code block.

In this work, we used some of the highest-weighted attributes from the aforementioned models. Based on these attributes, we described coding practices intended to optimize such attributes and, thus, optimize code readability as measured by the employed models.

2.3 Coding Standards

We found several works discussing the effect of coding standards adoption on their respective projects [4, 5, 11, 17, 18, 28]. Another important result brought by previous work relates to the "emergency" of coding conventions in open source projects, reporting two interesting phenomena [16]: code contributions more similar to existing code base are more likely to be accepted; and contributions made by a given developer along the time get more similar to the existing code base as that developer get more experienced in the project.

There are works presenting coding standards as a result of years of experience accumulated by their authors [21, 22]. However, only one previous work presenting the creation of a coding standard in a more systematic way was found [11], and its results were assessed in a case study.

3 METHOD

In this section, we describe the web application used to support the survey and present data about the participants invited. The method of analysis of the data obtained from the survey is also discussed.

3.1 Web Application

Tools like Google Forms and Survey Monkey are well suited to a wide range of general purpose internet-based surveys. However, considering some quirks regarding the survey presented in this work, e.g. the need of showing code snippets in a particular disposition and gathering auxiliary data like screen sizes and answering times, we decided to build an application from scratch.

One of our biggest concerns was to guarantee that most of the participants were not going to abandon the survey before finishing it. Therefore, one major design principle was to make all data fields optional. Every participant's action was recorded for subsequent analysis. Even some seemingly trivial actions were considered, as they could contribute to the final results in unanticipated ways.

The application comprises a) an initial screen to pick a user interface language; b) a screen with instructions; c) screens

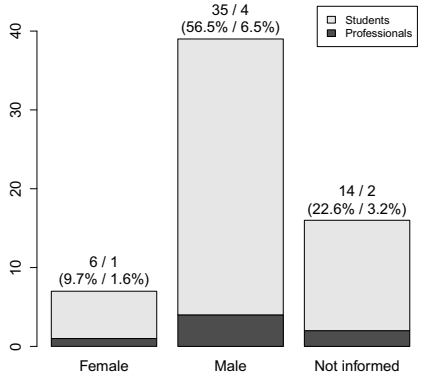

Figure 2: Distributions of Genders, as informed by participants

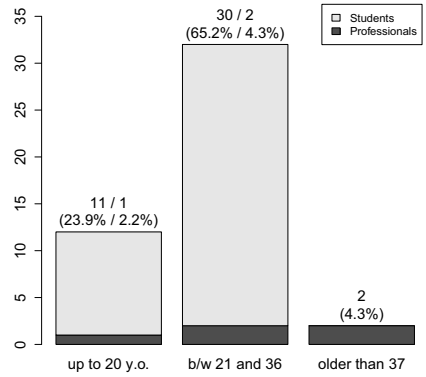

Figure 3: Distributions of ages. Uninformed ages were omitted

to show code snippet pairs and collect opinions and comments from participants, and d) a form to collect profile data. The screen showing code snippet pairs is schematically represented in the Figure 1.

3.2 Invited Participants

Developers invited to take the survey are from two groups. The first and larger one comprises 55 students of the Software Engineering discipline in the Computer Science course of an American University. This group took the survey as part of their extra class activities.

The second and smaller group is composed of 7 professional programmers, employees of a large Brazilian software development company. This group took the survey answering an invitation sent to all programmers of one of the company's software factories.

Figures 2, 3 and 4 present, respectively, the distributions of genders, age brackets, and programming experience time.

3.3 Code Snippets Composing

The search for code snippets was executed using the sample GitHub database accessible through Google BigQuery[1]. We queried repository names and source code file paths taking the following predicates into account: a) files with a `.java` extension, and b) files containing `import` clauses.

Results were sorted in descending order of number of `import` clauses. This way, files with a larger number of external dependencies were at the top of the list. This was employed to obtain files with non-trivial code, as we considered the number of dependencies as a proxy of complexity.

Source code files in the resulting list were inspected and their contents were assessed looking for sections useful to illustrate the coding practices which effects we intended to evaluate.

[1] https://cloud.google.com/bigquery/

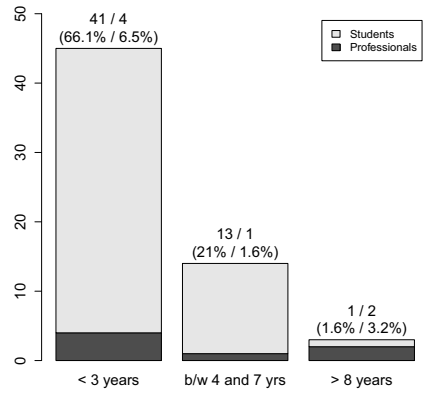

Figure 4: Programming Experience Time, as informed by participants.

3.4 Assessed Coding Practices

The coding practices assessed in this work were derived from attributes of the Buse and Weimer's [6] and Scalabrino's et al. [27] code readability models. In the following list, we enumerate the model metrics underlying our assessed coding practices:

(1) Buse and Weimer [6]:
  (a) Average blank lines;
  (b) Average keywords per line;
  (c) Average indents per line;
  (d) Maximum number of identifiers in a single line;
  (e) Average "." characters;
  (f) Maximum length of a single line;
  (g) Average "(" and "{" per line;
  (h) Average line length;

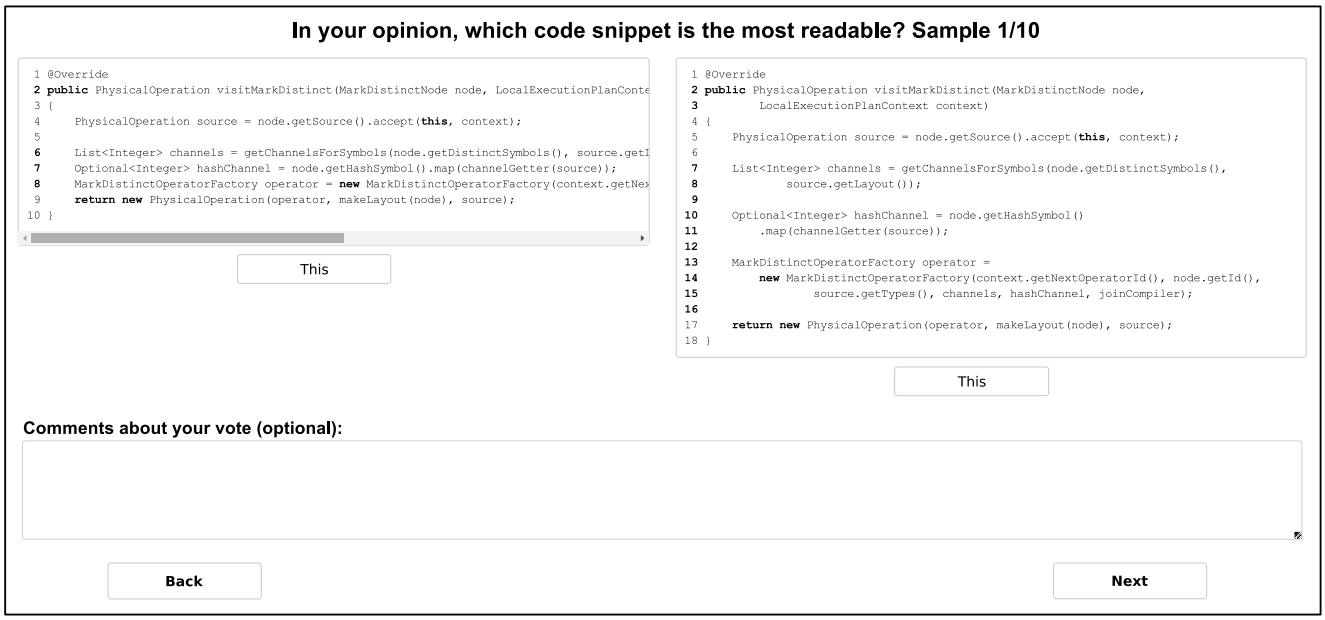

Figure 1: Most readable snippet selection screen. Participants were asked to pick the most readable snippet and provide comments about their decisions.

(i) Average number of identifiers;

(2) Scalabrino et al. [27]:

(a) Number of identifiers using dictionary words.

The following list presents the coding practices assessed in this paper. Every coding practice is identified by a code, in order to facilitate references throughout the text:

(P.1) There must be a blank line following code block opening and closing curly braces; **exception:** closing braces terminating code blocks of statements containing secondary paths of execution, e.g.: closing braces of `if` statements followed by an `else`; closing braces of `try` statements followed by a `catch`;

(P.2) Curly braces opening a code block must reside at the same line of their statements, in case there is one; closing braces must reside at their own lines;

(P.3) Blank lines must be used to create a vertical separation between related instructions;

(P.4) Line lengths must be kept within the limit of 80 characters;

(P.5) Indent markers must be made of 4 space characters;

(P.6) There should not be more than three levels of code block nesting;

(P.7) Avoid writing multiple statements separated by a ‘‘;’’ in a single line. Each statement should be in a line of its own;

(P.8) Avoid, whenever possible, using fully-qualified names to reference class names in code; use `import` clauses instead;

(P.9) Frequent calls to sub-properties of class member properties should be made storing a reference to that sub-property, avoiding multiple statements containing long chains of objects and sub-properties;

(P.10) Intermediary computations in long logical expressions should be stored in separate variables and subsequently composed in a single expression; avoid writing long chains of logical expressions;

(P.11) Identifier names should use dictionary words.

The source code files returned by the query described in Section 3.3 were employed to create code snippet pairs. For each one of the 11 assessed practices, a file was elected and a code section was extracted to build two code snippets: one of them adhering to the practice and the other, violating it. Upon the access of a new participant, the application randomly selected 10 snippet pairs and presented them, asking her to pick those snippets she thought were the most readable.

3.5 Data Analysis

In this section, we describe the methods of analysis employed to answer each of the proposed research questions. Throughout the description of the method and the presentation of the results, we use the notion of *diverging vote*, which consists in the vote given by participants on a code snippet *violating* the coding practice exercised by its code snippet pair.

3.5.1 Research Question RQ.1. In order to answer the research question RQ.1, we used a *two-tailed test for proportion* configured as follows:

(1) *Significance level*: 0.05;

(2) *Null hypothesis*: proportion of diverging votes equals to 0.5, i.e., votes equally distributed across adhering and violating snippets;
(3) *Alternative hypothesis*: proportion of diverging votes different from 0.5, indicating an effect induced by the coding practice.

We applied such test to each snippet pair individually. Being each snippet pair associated to one and only one coding practice, the results performed by a snippet pair work as a proxy of the effects induced by its associated coding practice. Proportions of diverging votes smaller than 0.5 indicate an enhancement on readability, while proportions greater than 0.5 indicate a decrease.

3.5.2 Research Question RQ.2. In order to answer research question RQ.2, a series of Fisher's Exact tests were performed. In those tests, votes on snippets pairs were considered as a categorical variable of two levels: "adhering" or "violating", depending on which snippet – respectively, the adhering or violating one – the vote was on.

The categorical vote variables were organized in records along categorical variables related to the profiles of their respective participants. The records were grouped practice-wise and, for each group, the *vote* variable was tested against all the other profile variables. The resulting p-values are presented in Table 2. Values below the significance level of 0.05 are highlighted.

3.6 Qualitative Analysis

As shown in Figure 1, there is a field for optional comments below each pair of snippets. Respondents were asked to comment about their decisions. In order to better understand practices that resulted inconclusive or conclusively worsened the perceived readability, we proceeded to a qualitative analysis of the content of comments. We considered comments provided together with diverging votes and looked for explanations about the reasoning and motivations underlying them.

4 RESULTS

Analyzing the received votes, we could note varying opinions about each coding practice. One categorical variable – *gender* – showed some correlation with opinions about practices P.7 and P.10. However, as it is discussed below, further analysis suggested that such correlations do not seem to indicate an important difference between genders' ways of reasoning about readability.[2]

4.1 Votes

Table 1 summarizes the votes on each code snippet pair. Code pairs are identified by the same code of their respective coding practices. Besides the percentage of diverging votes, there is a symbol indicating which kind of votes prevailed: the symbol ▲ indicates that code snippets *adhering* to the practice got the majority of the votes; the symbol ▼ indicates the prevalence of votes in the *violating* code snippets; and the symbol ♦ indicates a tie.

The total amount of votes per snippet pair varies due to two reasons: first, considering that the snippet pairs were showed randomly to participants, each snippet pair was exhibited a random number of times – even though every pair had the same chance of being exhibited. Second, voting in a snippet pair was optional and some pairs may have gotten fewer votes than their exhibitions.

Table 3: Frequencies of adhering/violating votes per gender on Practice P.7

	Female	Male
Adherent	2	33
Violating	5	4

Table 4: Frequencies of adhering/violating votes per gender on Practice P.10

	Female	Male
Adherent	4	9
Violating	1	27

4.2 Coding Practices and the Perceived Readability

Table 1 demonstrates the p-values for the statistical tests for proportions. All results below the significance level (p-value < 0.05) are highlighted. Coding practices P.3, P.5, P.6, and P.10 did not show enough evidence to reject the null hypothesis and, thus, there is no evidence of effects induced by those practices on the readability perceived by the participants.

Coding practices P.1, P.4, P.7, P.8, P.9, and P.11 pointed to an enhancement on the perceived readability. Practice P.2, on the other hand, showed a majority of violating votes and a p-value below the significance level. These results indicate that practice P.2 *decreased the levels of perceived readability.*

4.3 Developers' Profile and the Perceived Readability

As discussed in Section 3, a total of 77 Fisher's tests (11 practices $\times$ 7 categorical variables) were run in order to find possible correlations between votes and variables derived from the profiles provided by participants. The resulting p-values are listed in Table 2.

Most of the tested variables did not reach statistical significance. The only exception was the *gender* categorical variable in relation to the practices P.7 and P.10. Contingency tables showing the frequencies of votes by each gender for practices P.7 and P.10 are presented, respectively, in tables 3 and 4.

[2]Data gathered for this study, as well as the programs used to analyze them, are available online: https://doi.org/10.1145/3196321.3196342. Source code of the web application available upon request.

Table 1: Received Votes per Code Snippet Pair and P-Values of Tests for Proportions

Code Practice	Short Description	Adherent Votes	Violating Votes	% of Violating	Total Votes	Proportion Test P-Value
P.1	Blank following curly braces	45	7	13.46% ▲	52	**0.0000**
P.2	Opening braces at the same line as clause's	16	37	69.81% ▼	53	**0.0060**
P.3	Blanks separating related instructions	28	28	50.00% ♦	56	1.0000
P.4	Line lengths not exceeding 80 chars	44	15	25.42% ▲	59	**0.0003**
P.5	Indents as 4 spaces	21	29	58.00% ▼	50	0.3222
P.6	At most 3 levels of code block nesting	34	19	35.85% ▲	53	0.0545
P.7	Avoid mult. statements on a same line	48	10	17.24% ▲	58	**0.0000**
P.8	Avoid fully qualified names	52	6	10.34% ▲	58	**0.0000**
P.9	Avoid child properties; use references	45	9	16.67% ▲	54	**0.0000**
P.10	Variables to store intermediary logicals	21	34	61.82% ▼	55	0.1056
P.11	Names using dictionary words	42	12	22.22% ▲	54	**0.0001**

Table 2: P-Values for the series of Fisher's Exact Tests

	Factors						
Practice	Gender	Java Experience	Programming Experience	Age Bracket	Schooling	Learning Profile	Group (Student/ Professional)
P.01	0.2040	1.0000	0.3834	0.7274	1.0000	1.0000	0.1798
P.02	0.1520	0.3019	0.2010	0.3063	0.5117	0.5586	0.1550
P.03	0.0994	0.4909	0.0747	0.2109	0.6304	0.7385	0.1927
P.04	0.3470	1.0000	1.0000	0.8209	0.3643	1.0000	1.0000
P.05	0.6303	0.1714	0.6569	0.8539	0.6840	0.0538	0.6378
P.06	1.0000	1.0000	0.8713	0.7429	0.5042	0.0558	0.6117
P.07	**0.0020**	1.0000	0.6066	1.0000	0.7479	0.1125	0.5770
P.08	0.2476	1.0000	0.1438	0.3068	0.6730	1.0000	0.4970
P.09	1.0000	0.3082	1.0000	1.0000	0.7381	0.6037	0.5743
P.10	**0.0284**	0.5192	0.1290	0.1755	0.0513	0.1957	0.3588
P.11	0.1147	1.0000	0.1662	0.5475	0.7514	1.0000	0.0667

4.4 Developers' Comments

Participants were asked to comment their votes, as we presumed that such comments would allow us to understand the provided votes, specially in the cases of practices worsening readability.

In the following sections, we summarize comments that were provided together with diverging votes. Only comments from practices P.2, P.3, P.5, P.6, and P.10 are mentioned, once those were the practices presenting negative or inconclusive results.

4.4.1 Comments for Practice P.2. This practice led to a statistically significant decrease in readability. Curly braces in the violating snippet of the practice P.2 followed C# conventions, i.e., opening and curly braces in a line of their own. In general, respondents state appreciation for the extra spacing promoted by the use of C# convention, as put in comments like *"The code on the right uses white space and making clear blocks of code to signify what code is part of what conditions and such."* and *"The code seems to look slightly more readable, and grouping things like if clauses and arguments is more clear with a clear indication of a start and end."* Besides, they appreciate the vertical alignment of related curly braces, saying it is easier to see where a code block starts and ends, once matching curly braces are indented the same – e.g. *"Aligning curly braces allows for easy identification of enclosed statement blocks"*.

4.4.2 Comments for Practice P.3. This practice led to a tie, resulting as statistically inconclusive. The snippets were functionally equivalent, but with instructions in a slightly different order. Such reordering was not meant to exercise the coding practice: it was done only to allow the grouping of some related instructions. However, some comments show that such instruction reordering caught too much attention from some respondents (*"The other change at the bottom with the swapping of code makes no difference to me"*; *"some lines are change around"*; *"just the left one is ordered a little differently and has an extra blank line"*). This may have happened to other participants besides those who mentioned that in their comments. Among the 28 violating votes, only 7 comments were provided. Some affirmed seeing no advantage

in using blanks, and some even thought the blanks broke the code reading flow.

4.4.3 Comments for Practice P.5. The violating snippet used an indentation of 2 spaces, while the adhering one used 4 spaces. Such difference is barely noticed when comparing snippets side by side – with snippets presented vertically, such comparison would probably be easier. For some unknown reason, some developers understood that the adhering snippet, indented with 4 spaces, were actually using tabs instead of spaces, as mentioned in comments like *"the one I selected has less tabs"* and *"the right-side one looks more like a tab"*, and, based on that assumption, they voted on the violating snippet.

4.4.4 Comments for Practice P.6. From the 19 diverging votes, 8 were accompanied by comments. The violating snippet, containing deeper levels of `if` nesting and, as a consequence, more indenting, presented more white space. Interestingly, such increased white space caused by deeper indentation was appreciated by some of the participants who voted on the violating snippet, as it can be found in comments like *"Felt like my eyes had more space to navigate in between"* and *"The more white space is easier on my eyes"*. The other group of violating voters mentioned that, despite a deeper nesting, the violating code separated conditionals more clearly – they preferred a deeper nesting more than more complex conditionals: *"I prefer having a chain of nested "if" statements, rather than having an "if" statement condition so large that it has to be broken across multiple lines"* (even though such "large condition broken across multiple lines" was not the case in the adhering snippet) and *"It splits the conditionals into if else blocks rather than if, else if, and else. This makes it much easier to read and comprehend as it is in a true false format"*.

4.4.5 Comments for Practice P.10. Fourteen comments were provided. They basically criticize the fact that a new variable was created, but no gain resulted from that – *"There is no reason to create an extra boolean variable at all, it adds more clutter to the screen"* and *"Pulling out the boolean expression doesn't make it more readable for me"*. Some stated that they understood the motivation behind the creation of a new variable, intended to name an intermediary logical computation. But, according to them, the clarification provided by that additional name was not worth the effort and resources related to a new variable in memory – *"it does give insight into what it is checking is* `isOldSdk` *but by reading the if statement you can assume it is checking if one build version is greater than the other"*.

4.5 Discussion

Regarding the results of the tests for proportions, most of the practices resulted in a statistically significant increase in readability. On the other hand, only one resulted in a statistically significant decrease. These results are in agreement with the readability models that served as a foundation for the composition of our coding practices.

We found no evidence that either Java or general programming previous experience played a role, as none of the tests involving these categorical variables reached statistical significance. On the other hand, previous experiences with other similar languages, like C#, may have played some role, as suggested by the results of practice P.2 – the winner snippet followed C# conventions. The instructions presented at the beginning of the survey stated that the snippets were all written in Java, but it is plausible to expect that previous experience with C# may have influenced opinions. Previous experience with that language was not measured, though.

The results regarding gender look intriguing at first glance, but we think they are of lesser relevance, as the correlations do not seem to form a consistent pattern. From a code-compactness standpoint, both P.7 and P.10 recommend vertically "wider" code representations – the former, by recommending that each instruction be on a line of its own, and the latter, that logical expressions be broken in smaller, intermediary steps. Despite both practices point away from compactness, females were *more* often averse to practice P.7 and *less* often averse to P.10. Males, by their turn, showed an opposite trend.

Comments on practice P.2 cited the matching by indentation of curly braces as an important feature. However, in both samples, the contents of their code blocks were vertically aligned. This should have clearly established the boundaries of the given code blocks, despite the opening brace being misaligned in relation to its closing counterpart.

Some respondents commenting the practice P.4 saw a tab character on the adhering snippet. Some of them even pointed that such misunderstanding influenced their votes, as in *"I selected the one with less tabs"*. Four out of 14 comments mentioned this fact, and it is possible that more respondents thought this way. We inspected the snippets in order to check if they got rendered on screen with tabs due to some cause – perhaps a programming error in the routines that converted the snippets to their web representations, but that was not the case. The four spaces indentation probably simply resembled a tab character to those respondents, and they made their decisions based on that.

Features like *white spacing* seem to be under a complex dispute. Opinions of different developers about white spaces in a certain situation may be contradictory. Indeed, some comments of votes violating the practice P.6 argued they were based on the presence of more whitespace.

Opinions about whitespace gathered in this survey may be grouped in two categories: white space is positive, because it provides visual help to read the code, like in *"better in my opinion mostly due to spacing between lines"*; white space is negative because it actively hinders the code readability, like in *"(. . .) lines that deal with the same thing are separated by new lines (. . .)"*. These comments refer to the same snippet (P.3), suggesting that opinions may vary not only about the use of a given convention in diverse situations: *opinions about a given convention within a given circumstance may also strongly vary.*

Regarding the practice P.10, developers thought that the example was too simple and the practice was not worth the effort. This perhaps indicates that developers expect more flexible formatting rules in case of simple programs. Beside the violating – and most often preferred – snippet, there was an adhering snippet, naming an intermediary logical expression and providing a better context to the whole function performed by the code. It is possible that the presence of the adhering snippet and the explanation provided by it made the violating snippet look simpler, which, by its turn, may have led more respondents to classify this snippet as too simple to be worth the practice application.

5 THREATS TO VALIDITY

The source code files intended for the application of our coding practices were obtained by means of a query based on the predicates described in Section 3. However, the selection of code sections meant to illustrate the adherences and violations of the coding practices were made manually.

The comments and results obtained suggest there are some potential problems with the adopted approach. The presentation of a code snippets pair followed by a request for pointing the most readable invariably led the respondent to make a comparison between both snippets, which, as some comments reveal, confused them in cases where the difference between them was too subtle.

5.1 Internal Validity

For every coding practice, we selected a code section that provided opportunities for being reformatted in such a way that allowed us to illustrate violations and adherences of the referred coding practice. These code sections were manually selected, among the 40 source code files returned by the query discussed in Section 3.

The manual selection and reformatting of such source code files may have introduced errors capable of influencing respondents' opinions. For instance, by formating a code section intending to make it adhere to a particular coding practice, we may have introduced violations to other coding practices, negatively influencing opinions of some respondents.

The use of code static analysis tools could have attenuated such issue, scanning the resulting code for violations of a broad range of pre-programmed coding practices.

Interestingly, some comments suggest that developers carefully inspected both snippets within the pairs trying to spot differences between them. This, by its turn, suggests that decisions were not made based on how the code looked and felt, but, instead, on an objective classification of differences found. Such proceeding may have caused some distortions in votes on practices with snippets only subtly different. An A/B test exposing developers to only one of the versions (adhering or violating) could provide a rich set of data for further qualitative analyses.

By analyzing some comments, it seems that presenting violating and adhering snippets side by side may have influenced some judgments. Some developers voted against some practices generally stating that the coding practice did not promote a gain in readability proportional to its cost. This happened specially in practices P.10 and P.11, where some participants did not see the point of using more verbose coding on such self-explaining situations. It is possible that the presence of an adhering and more self-explaining code snippet may have fed respondents with valuable information. This way, possessing such information provided by the adhering snippet, they may have rated the practice as superfluous.

The examples used in the survey were simple. Such simplicity was intentionally promoted in order to make the survey feasible – overly complex examples could probably have discouraged some respondents. This was a significant risk, considering that an online survey may be abruptly abandoned as soon as the respondent feels bored. We acknowledge, though, that those simple snippets are not the best possible representatives of code bases commonly dealt with by developers during their daily routines.

5.2 External Validity

The survey respondents form a relatively restricted group of developers: students of an American university and professional programmers of a Brazilian company. Although they are representative of important groups of interest on software developers communities, they do not form a large and varied enough sample to allow the extension of our conclusions to more general groups of software developers.

6 CONCLUSIONS AND FUTURE WORK

The results allowed us to answer research question RQ.1, revealing that, in 8 out of 11 assessed coding practices, there is evidence that the coding practices affected readability as perceived by surveyed developers. In 3 out of 11 practices (P.3, P.5, and P.10), there was not enough evidence to allow us to refute the null hypothesis. Thus, we cannot affirm that these practices affected readability. Regarding research question RQ.2, except for the correlations between practices P.7 and P.10 and gender, we could find no evidence of a correlation between the studied profile factors and developer opinions. Besides, further analysis indicates that even these correlations do not represent a consistent difference between the way that females and males deal with the practices.

The dataset collected during this survey could be used to look for new correlations between code features and the readability levels perceived by our respondents, revealing unanticipated correlations and new perspectives over the data we already gathered [26].

Our group of respondents is mostly composed of students. The results presented here may contribute to the creation of coding samples used on classes, generating snippets and samples better suited to students' needs.

White spaces played an important role in this study. Many comments brought them up, even in conventions not directly related to them. Moreover, the practice P.3, regarding the

use of blank lines to visually group related statements, resulted in a tie. These results may provide an interesting path for further investigations, testing for some hypotheses about the reasons behind so heterogeneous opinions on blank lines and white spaces in general.

Being averse or sympathetic to a given convention may be, in some cases, just a matter of taste. As it has been shown, developers may have strongly opposite opinions about the same convention. Our results may be valuable for identifying such cases, helping developers tasked with the definition of a project's coding standard to decide whether they follow their own opinions or seek some advice.

REFERENCES

[1] 2016. Google C++ Style Guide. (2016). https://google.github.io/styleguide/cppguide.html

[2] Motor Industry Software Reliability Association et al. 2008. *MISRA-C: 2004: Guidelines for the Use of the C Language in Critical Systems*. MIRA.

[3] Christian Bird, Alex Gourley, Prem Devanbu, Anand Swaminathan, and Greta Hsu. 2007. Open borders? immigration in open source projects. In *Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on*. IEEE, 6–6.

[4] C. Boogerd and L. Moonen. 2008. Assessing the value of coding standards: An empirical study. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. 277–286. https://doi.org/10.1109/ICSM.2008.4658076

[5] Cathal Boogerd and Leon Moonen. 2009. Evaluating the relation between coding standard violations and faults within and across software versions. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 41–50.

[6] Raymond PL Buse and Westley R Weimer. 2010. Learning a metric for code readability. *Software Engineering, IEEE Transactions on* 36, 4 (2010), 546–558.

[7] Chromium Project. 2016. The Chromium Projects Coding Style. (2016). https://www.chromium.org/developers/coding-style

[8] Edgar Dale and Jeanne S Chall. 1948. A formula for predicting readability: Instructions. *Educational research bulletin* (1948), 37–54.

[9] Jonathan Dorn. 2012. A General Software Readability Model. *MSC Thesis avaiable from (http://www.cs.virginia.edu/weimer/students/dorn-mcs-paper.pdf)* (2012).

[10] William H DuBay. 2007. The Classic Readability Studies. *Online Submission* (2007).

[11] Xuefen Fang. 2001. Using a coding standard to improve program quality. In *Quality Software, 2001. Proceedings.Second Asia-Pacific Conference on*. 73–78. https://doi.org/10.1109/APAQS.2001.990004

[12] Rudolph Flesch. 1948. A new readability yardstick. *Journal of applied psychology* 32, 3 (1948), 221.

[13] GNOME Project. 2014. Gnome – C Coding Style. (2014). https://developer.gnome.org/programming-guidelines/stable/c-coding-style.html.en

[14] Les Hatton. 1995. *Safer C: Developing Software for in High-Integrity and safety-critical systems*. McGraw-Hill, Inc.

[15] Klaus Havelund and Al Niessner. 2014. Nasa JPL Java Coding Conventions. (2014).

[16] V.J. Hellendoorn, P.T. Devanbu, and A. Bacchelli. 2015. Will They Like This? Evaluating Code Contributions with Language Models. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. 157–167. https://doi.org/10.1109/MSR.2015.22

[17] Xiaosong Li. 2006. Using peer review to assess coding standards-a case study. In *Frontiers in education conference, 36th annual*. IEEE, 9–14.

[18] Xiaosong Li and Christine Prasad. 2005. Effectively teaching coding standards in programming. In *Proceedings of the 6th conference on Information technology education*. ACM, 239–244.

[19] Linux Foundation. 2016. Linux Kernel Coding Style. (2016). https://www.kernel.org/doc/Documentation/CodingStyle

[20] Bertha A.. Lively and Sydney Leavitt Pressey. 1923. *A method for measuring the" vocabulary burden" of textbooks*.

[21] Robert C. Martin. 2009. *Clean Code: A handbook of agile software craftsmanship*. Prentice Hall.

[22] Steve McConnell. 2004. *Code Complete, Second Edition*. Microsoft Press, Redmond, WA, USA.

[23] Microsoft Inc. 2016. C# Coding Conventions. (2016). https://msdn.microsoft.com/en-us/library/ff926074.aspx

[24] Vishal Midha, Rahul Singh, Prashant Palvia, and Nir Kshetri. 2010. Improving open source software maintenance. *Journal of Computer Information Systems* 50, 3 (2010), 81–90.

[25] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A simpler model of software readability. In *Proceedings of the 8th working conference on mining software repositories*. ACM, 73–82.

[26] Rodrigo Magalhães dos Santos and Marco Aurélio Gerosa. 2018. Impacts of Coding Practices on Readability - Dataset. (March 2018). https://doi.org/10.1145/3196321.3196342

[27] S. Scalabrino, M. Linares-Vsquez, D. Poshyvanyk, and R. Oliveto. 2016. Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 1–10. https://doi.org/10.1109/ICPC.2016.7503707

[28] Michael Smit, Barry Gergel, H James Hoover, and Eleni Stroulia. 2011. Maintainability and source code conventions: An analysis of open source projects. *University of Alberta, Department of Computing Science, Tech. Rep. TR11-06* (2011).

[29] Edward L Thorndike. 1921. The teacher's word book. (1921).