Readability and Understandability Scores for Snippet Assessment: an Exploratory Study

Carlos Eduardo C. Dantas
carloseduardodantas@iftm.edu.br
Federal University of Uberlândia
Brazil

Marcelo A. Maia
marcelo.maia@ufu.br
Federal University of Uberlândia
Brazil

ABSTRACT

Code search engines usually use readability feature to rank code snippets. There are several metrics to calculate this feature, but developers may have different perceptions about readability. Correlation between readability and understandability features has already been proposed, i.e., developers need to read and comprehend the code snippet syntax, but also understand the semantics. This work investigate scores for understandability and readability features, under the perspective of the possible subjective perception of code snippet comprehension. We find that code snippets with higher readability score has better comprehension than lower ones. The understandability score presents better comprehension in specific situations, e.g. *nested loops* or *if-else chains*. The developers also mentioned writability aspects as the principal characteristic to evaluate code snippets comprehension. These results provide insights for future works in code comprehension score optimization.

KEYWORDS

readability, understandability, code snippets, likert, code comprehension

1 INTRODUCTION

Code snippets (or code examples) are some lines of source code that can be reused to show how the developer can solve a specific programming task [9]. Developers often search for good reusable code snippets on the web [24]. In average, developers spend 70% of their time reading programs [14]. Some code search engines usually use readability metrics [21] [19] [3] trying to improve the code snippets ranking [8] [15]. These metrics have been employed in recent research, for instance to recommend readable APIs in code snippets [8] or to evaluate readability changes in projects history [18].

However, developers could have subjective perceptions of what means a readable code snippet. The readability metrics are often evaluated with personal opinions as response variable [16]. Consequently, these metrics could produce false positives/negatives. A potential opportunity to mitigate these mismatches in perception would be combining readability with other related features. Developers need to read and comprehend the code snippet syntax, but also need to understand the code snippet semantic, e.g., the *statements*, *beacons* or *motifs* [20]. If a source code is difficult to read, it is also difficult to understand [2]. Some metrics have also been proposed to calculate source code understandability [11] [12] [4]. For instance, the cognitive complexity metric of *SonarSource* tool[1] is related with some aspects of understandability [1].

The main goal in this research is to investigate readability and understandability metric scores on code snippets, to verify their usability on code snippet comprehension assessment.

We organize the investigation with the following research questions:

RQ #1) To what extent the readability and understandability metric scores can be used to code snippet assessment?

RQ #2) Which characteristics are important to developers on code snippets comprehension evaluation?

To evaluate the metric scores, we asked for five senior developers experienced in approve *pull requests* on *git* repositories (i.e., read, understand and evaluate source code produced by other developers) written in Java language, to evaluate the comprehension of two code snippets extracted from Google and CROKAGE for 30 input queries. A final open question was proposed for them to answer about the relevant code snippets characteristics in their evaluation.

The paper is organized as follows. Section 2 shows a motivating example. Section 3 discusses the related literature. Section 4 presents the study design proposed to collect data, modeling and data analysis approach. The results are reported and discussed in Section 5. Section 6 presents the threats that could affect the validity of this study. Finally, Section 7 summarizes our observations in lessons learned, and outlines directions for future work.

2 MOTIVATING EXAMPLE

A motivating example of subjective perceptions is shown in Figure 1. This example has the first code snippet suggested by Google, Microsoft Bing and CROKAGE [2] (tool that provides code snippets and their correspond comprehensive solution for each input query, both mined from Stack Overflow [5]) for the input query *Find maximum element of ArrayList in Java*. The Table 1 shows the readability [21] and understandability [4] score for each suggested code snippet. The CROKAGE code snippet has the lowest *LOC (lines of code*), but Google code snippet has more comments, and the *Collections.max()* method is implemented in a separate line. In CROKAGE, the line 13 has two concepts in the same line, which decreases the readability score in the used metric. The Microsoft Bing code snippet has highest *LOC*, and contains *for loop* statement instead the *Collections.max()* API call. In readability score, the CROKAGE code snippet is nearest to Microsoft Bing. But in understandability score, CROKAGE and Google has the same value, both better than Microsoft Bing code snippet.

The example shows the readability score could have divergent opinions, because some developers could prefer the lowest *LOC* instead of one concept per line. In this example, the understandability score has the trade-off between internal API call or a *for-loop*

[1] http://apisonar.com/

[2] http://isel.ufu.br:9000/

Google

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class ArrayListMaxValue {
    public static void main(String[] args) {
        // creating list.
        List<Integer> intValues = new ArrayList<>();

        // adding values
        intValues.add(6);
        intValues.add(64);
        intValues.add(45);
        intValues.add(60);
        intValues.add(9);
        intValues.add(2);

        // calling max() method.
        Integer max = Collections.max(intValues);
        System.out.println("ArrayList values : " + intValues);
        System.out.println("ArrayList max value : " + max);
    }
}
```

Microsoft Bing

```java
// Finding Maximum Element of Java ArrayList
import java.util.ArrayList;
import java.util.Collections;

class MinElementInArrayList {

    public static void main(String[] args)
    {
        // ArrayList of Numbers
        ArrayList<Integer> myList
            = new ArrayList<Integer>();

        // adding elements to Java ArrayList
        myList.add(16);
        myList.add(26);
        myList.add(3);
        myList.add(52);
        myList.add(70);
        myList.add(12);

        int maximum = myList.get(0);
        for (int i = 1; i < myList.size(); i++) {
            if (maximum < myList.get(i))
                maximum = myList.get(i);
        }
        System.out.println("Maximum Element in ArrayList = "
                           + maximum);
    }
}
```

Crokage

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public class MaxList {
    public static void main(String[] args) {
        List l = new ArrayList();
        l.add(1);
        l.add(2);
        l.add(3);
        l.add(4);
        l.add(5);
        System.out.println(Collections.max(l)); // 5
        System.out.println(Collections.min(l)); // 1
    }
}
```

Figure 1: Google, Microsoft Bing and Crokage code snippets returned for the input query *Find maximum element of ArrayList in Java*

statement, which could be less divergent than readability metric because generally snippets using internal APIs has less complexity and are reusable in other programs [15]. The readability and understandability metrics could complement each other on code comprehension.

Table 1: Readability and understandability (higher is better) scores for Google, Microsoft Bing and CROKAGE code snippets in Figure 1

Web Search Engine	Score	
	Readability	Understandability
Google	0.67	1.0
Microsoft Bing	0.51	0.8
CROKAGE	0.55	1.0

3 RELATED WORK

Several code search engines has been purposed to rank code snippets using the readability feature as part of the overall score. *Hora* [7] investigated how Google, a general-purpose web search engine rank the code snippets in terms of readability and reusability features. Their findings shows that readable and reusable code snippets are not necessarily top ranked, but other aspects as didactic code snippets or pages with multiple code snippets are more likely to be top ranked. Our research is not interested in discover how Google rank their code snippets, but to provide insights if readable and understandable code snippets are relevant for developers, and then other future researches could use these features as a part of overall score on new code search engines.

In other research, *Hora* [8] constructed the API *Sonar tool*, mining code snippets from 100 Java APIs on *github* to generate collections of API code snippets. He is also using readability to top rank readable API code snippets. The insights in our research could be useful to provide a better ranking, considering understandability in certain situations. *Moreno et al.* develop the *Muse* approach to rank code snippets producing an overall score using readability and reusability feature. But this research has employed other readability approach [3], and the other mentioned researches has used the *Scalabrino et al.* readability approach [20].

Another features could be used to produce an overall score. *Oliveira et al.* [16] introduced a separation between readability and legibility features, where legibility is related to how easy to identify elements in a program. For example, code without indentation or more than one statement in the same line contributes to decrease legibility. The readability tool used in this research consider some legibility aspects on their metrics, e.g., one concept per line.

Some researches studied the correlation between readability and understandability features. *Boehm et al.* [2] pointed the source code readability is related to its respective complexity and understandability, i.e., if the source code is difficult to read, it is also difficult to understand. But even *easy-to-read* source code can be difficult to understand, as presented by *Scalabrino et al.* [20]. Therefore, readability and understandability are employed as different features: while readability measures the effort to understand a code snippet in syntactic aspect, understandability measures complexity in dynamic aspect [19], i.e., both metrics complement each other in measure code comprehension.

The understandability feature has divergent results about metrics. *Scalabrino et al.* [20] made a study with 121 distinct metrics and found that none of them is relevant to measure understandability. However, *Marvin et al.* [1] found correlations between cognitive complexity metric [4] with subjective ratings of understandability, which is a relatively positive insight about the effectiveness of this metric. The cognitive complexity proposal is similar to the cyclomatic complexity of *McCabe* [13]. However, cognitive complexity aims to mitigate the limitations of cyclomatic complexity, such as source code nesting problem [22], and address modern language structures such as *try/catch* or *lambdas*.

4 STUDY DESIGN

The Figure 2 shows the overall approach to answer the two research questions. The major steps are: (1) Select Input Queries, (2) Extract Code Snippets, (3) Collect Metrics Values and (4) Developers Evaluation. The details of each step is in the following subsections. A replication package, including the readability and understandability tools, as the questions, code snippets characteristics, evaluations and the instructions for reproduction is available [6].

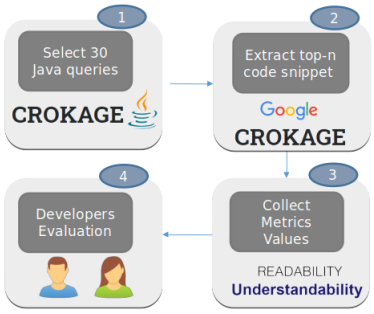


Figure 2: Overall architecture proposed in this research

4.1 Select Input Queries

CROKAGE have already collected 10,370 Java programming input queries performed from developers of more than 80 countries around the world. These queries contains users searches for code snippets implementing specific programming tasks in Java language. We selected 30 popular input queries performed by different users (i.e., distinct *IP address*) to conduct the experiment. To find the popular queries, we processed each query removing all punctuation symbols, stop words[3] and small queries (i.e., size lower than three). And then, we ordered the queries by the number of occurrences.

4.2 Extract Code Snippets

This step consists in extract two code snippets to each input query. These code snippets were extracted manually from Google (using private browsing on Google Chrome to avoid caches, user preferences) and CROKAGE. We collected the first code snippet that match each input query (e.g. assign the input query *"how to sort an array in java?"* has code snippets related to sort arrays and not *java.util.Set* or any other data structure), and contains reusable source code (e.g., we discarded code snippets containing references to unknown methods or variables) [4]. For each query, we added the tokens *"in java"*, to receive Java code snippets in the top ranking Google recommendations.

The Figure 3 shows the sites where code snippets were extracted, and their ranking in the web search. Using Google as web search, most of the snippets were in first result (top-1) and extracted in *geeksforgeeks.org*, *javatpoint.com* and *stackoverflow.com* sites. Using CROKAGE, all code snippets were extracted from *stackoverflow.com*, and 46.7% were the first result.

4.3 Collect Metrics Values

This step consists in extract the readability and understandability scores of each code snippet.

To measure readability, we use the prediction model proposed by *Scalabrino et al.* [21], which was used in other recent researches [8] [7] [18]. This model[5] includes a set of metrics including comments, identifiers consistency, textual coherence and number of meanings. The model produces scores between 0 (low readability) and 1 (high readability).

[3]https://bit.ly/1Nt4eMh
[4]https://tinyurl.com/jtjh75bx
[5]https://dibt.unimol.it/report/readability/

To measure understandability, we use the cognitive complexity code-based metric proposed by *Campbell* [4] and available in *SonarSource* tool. This metric were evaluated and employed in some past reaseaches [1] [23]. To measure understandability, we propose an adoption metric as follows:

$$understandability(cs_i) = \begin{cases} 1 - \frac{\#cc}{\#mcc} & \text{if } \#cc < 15 \\ 0.0 & \text{otherwise} \end{cases}$$

#cc is the complexity cognitive score extracted from *SonarSource* tool for the code snippet cs_i. The *#mcc* is the maximum recommend complexity cognitive value [6] , *#mcc* = 15. If a code snippet reaches *#cc* >= 15, the score output will be 0. This metric produces scores between 0 (low understandability) and 1 (high understandability).

4.4 Developers Evaluation

We invited five senior developers to analyse the comprehension of 60 code snippets. These developers are (5+ years) experienced as team leaders on Java projects, and they often evaluate *pull requests* submitted from other developers on their teams. *Pull requests* are usually rejected because could have issues in code comprehension or understanding [17].

For each of the 30 queries, the code snippets of each solution (Google and CROKAGE) were presented side by side, and asked for the developers to provide a *likert* value from 1 to 5 for the comprehension of each suggested code snippet. The code snippets for 30 questions are distributed on the following criteria:

(1) 10 questions with higher readability in one code snippet and similar understandability in both code snippets.
(2) 10 questions with higher understandability in one code snippet and similar readability in both code snippets.
(3) 10 questions with higher readability and understandability in one code snippet compared to the other.

The objective is to obtain the developers comprehension evaluation on each feature isolated, and both combined, to evaluate if the code snippets with better readability and understandability score are significant better evaluated than the lower ones. And finally, the developers answered in a open question the characteristics they included to evaluate each code snippet comprehension.

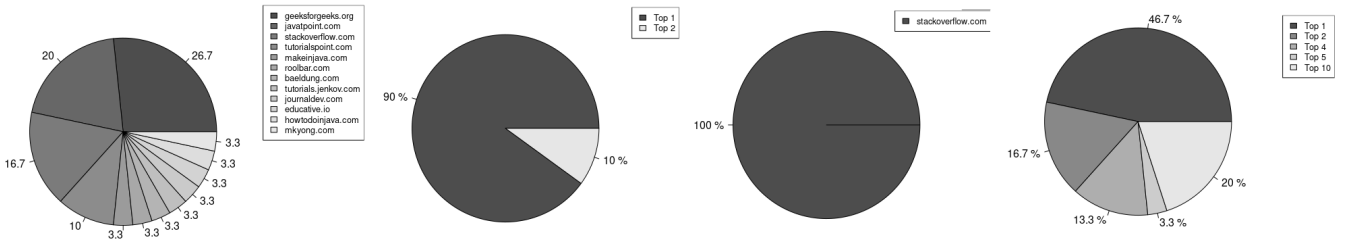
5 RESULTS AND FINDINGS

In this section, the results will be shown according to each research question. This research used Krippendorff's α reliability coefficient [10] to verify the agreement between the five developers. We obtained $\alpha = 0.334$, which is a low agreement. Only six of 30 code snippets with higher score had perfect agreement between the developers, as the same with one of 30 code snippets with lower score. The low agreement implies a subjective source code comprehension analysis.

RQ #1) To what extent the readability and understandability metric scores can be used to code snippet assessment

The Table 2 shows the likert evaluation results. We run Wilcoxon signed-rank test using a confidence level of 99% (p-value<0.01), and the comprehension of code snippets with higher readability score are statistically better than snippets with lower readability score. The Figures 4a and 4c shows a better rate for code snippets

[6]https://tinyurl.com/zfka2ew2



(a) Sites recommended by Google (b) Ranking Code Snippets extracted Google (c) Sites Recommended by CROKAGE (d) Ranking Code Snippets extracted CROKAGE

Figure 3: Sites and ranking of code snippets returned by Google and CROKAGE

Table 2: P-value and effect-size values for each feature

Feature	p-value	effect-size
Readability	< 0.01	0.857
Understandability	0.91	-
Readability + Understandability	< 0.01	0.838

with higher readability score. This result indicates the readability score match the developers perception on code comprehension. The Figure 4b shows statistically the same rate for higher and lower understandability score (as shown in Table 2), i.e., in this analysis, the understandability score is not relevant for the five developers on code snippets comprehension.

We investigate the understandability likert evaluation, and more significant likert differences were found in specific situations. For example, some code snippets with higher understandability code uses internal Java API methods to implement the task, and the code snippet with lower understandability score is using *nested loops* or *if-else chains*. However, the developers did not pointed difficult to comprehend few addictions of *loop*, *condition* or *try/catch block* on code snippets with lower understandability score. We extracted the highest understandability score differences between code snippets, and in this scenario, the effect size increases to 0.812 with *p-value = 0.06*, i.e., the understandability feature has better effect size in code snippets with high differences on understandability score. For example, on the input query *How to remove an element from an array in Java?*, the average likert for a code snippet with higher understandability score, using the *java.util.Stream.filter()* solution and *LOC* = 51 is 4.2, and the code snippet with *for nested loop* solution and *LOC* = 25 is 3.2. In another example, in the query *How to split a string in Java?*, the code snippet with lower understandability score has four extra lines with an additional *for statement* to print each splitted string. But the developers evaluate this code snippet as the best comprehend solution, i.e., the additional *for statement* had positive influence on their evaluation.

RQ #1 Answer: The readability score is associated with code snippets comprehension, but the low agreement between developers suggests subjective perception. The understandability score is related to code comprehension in specific situations, e.g., *nested loops* or *if/else chains*. These specific situations has higher difference on code snippets understandability score.

RQ #2) Which characteristics are important to developers on code snippets comprehension evaluation?

The Table 3 shows the mentioned characteristics on code snippets evaluation. Four of five developers (80%) mentioned writability

Table 3: Characteristics mentioned by the developers on code snippets evaluation

Characteristic	% mentions
Writability	80%
Variable and method aspects	60%
Comments	60%
Complexity	60%
Lines of Code (LOC)	20%
Performance	20%

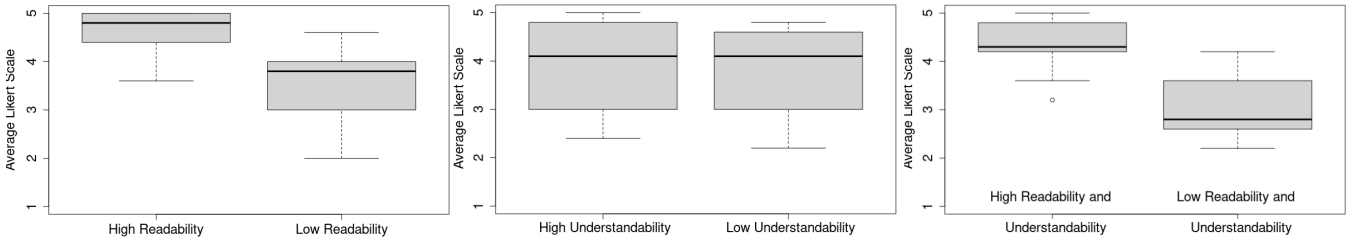
aspects, (e.g., simplicity and clarity on write code, expressiveness on self documented source code avoiding comment lines). Three of five developers (60%) mentioned about aspects of variables and methods (e.g., camel case pattern, easy naming comprehension, variable declarations aspects as default values). Also three of five developers (60%) mentioned comments, but all of them mentions that comments are only useful for source code with higher complexity. In their opinion, comments increases the number of lines, and may not be useful for code comprehension, giving a larger extension than the necessary. Other three of five developers (60%) mentioned complexity. Some *nested for loops* and *if else chains* were used in some code snippets instead of simple Java internal API calls. The reusability aspect was mentioned, i.e., more complex code snippets is more difficult to reuse in other software development project. One developer mentioned *LOC*, because in his opinion, fewer lines written in Java is generally easier to comprehend. Finally, one developer mentioned performance, i.e., if the solution is appropriate to run in production environment.

RQ #2 Answer: Some characteristics mentioned by developers are related to readability feature (variable and method aspects, comments, *LOC*) and understandability feature (complexity). However, most of the developers mentioned aspects of writability, which opens for new approaches investigating metrics for this characteristic.

6 THREATS TO VALIDITY

The main threat in this research is related to study generalization (programming language, number of participants, number of queries and code snippets).

Programming Language: the results are restricted to Java programming language, specially because limitations of the queries and the readability tool. The cognitive complexity tool supports more programming languages, e.g., Python, Javascript.



(a) Code Snippets comprehension for 10 queries with differences in Readability score

(b) Code Snippets comprehension for 10 queries with differences in Understandability score

(c) Code Snippets comprehension for 10 queries with differences in Readability and Understandability score

Figure 4: Box plots of Code Snippets comprehension

Number of participants: the five senior developers work in different companies (i.e., different core business and applications). We try to mitigate the few number of developers selecting team leaders with experience in evaluate and approve *pull requests* written by other developers. But novice developers could have different perceptions about readable and understandable code snippets.

Number of queries and code snippets: this research extracted 60 code snippets for 30 queries. A higher number of queries is an important factor to future research (e.g. produce an unified score using a combination of features).

False positives/negatives: the readability and understandability metrics could have some false positives. Complexity cognitive metric uses static heuristics. To minimize the effect, we manually analysed the score of each code snippet.

7 CONCLUSIONS AND FUTURE WORK

In this exploratory study, a quality analysis is conducted across code snippet comprehension using readability and understandability metric score. Our findings suggest the readability score could be used on code snippet assessment, e.g., code search engines. The understandability score have more subjective perceptions, specially in lower score differences between code snippets.

These results provide insights for several improvements. Future research could propose an empirical study to optimize a new unified score between understandability and readability features. Another code comprehension features could be evaluated, such as legibility and writability. Finally, a study with developers profiles could be addressed. Novice developers would have different perceptions about code comprehension than the team leaders used in this research.

REFERENCES

[1] Marvin Muñoz Barón, Marvin Wyrich, and Stefan Wagner. 2020. An Empirical Validation of Cognitive Complexity as a Measure of Source Code Understandability. *CoRR* abs/2007.12520 (2020). arXiv:2007.12520

[2] B. W. Boehm, J. R. Brown, and M. Lipow. 1976. Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering* (San Francisco, California, USA) *(ICSE '76)*. IEEE Computer Society Press, Washington, DC, USA, 592–605.

[3] Raymond P.L. Buse and Westley R. Weimer. 2010. Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering* 36, 4 (2010), 546–558.

[4] G. Ann Campbell. 2018. Cognitive Complexity — An Overview and Evaluation. In *2018 IEEE/ACM International Conference on Technical Debt (TechDebt)*. 57–58.

[5] Rodrigo Fernandes Gomes da Silva, Chanchal K. Roy, Mohammad Masudur Rahman, Kevin A. Schneider, Klérisson V. R. Paixão, Carlos Eduardo de Carvalho Dantas, and Marcelo de Almeida Maia. 2020. CROKAGE: effective solution recommendation for programming tasks by leveraging crowd knowledge. *Empir. Softw. Eng.* 25, 6 (2020), 4707–4758.

[6] Carlos Eduardo C. Dantas and Marcelo A. Maia. 2021. *Readability and Understandability Scores for SnippetAssessment: an Exploratory Study.* https://doi.org/10.5281/zenodo.5224346

[7] Andre Hora. 2021. Googling for Software Development: What Developers Search For and What They Find.

[8] André C. Hora. 2021. APISonar: Mining API usage examples. *Software: Practice and Experience* 51 (2021), 319 – 352.

[9] Iman Keivanloo, Juergen Rilling, and Ying Zou. 2014. Spotting Working Code Examples. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 664–675.

[10] K. Krippendorff. 2011. Computing Krippendorff's Alpha-Reliability.

[11] Jin-cherng Lin and Kuo-chiang Wu. 2006. A Model for Measuring Software Understandability. In *The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*. 192–192.

[12] Jin-Cherng Lin and Kuo-Chiang Wu. 2008. Evaluation of software understandability based on fuzzy matrix. In *2008 IEEE International Conference on Fuzzy Systems (IEEE World Congress on Computational Intelligence)*. 887–892.

[13] T.J. McCabe. 1976. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2, 4 (1976), 308–320.

[14] Roberto Minelli, Andrea Mocci and, and Michele Lanza. 2015. I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time *(ICPC '15)*. IEEE Press, 25–35.

[15] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How Can I Use This Method?. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1* (Florence, Italy) *(ICSE '15)*. IEEE Press, 880–890.

[16] Delano Oliveira, Reydne Bruno, Fernanda Madeiral, and Fernando Castor. 2020. Evaluating Code Readability and Legibility: An Examination of Human-centric Studies. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 348–359.

[17] Nick Papadakis, Ayan Patel, Tanay Gottigundala, Alexandra Garro, Xavier Graham, and Bruno da Silva. 2020. Why Did Your PR Get Rejected? Defining Guidelines for Avoiding PR Rejection in Open Source Projects. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 165–168.

[18] Valentina Piantadosi, Fabiana Fierro, Simone Scalabrino, Alexander Serebrenik, and Rocco Oliveto. 2020. How does code readability change during software evolution? *Empirical Software Engineering* 25 (11 2020), 1–39.

[19] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A simpler model of software readability. *Proceedings - International Conference on Software Engineering*, 73–82.

[20] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2017. Automatically assessing code understandability: How far are we?. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 417–427.

[21] Simone Scalabrino, Mario Linares-Vásquez, Rocco Oliveto, and Denys Poshyvanyk. 2018. A comprehensive model for code readability. *Journal of Software: Evolution and Process* 30 (06 2018).

[22] Mir Muhammd Suleman Sarwar, Sara Shahzad, and Ibrar Ahmad. 2013. Cyclomatic complexity: The nesting problem. In *Eighth International Conference on Digital Information Management (ICDIM 2013)*. 274–279.

[23] M. Wyrich, A. Preikschat, D. Graziotin, and S. Wagner. 2021. The Mind Is a Powerful Place: How Showing Code Comprehensibility Metrics Influences Code Understanding. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE Computer Society, Los Alamitos, CA, USA, 512–523.

[24] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What Do Developers Search for on the Web? 22, 6 (Dec. 2017), 3149–3185.