Expose - Advancing Code Readability: Mined & Modified Code for Dataset Generation

Lukas Krodinger

Master Thesis Proposal in M.Sc. Informatik
Faculty of Computer Science and Mathematics
Chair of Software Engineering II

Matriculation number	89801
Supervisor	Prof. Dr. Gordon Fraser
Advisor	Lisa Griebl

24th July 2023

0.1 MOTIVATION

In the realm of software development, the significance of code readability cannot be overstated. Together with understandability, it serves as the foundation for efficient collaboration, comprehension, and maintenance of software systems [PHD11; ASC02]. Maintenance alone will consume over 70% of the total lifecycle cost of a software product and for maintenance, the most time-consuming act is reading code [BW09; Dei85; Rug00; BB01]. Therefore, it is important to make sure that a high readability of code is ensured. In order to archive this, we need to measure readability and classify code regarding readability.

In the last years, researchers have proposed several metrics and models for assessing code readability with an accuracy of up to 81.8% [BW09; PHD11; Dor12; DCF+15]. In recent years, the newer deep learning based models are able to achieve an accuracy of up to 85.3% [MKX+18; MHO+22]. However, these models do not capture what developers think of readability improvements [FRH+19]. This suggests that there is room for improvement in readability classification of source code.

0.2 BACKGROUND AND RELATED WORK

0.2.1 READABILITY

In order to be able to improve readability, we first need to capture what readability is. We define readability as a subjective impression of the difficulty of code while trying to understand it [PHD11; BW09]. In other words, readability of code is a perceived barrier that needs to be overcome before it is possible to work with the code. The more readable code is, the lower the barrier [PHD11]. To give an example for high vs low readability, consider the code of Listing 1 and compare it to the code with the same functionality of Listing 2. You will notice that the first piece of code is much more readable than the second one.

Readability is not the same as complexity. Complexity is an "essential" property of software that arises from system requirements, while readability is an "accidental" property that is not determined by the problem statement [BW09; BK87].

There is another closely related term, namely understandability. Readability is the syntactic aspect of processing code, while understandability is the semantic aspect [PHD11]. For example, a developer can find a piece of code readable but still difficult to understand. Recent research gives evidence that there is no correlation between understandability and readability [SBV+17].

Listing 1: An example for highly readable code of the highly rated cassandra github repository

```
/**
 * Logs the output of the specified process.
 *
 * @param p the process
 * @throws IOException if an I/O problem occurs
 */
private static void logProcessOutput(Process p) throws IOException
{
    try (BufferedReader input = new BufferedReader(new
        InputStreamReader(p.getInputStream())))
    {
        StrBuilder builder = new StrBuilder();
        String line;
        while ((line = input.readLine()) != null)
        {
            builder.appendln(line);
        }
        logger.info(builder.toString());
    }
}
```

Listing 2: The same example modified to be badly readable

```
private
     static
void
    debug( Process
    v1
    )       throws IOException
{
// Doo debug
    try (BufferedReader b
= new
        BufferedReader(
            new InputStreamReader(
                v1.getInputStream()
                )
                )
                )
                {
    StrBuilder b2=new StrBuilder();String v2;while
        (null!=(v2=input.readLine())){b2.appendln(v2);} // Doo
        stuff
    m.info( builder.toString()
);
}
 }
```

0.2.2 CLASSICAL CALCULATION APPROACHES

A first estimation for source code readability was defined as the percentage of comment lines over total code lines [ASC02]. In the last years, researchers have proposed several more complex metrics and models for assessing code readability [BW09; PHD11; Dor12; SLP+16]. Those approaches used handcrafted features to calculate whether a piece of code is readable or not. They were able to achieve up to 81.8% accuracy in classification [SLP+16].

0.2.3 DEEP LEARNING BASED APPROACHES

More recent models use Deep Learning approaches in order to generate the features automatically. Those models have proven to be more accurate, achieving an accuracy of up to 85.3% [MKX+18; MHO+22].

All the mentioned models were trained on the data of Buse, Dorn and Scalabrio which was generated with surveys. They therefore asked developers several questions, including the question, how well readable the proposed source code is [BW09; Dor12; SLP+16].

Fakhoury et al. showed based on readability improving commit analysis that these models do not capture what developers think of readability improvements. They suggest taking other metrics such as incoming method invocations or method name fitting into account [FRH+19].

0.2.4 OTHER RELATED WORK

Loriot created a model that is able to fix Checkstyle[1] violations using Deep Learning. He therefore inserted formatting violations based on a project specific format checker ruleset into code in a first step. In a second step, he then learned how to undo those injections using a LSTM neural network. His approach is working on abstract token sequences. Note that all the training data is generated in a self-supervised manner [LMM22]. A similar idea has been explored by Yasunaga et al. [YL20]. We will use the idea of intentional degradation of code for data generation.

Another concept we will employ is from Allamanis et al. He cloned the top open source Java projects on GitHub[2] for training a Deep Learning model. Those top projects were selected by taking the sum of the z-scores of the number of watchers and forks of each project. As the projects have thousands of forks and

[1] `https://checkstyle.org/`
[2] `https://github.com/`

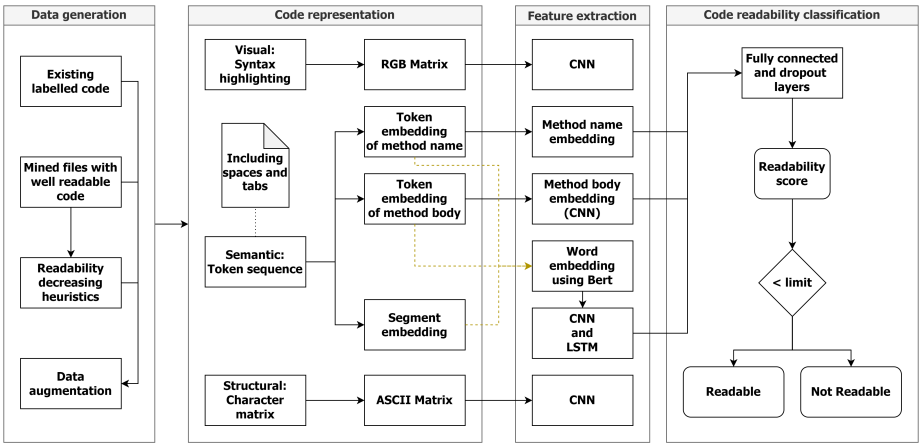

Figure 1: Overview of the planned approach.

stars and are widely used among software developers, they can be assumed to be of high quality [APS16].

0.3 PLANNED WORK AND CONTRIBUTION

We will investigate whether it is possible to score a higher accuracy as current models in classifying code readability using Deep Learning. Therefore, we will train the model from Mi et al. [MHO+22] with more data. We will consider augmenting the model with a method name classifier and incorporating semantic encoding for tabs and spaces. The training data will be generated in a novel way for classification of readability, inspired by Loriot's Styler [LMM22]. The method name classifier would be similar to Code2Vec [AZL+19]. The combination of all components would be novel to the best of our knowledge. You can find a visualization of the planned modifications of Mi et al.'s model in figure 1. We will focus on generating training data, as the approach will be usable for further research in the field of source code readability.

Deep Learning based models perform better the more training data they get [HNA+17]. Therefore, one approach in order to further improve existing models is to gather more training data. This requires, as it was done previously, a lot of effort and persons willing to rate code based on their readability. We present another approach for gathering training data.

In a first step, GitHub repositories with known high code quality are downloaded and labeled as highly readable. We therefore select repositories using a similar approach as Allamanis et al. [APS16] and then assume that they contain

only well readable code. In a second step, the code is manipulated so that it is less readable afterwards. This approach is similar to the approach of Loriot [LMM22].

RQ1: WHAT IS THE QUALITY OF OUR NEW DATA GENERATION APPROACH?

So far, the data for readability classification was generated manually. Therefore, human annotators ranked code snippets by their readability level based on a five-point Likert scale [Lik32] ranging from one (i.e., very unreadable) to five (i.e., very readable) [BW09; Dor12; SLP+16]. Our new data will be generated in an automatic approach, where no manual labeling of code is necessary. We want to compare the quality of this data to the data used for previous models.

RQ1.1: CAN CERTAIN CODE BE ASSUMED TO BE WELL READABLE?

To collect the necessary training data, we assume that the code from the repositories is readable under certain conditions. We want to check whether that assumption holds.

RQ1.2: CAN POORLY READABLE CODE BE GENERATED FROM WELL READABLE CODE?

It is not sufficient to have only well readable code for training a classifier. We also need poorly readable code. Therefore, we will try to generate such code from the well readable code. We will investigate whether this is possible in principle, and we will propose an automated approach for archiving this: Readability Decreasing Heuristics.

RQ1.3: WHICH HEURISTICS ARE BEST TO GENERATE POORLY READABLE CODE FROM WELL READABLE CODE?

We want to compare the modifications of the proposed heuristics for generating badly readable code to each other. We will investigate whether their results are sufficient for training a readability classifier.

RQ2: CAN THE NEW DATA IMPROVE EXISTING READABILITY MODELS?

It was shown that Deep Learning models get better the more training data is available [HNA+17]. This holds under the assumption that the quality of the data is the same or at least similar. We want to check if the quality of our new

data is sufficient for improving the Deep Learning based readability classifier of Mi et al. [MHO+22].

OPTIONAL RQ3: HOW CAN EXISTING DEEP LEARNING APPROACHES BE FURTHER IMPROVED?

In recent years it was shown that Deep Learning models can be further improved by modifying the structure of the architecture or by introducing new components, parts or layers to existing architectures. We suggest two improvements for the model of Mi et al. [MHO+22] namely embedding spaces and tabs as semantic tokens and adding a method name fitting component. We want to investigate if those changes can improve the model.

OPTIONAL RQ3.1: DOES THE EMBEDDING OF SPACES AND TABS IN SEMANTIC CODE REPRESENTATIONS IMPROVE READABILITY CLASSIFICATION?

The state-of-the-art model of Mi et al. [MHO+22] does consider spaces and tabs only in its visual component. We want to investigate if it can improve the quality of a Deep Learning based model if spaces and tabs are encoded as semantic tokens. We also want to investigate if this makes the visual component superfluous.

OPTIONAL RQ3.2: DOES THE USAGE OF A METHOD NAME FITTING CLASSIFIER IMPROVE READABILITY CLASSIFICATION?

Correct naming of identifiers is crucial for ensuring readability of software programs. It is of outstanding importance for readability of code that the name of methods fit the method bodies [LKB+19]. We want to introduce a new component to the model of Mi et al. [MHO+22] that is built similar to Code2Vec [AZL+19]. We want to investigate if the newly introduced component increases the quality of the resulting model.

Bibliography

[ASC02] K. K. Aggarwal, Y. Singh and J. K. Chhabra. 'An integrated measure of software maintainability'. In: *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No. 02CH37318)*. IEEE. 2002, pp. 235–241 (cit. on pp. 1, 3).

[APS16] M. Allamanis, H. Peng and C. Sutton. 'A convolutional attention network for extreme summarization of source code'. In: *International conference on machine learning*. PMLR. 2016, pp. 2091–2100 (cit. on p. 4).

[AZL+19] U. Alon, M. Zilberstein, O. Levy and E. Yahav. 'code2vec: Learning distributed representations of code'. In: *Proceedings of the ACM on Programming Languages* 3.POPL (2019), pp. 1–29 (cit. on pp. 4, 6).

[BB01] B. Boehm and V. R. Basili. 'Defect reduction top 10 list'. In: *Computer* 34.1 (2001), pp. 135–137 (cit. on p. 1).

[BK87] F. Brooks and H. Kugler. *No silver bullet*. April, 1987 (cit. on p. 1).

[BW09] R. P. Buse and W. R. Weimer. 'Learning a metric for code readability'. In: *IEEE Transactions on software engineering* 36.4 (2009), pp. 546–558 (cit. on pp. 1, 3, 5).

[DCF+15] E. Daka, J. Campos, G. Fraser, J. Dorn and W. Weimer. 'Modeling readability to improve unit tests'. In: *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015, pp. 107–118 (cit. on p. 1).

[Dei85] L. E. Deimel Jr. 'The uses of program reading'. In: *ACM SIGCSE Bulletin* 17.2 (1985), pp. 5–14 (cit. on p. 1).

[Dor12] J. Dorn. 'A general software readability model'. In: *MCS Thesis available from (http://www. cs. virginia. edu/weimer/students/dorn-mcs-paper. pdf)* 5 (2012), pp. 11–14 (cit. on pp. 1, 3, 5).

[FRH+19] S. Fakhoury, D. Roy, A. Hassan and V. Arnaoudova. 'Improving source code readability: Theory and practice'. In: *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*. IEEE. 2019, pp. 2–12 (cit. on pp. 1, 3).

[HNA+17] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang and Y. Zhou. 'Deep learning scaling is predictable, empirically'. In: *arXiv preprint arXiv:1712.00409* (2017) (cit. on pp. 4, 5).

[Lik32] R. Likert. 'A technique for the measurement of attitudes.' In: *Archives of psychology* (1932) (cit. on p. 5).

[LKB+19] K. Liu, D. Kim, T. F. Bissyandé, T. Kim, K. Kim, A. Koyuncu, S. Kim and Y. Le Traon. 'Learning to spot and refactor inconsistent method names'. In: *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE. 2019, pp. 1–12 (cit. on p. 6).

[LMM22] B. Loriot, F. Madeiral and M. Monperrus. 'Styler: learning formatting conventions to repair Checkstyle violations'. In: *Empirical Software Engineering* 27.6 (2022), p. 149 (cit. on pp. 3–5).

[MHO+22] Q. Mi, Y. Hao, L. Ou and W. Ma. 'Towards using visual, semantic and structural features to improve code readability classification'. In: *Journal of Systems and Software* 193 (2022), p. 111454 (cit. on pp. 1, 3, 4, 6).

[MKX+18] Q. Mi, J. Keung, Y. Xiao, S. Mensah and Y. Gao. 'Improving code readability classification using convolutional neural networks'. In: *Information and Software Technology* 104 (2018), pp. 60–71 (cit. on pp. 1, 3).

[PHD11] D. Posnett, A. Hindle and P. Devanbu. 'A simpler model of software readability'. In: *Proceedings of the 8th working conference on mining software repositories*. 2011, pp. 73–82 (cit. on pp. 1, 3).

[Rug00] S. Rugaber. 'The use of domain knowledge in program understanding'. In: *Annals of Software Engineering* 9.1-4 (2000), pp. 143–192 (cit. on p. 1).

[SBV+17] S. Scalabrino, G. Bavota, C. Vendome, M. Linares-Vásquez, D. Poshyvanyk and R. Oliveto. 'Automatically assessing code understandability: How far are we?' In: *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2017, pp. 417–427 (cit. on p. 1).

[SLP+16] S. Scalabrino, M. Linares-Vasquez, D. Poshyvanyk and R. Oliveto. 'Improving code readability models with textual features'. In: *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. IEEE. 2016, pp. 1–10 (cit. on pp. 3, 5).

[YL20] M. Yasunaga and P. Liang. 'Graph-based, self-supervised program repair from diagnostic feedback'. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10799–10808 (cit. on p. 3).