Assessing Consensus of Developers' Views on Code Readability

Agnia Sergeyuk, Olga Lvova, Sergey Titov,
Anastasiia Serova, Farid Bagirov, Timofey Bryksin
JetBrains Research
{agnia.sergeyuk, olga.lvova, sergey.titov}@jetbrains.com
{anastasiia.serova, farid.bagirov, timofey.bryksin}@jetbrains.com

Abstract

The rapid rise of Large Language Models (LLMs) has changed software development, with tools like Copilot, JetBrains AI Assistant, and others boosting developers' productivity. However, developers now spend more time reviewing code than writing it, highlighting the importance of Code Readability for code comprehension. Our previous research found that existing Code Readability models were inaccurate in representing developers' notions and revealed a low consensus among developers, highlighting a need for further investigations in this field.

Building on this, we surveyed 10 Java developers with similar coding experience to evaluate their consensus on Code Readability assessments and related aspects. We found significant agreement among developers on Code Readability evaluations and identified specific code aspects strongly correlated with Code Readability. Overall, our study sheds light on Code Readability within LLM contexts, offering insights into how these models can align with developers' perceptions of Code Readability, enhancing software development in the AI era.

1. INTRODUCTION

Large Language Models (LLMs) have seen rapid advancement, particularly in software development applications, where they serve as coding assistants and power tools like Copilot[1], JetBrains AI Assistant[2], Codeium[3], and others.

The evolution of AI-supported programming tools is reshaping software development practices — while AI enhances productivity, developers spend more time reviewing code than writing it (Mozannar, Bansal, Fourney, & Horvitz, 2023). Given that code comprehension time is generally related to Code Readability—the easier the code is to read, the less time it takes for the developer to comprehend it—optimizing the programmer's workflow involves providing suggestions from an LLM that align with developers' understanding of Code Readability.

In academia, Code Readability is defined as a subjective, mostly implicit human judgment of how easy the code is to understand (Posnett, Hindle, & Devanbu, 2011; Buse & Weimer, 2008; Scalabrino, Linares-Vasquez, Poshyvanyk, & Oliveto, 2016). However, aligning LLMs with developers' understanding of Code Readability necessitates an explication of developers' notion of what is readable code.

In our previous research (Sergeyuk et al., 2024), we studied if existing predictive models of Code Readability (Posnett et al., 2011; Scalabrino, Linares-Vásquez, Oliveto, & Poshyvanyk, 2018; Dorn, 2012; Mi, Hao, Ou, & Ma, 2022) may be a proxy of developers' Code Readability notion. This study, in addition to defining 12 Code Readability-related aspects obtained via the Repertory Grid Technique (Edwards, McDonald, & Michelle Young, 2009), revealed a weak correlation between current Code Readability models and developer evaluations, pointing to a significant gap in these models' ability to reflect developers' perspectives on Code Readability. It underscored the need for developing more accurate Code Readability metrics and models. We also found that developers' evaluations of Code Readability were not always consistently aligned with one another. We hypothesize that these results are dictated by the subjectivity of Code Readability and its aspects, along with other confounding variables.

[1]Copilot `https://github.com/features/copilot`
[2]JetBrains AI Assistant `https://plugins.jetbrains.com/plugin/22282-jetbrains-ai-assistant`
[3]Codeium `https://codeium.com/`

Therefore, we present a work that builds on top of the previous study, presenting the results of a survey we executed to delve deeper into developers' agreement level on Code Readability and its aspects.

We conducted a survey involving 10 Java developers from the same company, all with similar coding experience. Our aim was to assess whether a group of developers with similar backgrounds would reach a consensus on Code Readability assessments and related aspects, and which of those aspects are correlated the most with Code Readability. Each developer evaluated the same set of 30 Java code snippets using a 5-point Likert scale, rating code across 13 Code Readability-related dimensions.

The results of the study indicate a statistically significant intraclass correlation on Code Readability and several related metrics. This suggests that there is a degree of agreement among developers regarding what constitutes Code Readability. We also found a significant correlation of 12 Code Readability-related aspects evaluations with an assessment of Code Readability itself. It implies that LLMs could be tailored to Code Readability notion by adjusting metrics that are stable among developers and strongly related to Code Readability.

Overall, our work represents an approach to a deeper and at the time more explicit understanding of Code Readability concept among developers, which is instrumental in the present rapidly evolving AI-centered world of software development.

2. BACKGROUND

The development of code-fluent LLMs as coding assistants has fundamentally transformed the coding experience. Several research studies were conducted to examine how humans and AI interact in-depth and to understand how LLMs influence programmer behavior during coding activities, *e.g.,* (Mozannar et al., 2023; Liang, Yang, & Myers, 2023; Vaithilingam, Zhang, & Glassman, 2022; Barke, James, & Polikarpova, 2023).

A comprehensive study conducted by researchers from Cambridge and Microsoft informed the understanding of how developers interact with AI tools and how to improve this experience (Mozannar et al., 2023). Mozannar and colleagues studied the impact of GitHub Copilot on programmers' behavior during coding sessions. As a result, they identified 12 common programmer activities related to AI code completion systems. The researchers found that developers spend more time reviewing code than writing it. Indeed, approximately 50% of a programmer's coding time involved interactions with the model, with 35% dedicated to double-checking suggestions.

The investigation by Carnegie Mellon University researchers underscored the challenges encountered by developers while working with AI coding assistants (Liang et al., 2023). Their survey results suggest that developers mainly use these tools to save time, reduce keystrokes, and recall syntax. However, according to the same survey's results, the generated code is limited in meeting both functional and non-functional requirements. Additionally, developers struggle to comprehend the outputs of LLM due to the code being too long to read quickly.

Previous studies highlighted the importance of aligning Code Readability models with human notions, reducing coders' time and mental effort to comprehend AI coding assistants' suggestions. Developers need to quickly comprehend the code proposed by an AI coding assistant before integrating it into a project and implementing any changes. A critical aspect of this process is what is commonly referred to as Code Readability — the ease with which developers can read and understand code. In this notion, Code Readability forms a perceived barrier to comprehension that developers must overcome to efficiently work with code (Posnett et al., 2011; Buse & Weimer, 2008; Scalabrino et al., 2018).

Addressing developers' expectations regarding the readability of model-suggested code may involve various fine-tuning methods. Specifically, in addition to the fine-tuning process itself, when the developer modifies the model's weights and parameters, contrastive (Le-Khac, Healy, & Smeaton, 2020) and reinforcement (Lambert, Castricato, von Werra, & Havrilla, 2022) learning are valuable tools for this purpose. Implementing these methods encompasses the definition of a learning objective — informa-

tion about what output is "desirable" and what is not. Frequently, this objective is formed by annotating the models' outputs or by formulating rules that indicate users' satisfaction with the produced code.

Our previous research (Sergeyuk et al., 2024) explored the potential of existing state-of-the-art Code Readability models (Posnett et al., 2011; Dorn, 2012; Scalabrino et al., 2018, 2016; Mi, Keung, Xiao, Mensah, & Gao, 2018; Mi et al., 2022) to be learning objectives to guide the process of fine-tuning.

Posnett et al.'s Model. Posnett, Hindle, and Devanbu introduced a Simpler Model of Code Readability based on three features: Halstead volume, token entropy, and line count, surpassing the performance of Buse and Weimer's earlier model (Posnett et al., 2011). They employed forward stepwise refinement for feature selection, manually incorporating features driven by intuition and familiarity with Halstead's software metrics.

Dorn's Model. Dorn developed a General Software Readability Model, expanding beyond Java to include multiple programming languages (Dorn, 2012). Dorn's approach extended beyond syntactic analysis to include structural patterns, visual perception, alignment, and natural language elements, transformed into numerical vectors. This model also outperformed the retrained Buse and Weimer's model, emphasizing the value of using a wider range of code characteristics in readability assessments.

Scalabrino et al.'s Model. Scalabrino, Linares-Vasquez, and Oliveto proposed a Comprehensive Model integrating syntactic, visual, structural, and textual elements of code (Scalabrino et al., 2018). Their binary Code Readability classifier with 104 features surpassed previous models, emphasizing the benefit of textual alongside structural and syntactic features.

Mi et al.'s Model. Mi, Hao, Ou, and Ma introduced a deep-learning-based Code Readability model leveraging visual, semantic, and structural code representations (Mi et al., 2022). This model outperformed traditional machine learning models on a combined dataset, showcasing the potential of deep learning in automating Code Readability evaluation.

In our previous study, we utilized the Repertory Grid technique (Kelly, 2003) to establish a user-centric understanding of aspects related to Code Readability. This understanding served as a proxy for a unified perception of Code Readability among the developers who participated in the consequent survey. During this survey, they assessed code snippets on various readability-related aspects and provided an overall judgment on whether the presented snippet was readable or not. The data from the survey was then used to assess the agreement between the models described above and human evaluations of Code Readability. Overall, we found 12 readability-related bi-polar code aspects presented in Table 1.

Our research uncovered discrepancies in the correlation between existing Code Readability models and its human evaluations. While Scalabrino's model (Scalabrino et al., 2018) showed a moderate correlation with human assessments, other models like Posnett et al.'s (Posnett et al., 2011), Dorn's (Dorn, 2012), and Mi et al.'s (Mi et al., 2022) demonstrated weaker correlations. This variation suggests that using these Code Readability models as learning objectives to fine-tune code-fluent LLMs for improved readability might not be optimal. A more precise model is needed to guide this adjustment process.

Additionally, we found that developers assess Code Readability inconsistently, highlighting the need for more standardized and validated definitions of Code Readability.

To address these findings, our current study aims to investigate if confounding variables contributed to the previously observed inconsistency in Code Readability assessments by developers and if agreement on Code Readability is achievable. We also seek to identify stable aspects of Code Readability that could serve as a foundation for defining Code Readability and forming learning objectives for future LLM adjustments. Specifically, our research questions are as follows:

RQ 1. Do Java developers with similar backgrounds consistently assess Code Readability and its related aspects?

RQ 2. Do previously elicited code aspects represent Code Readability?

Readable pole	Unreadable pole
Code is concise	Code is too long
Code reads well from top to bottom	While reading, the eyes jump from top to bottom and back up again
∗Code is not sufficiently explained and needs additional info to understand what it does	Code is overexplained
The goal of the code is clear	The goal of the code is not clear
Code uses basic, known code patterns	Code looks unfamiliar, non-standard
Functionality is separated logically	Code needs refactoring
Code is flat and linear	Code is overly nested
There is one action per line of code	There are multiple actions on one line
Code uses named constants	Code uses “magic numbers”
Naming clarifies code functionality	Naming is confusing
Code conforms to style guides	Code is poorly formatted
There is balance in the color blocks	There are huge chunks of color blocks that stand out in a distracting way

∗This characteristic forms a continuum, being “Readable” in the middle and “Unreadable” at the extremes.

Table 1 – Code Readability Aspects

3. METHODOLOGY

3.1. Sample

The sample was gathered by sending the survey link to the internal channels of JetBrains with the invitation to Java programmers to participate in the study on Code Readability.

Based on preliminary power analysis, the sample was designed to consist of 10 Java programmers with varying experience levels. Despite their different experience levels, we assume that they share the same understanding of functional and non-functional requirements, as they have actively participated in developing a shared codebase.

All participants in our study are proficient Java developers. We assessed their experience using both subjective and objective measures. Six participants self-assessed as “Advanced”, indicating *extensive* experience and high proficiency in Java programming. Four participants self-assessed as “Intermediate”, signifying a *strong* understanding and ability to work on complex projects. The distribution of objective experience measures is presented in Table 2.

Experience	Frequency
More than 10 years	4
9–10 years	2
5–6 years	1
3–4 years	1
1–2 years	2

Table 2 – Distribution of Years of Experience

3.2. Materials

In the current survey, we employed materials gathered in the previous study (Sergeyuk et al., 2024) — a set of 30 AI-generated Java code snippets and a rating list of 12 Code Readability aspects. We justified the reuse of these materials to retest our previous approach and investigate if our data collection and analysis methodology might have influenced the earlier results on the agreement of code readability assessment by humans. Therefore, we maintained consistency by using the same approach and materials but with greater attention to confounding variables and data analysis.

The code snippets represented the readability of outputs generated by LLMs, which is important for us

due to the fact that the overarching goal of this research is to enhance the Human-AI Experience. To create snippets, we selected tasks from the Code Golf game[4] as prompts for ChatGPT 3.5 Turbo (due to the timing of the study) to generate Java language solutions for these tasks. Subsequently, we ensured that the snippets were meaningful and executable, adhering to a length limit of 50 lines, as defined by previous Code Readability models examined in prior research.

The primary aim of the list of Code Readability-related aspects (see Table 1), which we formulated from in-depth interviews using the Repertory Grid technique, was to offer consistent guidance to respondents during the evaluation of Code Readability. This aimed to ensure that developers assessed code uniformly and focused on key aspects related to readability.

3.3. Data Collection

On the greeting page of the survey, participants gave their consent and professional background information. After that, they were presented with a random sequence of the same set of 30 Java code snippets. Participants evaluated Code Readability of the snippet using the list of 12 bipolar characteristics and Code Readability itself with a five-point Likert scale measuring how much the code leans to the readable or unreadable pole. Participants could take breaks while completing the task, leading to completion times ranging from 40 minutes to 5 hours. Therefore, we believe that the random presentation of tasks and the flexible break schedule mitigated the effects of fatigue on the evaluations.

3.4. Data Analysis

To answer **RQ 1**, we calculated the intraclass correlation coefficient (ICC) to assess the agreement level of developers evaluating Code Readability and its aspects (Liljequist, Elfving, & Skavberg, 2019).

Additionally, to answer **RQ 2**, we calculated the Pearson's correlation coefficient (Cohen et al., 2009) of Code Readability-related aspects evaluations with overall Code Readability scores to see what metrics are related to Code Readability.

4. FINDINGS

RQ 1. Do Java developers with similar backgrounds consistently assess Code Readability and its related aspects?

The agreement level on assessments of Code Readability and related code aspects was found to be mostly from moderate to good (Koo & Li, 2016). The numerical values of ICC with corresponding Medians are presented in Table 3. The results of our study support the idea that developers of similar backgrounds would agree on evaluations of Code Readability and its related aspects.

Prior Code Readability studies show that human annotators exhibited imperfect agreement, with a correlation around .5 with the mean readability score (Buse & Weimer, 2008; Dorn, 2012). In our previous study (Sergeyuk et al., 2024), we did not find even this level of agreement. This discrepancy with the current results might be accounted for by the developers' shared backgrounds. In the current study, developers had similar backgrounds, all having experience consistently contributing to a specific codebase. Therefore, their views on some programming conventions are closed. In contrast, the developers in our previous study had a wide range of years of experience and worked at vastly different companies. Additionally, it might be the case that using Kripendorf's alpha with many missing values affected our previous findings, and that effect was mitigated by our data gathering this time. Namely, we avoid missing values in our study by presenting a fixed set of the same snippets to a fixed number of nonrandom raters and calculating ICC on that data.

Findings from the current study support the possibility of aligning LLMs' outputs with users' notions of readability. However, such alignment may be uniquely achievable within a specific company or among a group of developers with close views on various coding practices.

It is also noteworthy that not all Code Readability-related aspects have received a significant level of

[4]Code Golf game `https://code.golf/`

Code Aspect	Poles *(if represented numerically — from 2 to -2)*	ICC	Median
Readability	Readable / Unreadable	**0.78**	1
Code Structure	Functionality is separated logically / Code needs refactoring	**0.81**	1
Nesting	Code is flat and linear / Code is overly nested	**0.80**	2
Understandable Goal	The goal of the code is clear / The goal of the code is not clear	**0.79**	2
Code Length	Code is concise / Code is too long	**0.78**	2
Inline Actions	There is one action per line of code / There are multiple actions on one line	**0.76**	2
Reading Flow	Code reads well from top to bottom / While reading, the eyes jump from top to bottom and back up again	**0.75**	2
Sufficient Contextual Info	Code is not sufficiently explained and needs additional info to understand what it does / Code is overexplained	**0.74**	0
Code Style	Code conforms to style guides / Code is poorly formatted	**0.70**	1
Magic Numbers	Code uses named constants / Code uses "magic numbers"	**0.69**	0
Naming	Naming clarifies code functionality / Naming is confusing	**0.67**	1
Code Patterns	Code uses basic, known code patterns / Code looks unfamiliar, non-standard	**0.53**	2
Visual Organization	There is balance in the color blocks / There are huge chunks of color blocks that stand out in a distracting way	-0.03	0

Significant ICC values ($p < 0.05$) are highlighted in bold.

Table 3 – Agreement on Code Readability

agreement between developers. *Visual Organization* scale, which represents the balance between color blocks in the code snippets, *e.g.,* several lines of comments or big arrays, has a nonsignificant level of agreement. Having nonformal feedback from participants, we hypothesize that the wording and concept of this scale were unclear for developers and should be refined in future studies.

RQ 2. Do previously elicited code aspects represent Code Readability?

The results indicate that aspects of Code Readability correlate moderately to strongly with the Code Readability itself. We present a heatmap of statistically significant correlations in Figure 1.

Code aspects from our study, which we identified through in-depth interviews using the Repertory Grids Technique with developers, align with prior research and models of Code Readability. This alignment, along with the way these aspects were elicited, provides some grounds to hypothesize that they are indeed connected with Code Readability. Characteristics from our study resemble the combination of structural characteristics with visual, textual, and linguistic features as proposed by later Code Readability models (Dorn, 2012; Scalabrino et al., 2018; Mi et al., 2022). Moreover, Fakhoury et al. (Fakhoury, Roy, Hassan, & Arnaoudova, 2019) investigated commits that were explicitly aimed at Code Readability-enhancement and observed notable changes in *Complexity, Documentation, and Size* metrics that resemble *Code Structure, Nesting, Sufficient Contextual Info, and Code Length* metrics from our list. In the study of Fakhoury et al., it was also noted that *Code Style and Magic Numbers Usage* are the aspects where improvements in Code Readability-related commits are prominent. In another study, Peitek et al. (Peitek, Apel, Parnin, Brechmann, & Siegmund, 2021) examined 41 complexity metrics and their influence on program comprehension, discovering that factors such as *Textual Length and Vocabulary Size* increase cognitive load and working memory demand for programmers.

Further evidence supporting the idea that the code aspects we elicited in our previous study represent developers' notion of Code Readability is the statistically significant correlation between the entire list of 12 Code Readability-related aspects and evaluations of Code Readability itself. However, there are some differences in the strength of these correlations. The strongest correlation of Code Readability evaluation is with Naming, Code Length, Understandable Goal, and Reading Flow metrics. Combined

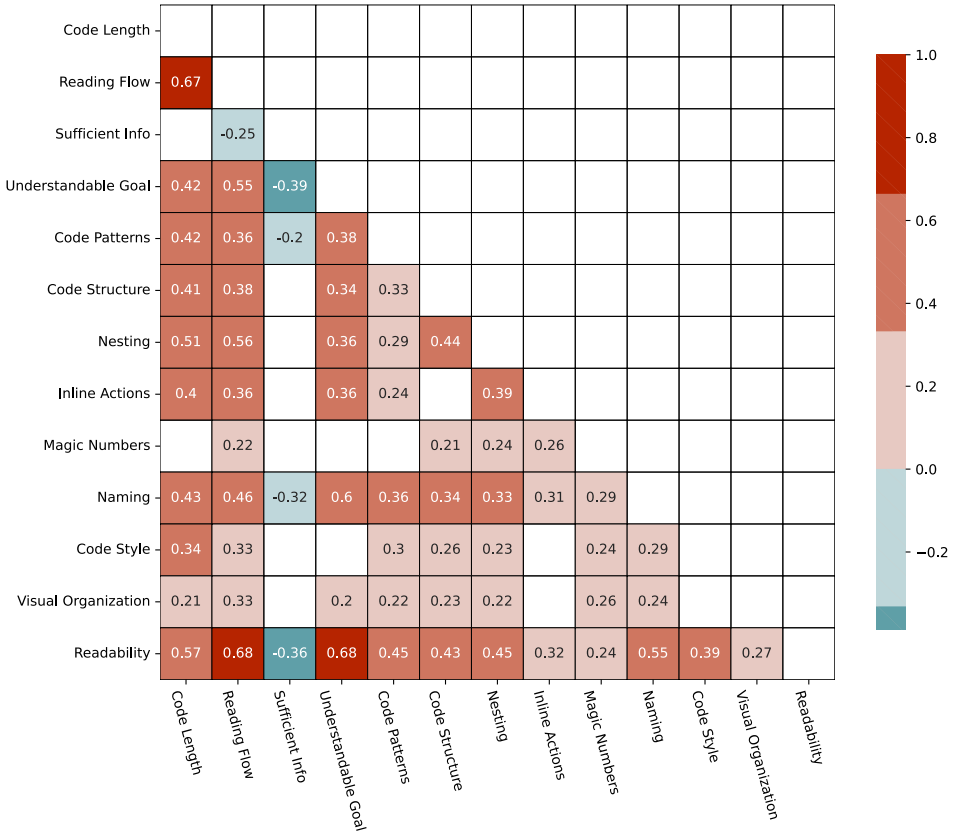

Figure 1 – Correlations of Code Readability-related aspects

with the fact that Code Length and Understandable Goal are also metrics that gained a good level of agreement among developers who assessed snippets, we can hypothesize that these two code aspects are most representative of Code Readability and could be used as guidance for LLMs alignment.

5. CONCLUSION AND FUTURE WORK

This study explored the possibility of agreement among developers on Code Readability evaluations, with the aim of potentially utilizing Code Readability as a learning objective for LLMs. Our findings indicate that developers with similar professional backgrounds tend to exhibit a good level of agreement in Code Readability evaluations. Additionally, certain code aspects related to Code Readability, *i.e., Code Length* and *Understandable Goal*, demonstrate promising potential as representatives of the key scales influencing Code Readability.

With this supporting evidence in hand, our future endeavors will focus on further exploration of Code Readability aspects and their potential representations for LLM adjustment with the overarching objective of enhancing user experience with AI assistants in programming.

6. References

Barke, S., James, M. B., & Polikarpova, N. (2023, apr). Grounded copilot: How programmers interact with code-generating models. *Proc. ACM Program. Lang.*, *7*(OOPSLA1).

Buse, R. P., & Weimer, W. R. (2008). A metric for software readability. In *Proceedings of the 2008 international symposium on software testing and analysis* (p. 121–130). Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1390630.1390647` doi: 10.1145/1390630.1390647

Cohen, I., Huang, Y., Chen, J., Benesty, J., Benesty, J., Chen, J., ... Cohen, I. (2009). Pearson correlation coefficient. *Noise reduction in speech processing*, 1–4.

Dorn, J. (2012). *A general software readability model.* Retrieved from http://www.cs.virginia.edu/weimer/students/dorn-mcs-paper.pdf

Edwards, H. M., McDonald, S., & Michelle Young, S. (2009). The repertory grid technique: Its place in empirical software engineering research. *Information and Software Technology*, *51*(4), 785-798. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950584908001298 doi: https://doi.org/10.1016/j.infsof.2008.08.008

Fakhoury, S., Roy, D., Hassan, A., & Arnaoudova, V. (2019). Improving source code readability: Theory and practice. In *Proceedings of the 27th international conference on program comprehension* (p. 2-12). IEEE. Retrieved from https://doi.org/10.1109/ICPC.2019.00014 doi: 10.1109/ICPC.2019.00014

Kelly, G. (2003). *The psychology of personal constructs: Volume two: Clinical diagnosis and psychotherapy*. Routledge.

Koo, T. K., & Li, M. Y. (2016). A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of chiropractic medicine*, *15*(2), 155–163.

Lambert, N., Castricato, L., von Werra, L., & Havrilla, A. (2022). Illustrating reinforcement learning from human feedback (rlhf). *Hugging Face Blog*. (https://huggingface.co/blog/rlhf)

Le-Khac, P. H., Healy, G., & Smeaton, A. F. (2020). Contrastive representation learning: A framework and review. *IEEE Access*, *8*, 193907-193934. doi: 10.1109/ACCESS.2020.3031549

Liang, J. T., Yang, C., & Myers, B. A. (2023). *Understanding the usability of ai programming assistants.*

Liljequist, D., Elfving, B., & Skavberg, K. (2019). Intraclass correlation–a discussion and demonstration of basic features. *PloS one*, *14*(7), e0219854.

Mi, Q., Hao, Y., Ou, L., & Ma, W. (2022). Towards using visual, semantic and structural features to improve code readability classification. *Journal of Systems and Software*, *193*(C). Retrieved from https://doi.org/10.1016/j.jss.2022.111454 doi: 10.1016/j.jss.2022.111454

Mi, Q., Keung, J., Xiao, Y., Mensah, S., & Gao, Y. (2018). Improving code readability classification using convolutional neural networks. *Information and Software Technology*, *104*, 60-71. Retrieved from https://www.sciencedirect.com/science/article/pii/S0950584918301496 doi: https://doi.org/10.1016/j.infsof.2018.07.006

Mozannar, H., Bansal, G., Fourney, A., & Horvitz, E. (2023). *Reading between the lines: Modeling user behavior and costs in ai-assisted programming.*

Peitek, N., Apel, S., Parnin, C., Brechmann, A., & Siegmund, J. (2021). Program comprehension and code complexity metrics: An fmri study. In *2021 ieee/acm 43rd international conference on software engineering (icse)* (p. 524-536). doi: 10.1109/ICSE43902.2021.00056

Posnett, D., Hindle, A., & Devanbu, P. (2011). A simpler model of software readability. In *Proceedings of the 8th working conference on mining software repositories* (p. 73–82). Association for Computing Machinery. Retrieved from https://doi.org/10.1145/1985441.1985454 doi: 10.1145/1985441.1985454

Scalabrino, S., Linares-Vásquez, M., Oliveto, R., & Poshyvanyk, D. (2018). A comprehensive model for code readability. *Journal of Software: Evolution and Process*, *30*(6), e1958. Retrieved from https://doi.org/10.1002/smr.1958 doi: 10.1002/smr.1958

Scalabrino, S., Linares-Vasquez, M., Poshyvanyk, D., & Oliveto, R. (2016). Improving code readability models with textual features. In *2016 ieee 24th international conference on program comprehension (icpc)* (p. 1-10). IEEE. doi: 10.1109/ICPC.2016.7503707

Sergeyuk, A., Lvova, O., Titov, S., Serova, A., Bagirov, F., Kirillova, E., & Bryksin, T. (2024). *Reassessing java code readability models with a human-centered approach.*

Vaithilingam, P., Zhang, T., & Glassman, E. L. (2022). Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Extended abstracts of the 2022 chi conference on human factors in computing systems* (p. 1-7). Association for Computing Machinery. Retrieved from https://doi.org/10.1145/3491101.3519665 doi: 10.1145/3491101.3519665