Towards Understanding Code Readability and Its Impact on Design Quality

Umme Ayda Mannan
Oregon State University
mannanu@oregonstate.edu

Iftekhar Ahmed
University of California, Irvine
iftekha@uci.edu

Anita Sarma
Oregon State University
anita.sarma@oregonstate.edu

ABSTRACT

Readability of code is commonly believed to impact the overall quality of software. Poor readability not only hinders developers from understanding what the code is doing but also can cause developers to make sub-optimal changes and introduce bugs. Developers also recognize this risk and state readability among their top information needs. Researchers have modeled readability scores. However, thus far, no one has investigated how readability evolves over time and how that impacts design quality of software. We perform a large scale study of 49 open source Java projects, spanning 8296 commits and 1766 files. We find that readability is high in open source projects and does not fluctuate over project's lifetime unlike design quality of a project. Also readability has a non-significant correlation of 0.151 (Kendall's τ) with code smell count (indicator of design quality). Since current readability measure is unable to capture the increased difficulty in reading code due to the degraded design quality, our results hint towards the need of a better measurement and modeling of code readability.

CCS CONCEPTS

• **Software and its engineering** → *Software reliability*; *Software maintenance tools*; *Software design tradeoffs*;

KEYWORDS

Readability, Design quality, Code smell

ACM Reference Format:
Umme Ayda Mannan, Iftekhar Ahmed, and Anita Sarma. 2018. Towards Understanding Code Readability and Its Impact on Design Quality. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering (NL4SE '18), November 4, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3283812.3283820

1 INTRODUCTION

Software development is not an individualistic activity; it is a team effort. In order to make changes, developers need to understand their own, and their peers' code. This implies that the code needs to be well written and readable. This is especially relevant to Open Source Software (OSS) projects, since new contributors to the project might not have enough experience, might be unfamiliar with the existing code base, and they might be reluctant to ask other developers for clarifications. It comes to us as no surprise that developers have identified readability-understandability as one of their top three information needs [11].

Code that is difficult to read can have wide ranging consequences. It can lead to changes being sub-optimal or changes that introduce potential bugs. It is well established that maintenance of a code base becomes difficult as the project evolves and the team size grows [8]. Researchers have looked at the impact of readability. Beacker et al. [7] looked at the link between readability and understandability. Researchers also identified that comments and full word identifiers play an important part in understanding the code [14, 27, 28]. However, current studies have not investigated the evolution of readability and it's impact on design quality. Researchers have shown that as a project grows older, it's design quality degrades and technical debt accumulates [6]. Intuitively degradation in design quality should have association with readability because code smells make it difficult to understand the code by increasing duplication, obfuscation etc. [6] Moreover code smells makes maintenance [18] difficult as minor alterations require complex edits to the source code which may have unintended side effects and ultimately lead to bugs [25].

We conducted our empirical study to answer questions such as: Can code readability metric reliably measure the evaluation of source code? Do they increase, or decrease, over time? Do they have an impact on the overall design quality? Without answering these questions researchers cannot create models or tools that reflect the right levels of readability status of a code base.

In this paper, we answer the following research questions:

- RQ1: What is the average readability scores of open source projects?
- RQ2: How does readability evolve over the lifetime of a project?
- RQ3: Does readability impact design quality of a project?

2 RELATED WORK

Researchers have been looking at ways of measuring readability and its impact on software quality. Buse et al. [9] developed the first model to measure software readability. They defined readability as "a human judgment of how easy a text is to understand" Based on their survey data, Posnett et al. [32] developed an improved readability model using a smaller set of features. Their features focus on the textual complexity or entropy of the source code. Butler et al. [12] investigated the relationship between code quality and readability. Buse et al. [9] also identified the relationship between code readability, defects and code churn. Aggarwal et al. [5] showed that readability impacts maintainability. Readability also impacts the utility of test cases. Daka et al. [13] found that unreadable tests are difficult to maintain and lose value to developers. They showed that using readability to augment automated unit test generation can help the quality of tests.

Researchers have been also investigating code smells which are symptoms of poor design and implementation choices [18] and their affect on maintainability of software [26]. Researchers have shown association between code smells and bugs [29, 30], and code maintainability problems [18]. Zazworka et al. [36] found that the code smell like *God Class* is related to technical debt. Ahmed et al. [6] found that ignoring the smells leads to "software decay". Code smells have been also associated with fault-proneness [19, 25].

Readability is not only an academic research topic, in fact Buse et al. [11] found that developers rank readability of code as one of their top information needs. On the other hand, Yamashita et al. [34] found that a considerable portion (32%) of developers are not aware of code smells. Both readability and code smells impact maintainability, has association with bugginess, so we posit that there is association between these two. To the best of our knowledge no one has investigated the relationship between code smells and readability of a code and our work sheds light on this association.

3 METHODOLOGY

In this study, our goal is to see the average readability of open source projects, how they evolve over time and how that impact software design quality. In the following paragraphs we discuss the various steps we took to collect and analyze the data.

Project Selection Criteria: We selected active open source Java projects from GitHub. We chose Java because it is the most popular programming language [33], and there are more code smell detection tools for Java than other languages [16]. Initially, we randomly selected 200 projects by using the GitHub search mechanism. From these, we eliminate projects that were too small, that is, having fewer than 10 files, or fewer than 500 lines of code. These filters were essential to ensure that the projects we analyzed are representative for real-world projects, and are not throw-away code or class projects, so we followed the guidelines proposed by Kalliamvakou et al. [24]. This left us with 120 projects. Due to time constraints we randomly selected 49 projects out of the remaining 120 for this study. Table 1 provides a summary of features and other descriptive information of our selected 49 projects.

Table 1: Project characteristics

Dimension	Max	Min	Average	Std. dev.
Line count	116,238	534	5,837	14,511.73
Duration (weeks)	350	10	41.37	43.18
# of Developers	105	4	10.78	11.04

Measuring readability score: For our study first we calculate the readability score for each method of our 49 projects using the state of art model proposed by Posnett et al. [32]. They propose the z metric, which uses the following formula:

$$z = 8.88 - 0.033V + 0.40Lines - 1.5Entropy \quad (1)$$

We collect the *Lines* metric using the Cloc [1] tool.
We use the Halstead metrics [20] to calculate the Volume, $V = N \log_2 n$, where the length N is the sum of total number of operators ($N1$) and operands ($N2$). The Program Vocabulary (n) is the sum of the number of unique operators ($n1$) and unique operands ($n2$). *Entropy* for each method is calculated using the Python *Entropy* library [2]. *Entropy*, $H(X)$, is calculated using the following formula:

$$H(X) = \sum_{i=1}^{n} P(x_i) \log_2 P(x_i) \quad (2)$$

where X is a document and x_i is a term in X. $P(x_i)$ is the probability of a change occurring in a document. n is the total number of documents, in our case, Java source files. We then use the logit function $\frac{1}{1+e^{-z}}$ to calculate the actual readability score. This normalizes the results to be in the $[0, 1]$ range. After calculating the readability score of each method in the file, we use the median readability of all methods as the readability measure for the whole file. We chose this metric, as opposed to simply adding the readability metrics, because concatenating multiple readable methods should not have a negative impact on the overall readability of a file.

Measuring the evolution of readability: Our aim is to see how readability evolve over time. Previous studies used different ways of partitioning time. Izurieta et al. [22, 23] used releases as the unit of time, others individual commits, or discrete time units (years, months, weeks, days etc.). Though all of these approaches should lead to similar findings, the "resolution" may be different. Furthermore, none of these approaches lead to a true comparison across projects. Projects work at different phases. Projects are of different size, maturity level, and follow different release cycles and policies. Individual commits are the only "level" measure, but would be too fine grained for our purpose. We therefore selected the week as our unit of measure because, while subject to some variation from project to project, did give us fine-grained insight into the evolution of projects.

Measuring Code Smells: To perform code smell analysis for the selected projects, we selected inFusion [3] which detects a wide range of 18 different code smells and have been validated [6] and used by many other studies [15, 17, 21].

Data Analysis: To answer our third research question, we measure correlation using Kendall's τ between total code smell count of a file and readability score. We also perform a Wilcoxon rank sum test to test if there is difference in the mean readability score between smelly code and non-smelly code.

4 RESULTS

In the following section, we present our results structured around our three research questions.

RQ1: What is the average readability scores of open source projects?

To answer our first research question, we collected the readability scores from 1766 files spanning 49 projects. From Figure 1, we see that the readability score of all files are higher than 0.95 on a scale of 0 to 1. In our data set, the mean readability score is 0.99 with a standard deviation of 0.04. This indicates that, in general, OSS projects have excellent readability measured using Equation 1 .

Observation 1: The OSS projects have high *readability* scores.

RQ2: How does readability evolve over the lifetime of a project?

Prior research has shown that, as the code base becomes larger and more people contribute, the project quality degrades. It is common belief that readability impact the code quality. We hypothesize that readability would also show some sort of degradation trend

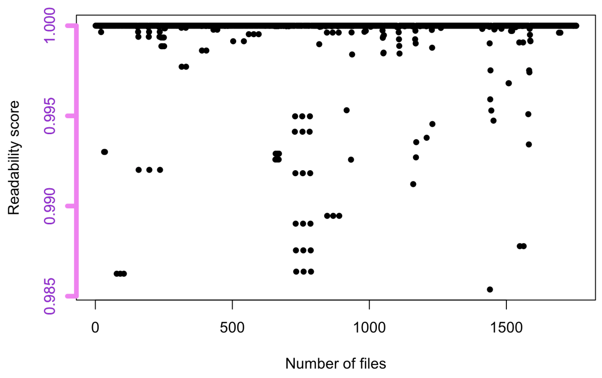

Figure 1: Median Readability Score

over time. To test this hypothesis, we perform a trend analysis of readability scores over a 350 week period. From Figure 2 we see that, though there is a small amount of fluctuation, eventually the readability score remains steady over time. The shaded band is the 95% confidence interval indicating that the true regression line lies within the shaded region. We know that the code quality measured in terms of design quality degrades over time [6]. Once we saw that readability stay at a constant level throughout a projects lifetime, we hypothesize that the correlation between code smell and readability is very low. To check our hypothesis, we calculated the correlation between code smell and readability.

Observation 2: The OSS projects maintain their high *readability* scores over time.

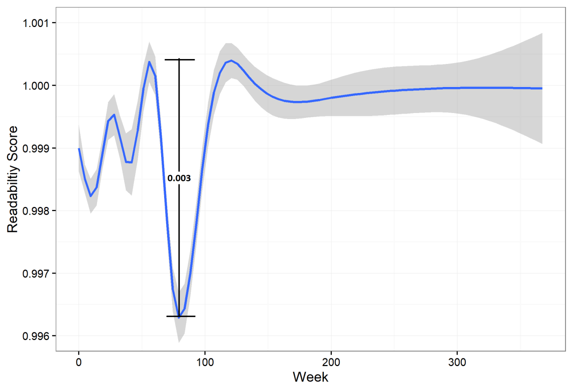

Figure 2: Readability Score evolution over time. (Note: the readability never exceeds 1. The line temporarily exceeding 1 is an artifact of the smoothing algorithm.)

RQ3: Does readability impact design quality of a project?

To answer our third research question, we check the correlation between count of code smells and readability and find that readability has a statistically non-significant correlation of 0.151 with code smell count (Kendall's τ correlation coefficient test, $p < 0.13$).

Observation 3: *Readability* scores do not correlate with design quality, measured by code smells.

We also compare the mean readability score of program elements that have code smells with the mean readability score of the program elements that do not have code smells. The mean readability score was 0.991 for program elements with code smell and 0.998 for program elements without code smells. The difference in the mean readability score between these two groups is statistically significant (Wilcoxon rank sum test, W = 31 737, $p < 1.879 \times 10^{-7}$). Therefore, we find that program elements that have code smells are, on average, less readable than entities that do not have a code smell. However, when we checked the effect size using Cohen's D test using the `effsize` R library [4], we found that the d estimate was 0.098 and was negligible.

Observation 4: *Readability* of files with code smells is smaller compared to files without code smells; However, the effect size is negligible.

5 DISCUSSION

Prior research on design quality [6] has shown that projects degrade over time, and they accumulate design debt. One might assume that lack of readability contributes to this degradation, even if only partially. This is intuitive since the lack of readability should impede a developer from correctly comprehending the different aspects of a program, which in turn would limit their knowledge of the dependencies in the code, and the potential impact of the changes they are performing. The lack of awareness regarding the impact of changes have been associated with worsening of design quality [35].

However, our analysis reveals almost perfect readability scores across all projects in our data set. Moreover, we find that these projects start with high readability scores and maintain these scores, despite small fluctuations. One possible reason behind such good readability scores is, since OSS projects rely on volunteer contributions and, therefore, need to keep their source code readable. However, it is strange that these projects keep such good readability scores despite receiving contributions from many developers, mostly volunteers with varying levels of experience and project knowledge. Another reason for having high readability scores could be that the code base is public, and developers make extra effort to keep the code readable. Or, it might be that if there are readability issues, any contributor can fix these issues (the "many eyeballs" effect). We speculate that the widespread use of different style guides and code review processes that are required when contributing source code are forcing developers to write highly readable code.

We also found that readability scores are not correlated with the code smell count. Based on our findings, we can speculate that current code readability measurement is not an effective metric when design quality is a concern. Also, it is possible that the metrics used by researchers [9, 32] to measure readability is not appropriate for large projects. The metric we use was proposed by Posnett et al. [32], which was evaluated on small programs; so it might have limited generalizability. Further research on large scale corpus is needed to identify better metrics for modeling readability scores.

Once better models are identified, additional research that investigates the impact of these metrics on code quality is needed. Using the current models we do not find any significant impact of readability on code quality. However, with better measurement metrics and models, we could see more significant impacts; Perhaps even finding thresholds beyond which readability scores severely impact quality, which developers can use to generate warnings. Other kinds of support, for example dashboards or visualizations, can then display the "health" metrics of the code base, including readability scores to developers and managers. Recall both developers and managers had noted readability-understandability to be their top-3 information needs [10].

6 THREATS TO VALIDITY

Our samples have been from a single project host site, Github. This may be a source of bias, and our findings may be limited to open source projects from Github. However, we believe that the large number of projects sampled more than adequately addresses this concern. Since we are relying on inFusion to detect smells, the accuracy of our results depends on the accuracy of this tool. The efficacy of it's threshold-based detection strategies has been evaluated in a previous study [6]. InFusion uses static program analysis to identify smells, and research [31] shows that "intrinsically historical" code smells such as Parallel Inheritance are difficult to detect solely through static analysis. The number of such smells might be different when historical information is taken into account.

7 CONCLUSIONS

Developers need to read and understand code in order to maintain it effectively, therefore making readability a key information need [11]. We studied the efficacy of current readability model in measuring design quality across a large set of projects. Our results show that projects have well maintained high readability scores. Moreover, surprisingly readability is not associated with code smells, which are perceived as important indicators of design quality. Finally, we speculate that such lack of correlation is due to the shortcomings of the models for measuring readability. We hope that this paper provides a call to action for the research community to engage with the topic of identifying better measurements for readability, which intuitively should have a significant impact on defect prediction, and ultimately on the code quality.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 1560526 and 1815486.

REFERENCES

[1] [n. d.]. Cloc tool. https://github.com/AlDanial/cloc/. Accessed: 2017-03-1.
[2] [n. d.]. Entropy library. https://pypi.python.org/pypi/entropy/0.9/. Accessed: 2017-03-1.
[3] [n. d.]. InFusion. http://www.intooitus.com/inFusion.html. Accessed: 2014-01-01.
[4] 2017. effsize library. https://cran.r-project.org/web/packages/effsize/effsize.pdf. Accessed: 2017-04-19.
[5] Krishan K Aggarwal, Yogesh Singh, and Jitender Kumar Chhabra. 2002. An integrated measure of software maintainability. In *Reliability and maintainability symposium, 2002. Proceedings. Annual.* IEEE, 235–241.
[6] Iftekhar Ahmed, Umme Ayda Mannan, Rahul Gopinath, and Carlos Jensen. 2015. An empirical study of design degradation: How software projects get worse over time. In *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on.* IEEE, 1–10.
[7] Ronald M Baecker and Aaron Marcus. 1989. *Human factors and typography for more readable programs.* ACM.
[8] Rajiv D. Banker, Srikant M. Datar, Chris F. Kemerer, and Dani Zweig. 1993. Software Complexity and Maintenance Costs. *Commun. ACM* 36, 11 (Nov. 1993), 81–94. https://doi.org/10.1145/163359.163375
[9] Raymond PL Buse and Westley R Weimer. 2008. A metric for software readability. In *Proceedings of the 2008 international symposium on Software testing and analysis.* ACM, 121–130.
[10] Raymond P.L. Buse and Thomas Zimmermann. 2011. *Information Needs for Software Development Analytics.* Technical Report MSR-TR-2011-8. Microsoft Corporation.
[11] Raymond PL Buse and Thomas Zimmermann. 2012. Information needs for software development analytics. In *Proceedings of the 34th international conference on software engineering.* IEEE Press, 987–996.
[12] Simon Butler, Michel Wermelinger, Yijun Yu, and Helen Sharp. 2010. Exploring the influence of identifier names on code quality: An empirical study. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on.* IEEE, 156–165.
[13] Ermira Daka, José Campos, Gordon Fraser, Jonathan Dorn, and Westley Weimer. 2015. Modeling readability to improve unit tests. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering.* ACM, 107–118.
[14] Eric Enslen, Emily Hill, Lori Pollock, and K Vijay-Shanker. 2009. Mining source code to automatically split identifiers for software analysis. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on.* IEEE, 71–80.
[15] Vincenzo Ferme, Alessandro Marino, and F Arcelli Fontana. 2013. Is it a Real Code Smell to be Removed or not?. In *International Workshop on Refactoring & Testing (RefTest), co-located event with XP 2013 Conference.*
[16] Francesca Arcelli Fontana, Elia Mariani, Andrea Mornioli, Raul Sormani, and Alberto Tonello. 2011. An experience report on using code smells detection tools. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on.* IEEE, 450–457.
[17] Francesca Arcelli Fontana and Marco Zanoni. 2011. On investigating code smells correlations. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on.* IEEE, 474–475.
[18] Martin Fowler and Kent Beck. 1999. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.
[19] Tracy Hall, Min Zhang, David Bowes, and Yi Sun. 2014. Some code smells have a significant but small effect on faults. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 4 (2014), 33.
[20] Maurice Howard Halstead. 1977. *Elements of software science.* Vol. 7. Elsevier New York.
[21] Mario Hozano, Henrique Ferreira, Italo Silva, Baldoino Fonseca, and Evandro Costa. 2015. Using developers' feedback to improve code smell detection. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing.* ACM, 1661–1663.
[22] Clemente Izurieta and James M Bieman. 2007. How software designs decay: A pilot study of pattern evolution. In *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on.* IEEE, 449–451.
[23] Clemente Izurieta and James M Bieman. 2008. Testing consequences of grime buildup in object oriented design patterns. In *Software Testing, Verification, and Validation, 2008 1st International Conference on.* IEEE, 171–179.
[24] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proceedings of the 11th working conference on mining software repositories.* ACM, 92–101.
[25] Foutse Khomh, Massimiliano Di Penta, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. An exploratory study of the impact of antipatterns on class change-and fault-proneness. *Empirical Software Engineering* 17, 3 (2012), 243–275.
[26] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. 2012. Technical debt: From metaphor to theory and practice. *Ieee software* 29, 6 (2012), 18–21.
[27] Dawn Lawrie, Henry Feild, and David Binkley. 2006. Syntactic identifier conciseness and consistency. In *Source Code Analysis and Manipulation, 2006. SCAM'06. Sixth IEEE International Workshop on.* IEEE, 139–148.
[28] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. 2007. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering* 3, 4 (2007), 303–318.
[29] Wei Li and Raed Shatnawi. 2007. An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *Journal of systems and software* 80, 7 (2007), 1120–1128.
[30] Steffen Olbrich, Daniela S Cruzes, Victor Basili, and Nico Zazworka. 2009. The evolution and impact of code smells: A case study of two open source systems. In *Proceedings of the 2009 3rd international symposium on empirical software engineering and measurement.* IEEE Computer Society, 390–400.
[31] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia, and Denys Poshyvanyk. 2013. Detecting bad smells in source code using change history information. In *Automated software engineering (ASE), 2013 IEEE/ACM 28th international conference on.* IEEE, 268–278.
[32] 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.
[33] TIOBE. [n. d.]. TIOBE Index. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html.
[34] Aiko Fallas Yamashita and Leon Moonen. 2013. Do developers care about code smells? An exploratory survey.. In *WCRE*, Vol. 13. 242–251.
[35] Zhifeng Yu and Václav Rajlich. 2001. Hidden dependencies in program comprehension and change propagation. In *Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on.* IEEE, 293–299.
[36] Nico Zazworka, Michele A Shaw, Forrest Shull, and Carolyn Seaman. 2011. Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt.* ACM, 17–23.