FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 46 (2021) No. 2
ISSN 0867-6356
sciendo DOI: 10.2478/fcds-2021-0009 e-ISSN 2300-3405

Effect or Program Constructs on Code Readability and Predicting Code Readability Using Statistical Modeling

Aisha Batool, Muhammad Bilal Bashir, Muhammad Babar*
Adnan Sohail, Naveed Ejaz

Abstract. In software, code is the only part that remains up to date, which shows how important code is. Code readability is the capability of the code that makes it readable and understandable for professionals. The readability of code has been a great concern for programmers and other technical people in development team because it can have a great influence on software maintenance. A lot of research has been done to measure the influence of program constructs on the code readability but none has placed the highly influential constructs together to predict the readability of a code snippet. In this article, we propose a novel framework using statistical modeling that extracts important features from the code that can help in estimating its readability. Besides that using multiple correlation analysis, our proposed approach can measure dependencies among different program constructs. In addition, a multiple regression equation is proposed to predict the code readability. We have automated the proposals in a tool that can do the aforementioned estimations on the input code. Using those tools we have conducted various experiments. The results show that the calculated estimations match with the original values that show the effectiveness of our proposed work. Finally, the results of the experiments are analyzed through statistical analysis in SPSS tool to show their significance.

Keywords: Code Readability, Program Constructs, Code Readability Metrics, Statistical Modeling, Code Readability Prediction

*Computing & Technology Department, IQRA University, Islamabad, Pakistan
Email: aishabatool_comsats@yahoo.com, {bilal.bashir, adnan.sohail, naveed.ejaz}@iqraisb.edu.pk
Department of Computer Science, Allama Iqbal Open University (AIOU), Islamabad, Pakistan
Corresponding Author*: muhammad.babar@aiou.edu.pk

1. Introduction

Source code readability is the capability of the code that makes it legible and comprehensible for its users including programmers, testers, and maintainers. Source code readability is a fundamental notion related to the comprehension of text. Readable code has many advantages to offer not only to the owner of the code (who actually wrote it in the first place) but to the users of the code (other technical people like co-programmers, testers who want to test it, and later on maintainers who want to modify it). It is also related to other software quality attributes such as modifiability, understandability, reusability, and maintainability. The readable code can be easily modified by the programmers because it is easy to understand. Similarly a readable code can be re-used in a different module or even in a similar new project.

Maintainability is the ability of a software to go through revision(s) with minimum possible effort. Maintenance can be either corrective (fixing a logical bug) or progressive (adding a new feature in the software). It is observed that software developers these days, spend more time maintaining and evolving existing software than writing new code. Besides that research has proven that maintenance consumes approximately 75% of the software development cost [27]. In the study of Collar and Ricardo [3] authors have shown with the help of experiments using procedural language (Visual Basic.NET) that how readability can be analyzed and can affect the maintenance phase. Keeping in mind the importance of maintainability, reading the code is the first step towards maintenance. The researchers have distinguished the act of reading a piece of code is the most prolonged activity among all maintenance activities. There are many elements that affect readability in positive and negative way but Hofmeister et. al [9] discuss in their research that the identifier names have strong affect on code readability and understandability. Their findings reveal that better readability saves a lot of time during maintenance that can help in reducing maintenance and ultimately software development cost.

Code readability is gaining more and more attention of researchers [22, 27, 26, 18, 1] and specifically in [26] researchers are trying to identify the impact of different program constructs on the code readability. For this purpose, authors have conducted a survey from experienced programmers and used statistical techniques to determine, which constructs influence the readability and by what factor. According to their findings, meaningful identifiers, consistency, and comments are the most significant and having positive impact on code readability. On the contrary, complex arithmetic expressions and nested loops put negative impact on the same. Although literature witnesses many software readability metrics; text readability metrics [8, 12, 7, 23] and code readability metrics [18, 1] but still not a single comprehensive set of parameters are there to compute the code readability. Well known natural language readability indexes are Gunning's Fog [7], Flesh Kincaid [12], Automated Readability Index (ARI) [23] and SMOG [8]. These metrics use sentence length; characters per word, syllables count and so on as parameters. Flesh-Kincaid readability index is also being used in text editor (Microsoft Word) as standard readability metric [18]. Source code readability metrics also include Halstead's Product Metrics and Buse and Westley readability metrics [1] and a new metric proposed by Namani1 and Kumar [18]. In

these metrics features that are used for computing readability include lines of code, number of identifiers, identifiers length, keywords frequency and so on.

Namani and Kumar [18] have used seven program constructs; Line of code, Line length, Blank lines, Blank spaces after directive statements, Number of methods, Comments, and Line breaks. Many authors considered natural language readability indexes as code readability metrics [5]. Authors are also focusing on the textual features and neural machine translation to improve the readability of programming code [27, 21]. Various surveys have been conducted in the research studies on readability and it is found that programmers mostly take help from stack overflow for small programming tasks as it offers better code as far as readability is concerned [29]. In another research, authors have investigated if for small programs, is it worth writing comments in it and if it makes any negative or positive impact on the readability [19].

Although the existing research provides a lot of details on the program constructs leaving good or bad impact on the code readability but it lacks with a standard expression or an equation that puts all the most influential constructs together to predicate the code readability. There is a need of a specific framework that can relate the program constructs and their influence with human judgment to identify the most influential ones. A specific approach is required to measure the dependency of program constructs to estimate the readability. A particular code readability tool is also required to estimate code and text readability of given text or code chunk. In this research, all these issues have been handled and details are presented in coming sections.

The rest of the paper is organized as follow: Section 2 describes literature review. Section 3 provides the description of the proposed scheme. The analysis and results are provided in Section 4. The conclusion is specified in Section 5.

2. Literature Review

This section presents review on the survey literature on code readability and code readability metrics. The following text presents all the related information on it.

Software understandability becomes a critical challenge when the developers try to improve the source code written by other developers. The "integral if understandability" model is proposed to discuss the factors which affect the understandability of software [14]. These factors include software documentation, software structure, program components, source code, and the input data. Conventionally, readability metrics extract features from textual documents and do not consider the conceptual contents and writing styles. The authors in [24] have proposed a readability estimation model that is based on the combination of traditional features and the statistical estimation methods. The proposed model uses unigram language models to determine the probability values for readability estimation. In addition, the authors have articulated that the combination of surface linguistic features and content information is better than using these methods alone. The proposed estimation model helps in combining the surface linguistic features and content-based features for a single classifier. However, the proposed methods are used for textual classification and do

not covers the readability aspects of the source codes.

Machine learning algorithms can help in the prediction of readability of textual documents. The authors in [11] have used diverse linguistic features based learning model for readability prediction. To this end, an analysis of 540 textual documents were made, which they had collected from various online sources including Wikipedia articles, newspapers, manual transcripts, web blogs, and machine translated data. The proper naming of identifiers in the source code enhances the readability. A specific tool is designed to assist programmers in dynamically naming identifiers [20]. The tool is based on heuristic-based dynamic reporting methods whereby the programmers can choose from suggested identifiers. The authors have conducted study with two groups of programmers called controlled group and experimental group. A study to find the cohesion and difficulty at various levels of textual documents was presented by authors in [16]. The authors presented Coh-Metrix, which is a computational tool for measurement of cohesion and text difficulty. The study was made over a corpus of 32 textual documents, which also include the readability score manually annotated by researchers previously. The analysis of results show that variables in Coh-Mextrix are highly correlated with each other, which accounts about 91% of the variance of base dataset that was manually annotated by Burmuth. However, the researchers were unable to conclusively determine the effectiveness of each variable.

Another study has been conducted by researchers in order to validate the Coh-Metrix for textual documents [17]. The authors selected 34 articles initially, that performed similar experiments, and selected 13 of them which met their selection criteria. Two of the selected studies were redundant so authors selected 19 pairs of text from remaining 12 studies. However, the study overall validates the causal and co-reference indexes by successfully detecting the texts which were intentionally manipulated by the researchers. Code readability becomes a critical challenge when the source code is shared among many programmers in different teams, sometimes in different countries. An extensive study has been conducted by the researchers in order to assess the impact of different program constructs on the code readability [16, 17]. To propose the source code readability measure [2], the authors have performed a thorough study. First of all, they selected 100 code snippets of Java programming language from five open source projects. After that the authors conducted a survey from 120 participants from University of Virginia and instructed them to annotate and assign the score to each code snippet according to their own understanding. The experiment was performed using GUI based code ranking tool.

The authors in [4] have improved the work of Buse and Westley [2]. They use entropy based predictive modeling approach in order to improve code readability. The main contributions of the article are that proposed model that looks simple and demonstrates the improved performance. The authors first perform the correlation analysis to find the impact of code size on the code readability. To this end, the authors consider six features namely average mathematical equations, average number of comments, maximum indentations in the code, maximum number of words, maximum length of code lines, and maximum occurrence of a specific character in the code snippets. Coding conventions are devised to ensure the consistency in coding and improving code readability and maintenance at later stages. The authors in [25]

conduct a study to find whether the violations of coding conventions impact the code readability. In addition they study the types of coding violations that impact the readability. A specific empirical study of the associations between software complexity and code readability is also presented [6]. Program repair ingredients are also repaired using via deep learning similarities of code [28].

3. Proposed Work

In this section we present our proposed work related to the source code readability, effects of program constructs on readability, and predicting the code readability with the help of program constructs. The details are presented as follows.

The main objective of our work is to estimate the relevant contribution of every program construct in readability and to express code readability by a formula using small number of features. Figure 1 highlights our proposed approach with the help of a simple model. The code snippets of `C#` programming language in the form of questionnaire are distributed among programming experts to get the human judgments about code readability. Snippet selection is a bit restricted that it must be as small as annotators can easily read and also contain some of the readability concerned features. A CRT (Code Readability Tool) is developed to extract the readability parameters and to automate the code readability metrics (see Section 3.4 for details). A rank correlation is calculated between code features and human judgments. These ranks are analyzed and significant features are included for regression analysis. Now these significant features are used in regression analysis, and required regression equation is formed. This regression equation is used to predict the readability values of each snippet and finally the human judgments and predicted values are compared.

Results show that original and predicted values are almost same with a minor error; therefore, we propose this equation to be used for the prediction of source code readability. An application is developed using `C#` for the calculation of software readability. Software readability includes source code as well as documentation readability. Therefore, application is able to calculate Automated Readability Index, Flesh-Kincaid and Gunning Fog index for documentation readability. The detail of proposed scheme is given in sub sections.

3.1. Human Readability Judgment

Formally the readability is a mapping of finite rank (in the form of a value) to a sample of code or text. Similarly estimation of code readability requires having a set of code snippets assigned with readability ranking by human (most likely programmers). For this purpose, a set of code samples were selected and given to annotators (programmers who can understand those codes, have knowledge about program constructs used in them and can rank them as per their knowledge and skill).

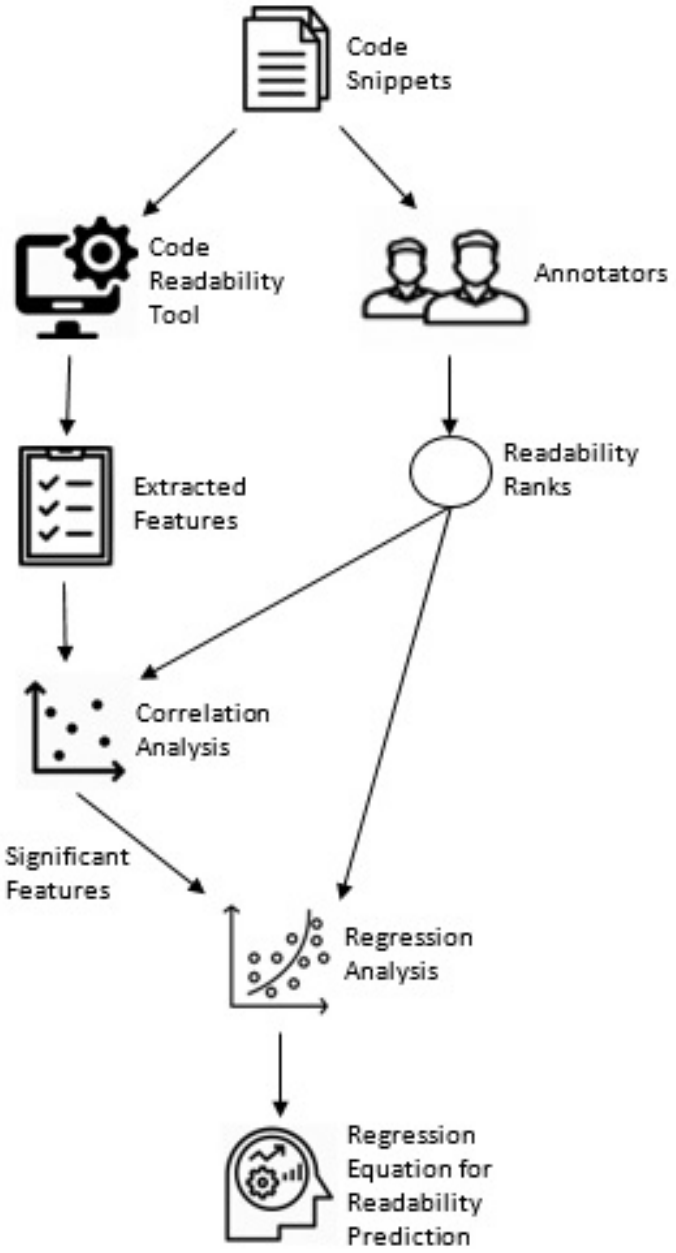

Figure 1. Model of Proposed Work

3.1.1. Selection of Code Snippets

For the purpose of human judgment, we selected code snippets of C# programming language. While selection of the code samples, we made sure that they are neither too short or too long. Also code snippets contain program constructs that we want to compute significance for. We selected around 100 such snippets and organized 20 questionnaires that we distributed among programmers. Some code samples have been made public for the researchers and students and they can be downloaded from the online source `https://github.com/bilalbezar/csharp`. The code snippets are of different length ranging from `11` to `216` lines of code. The average lines of code per file is `41`.

3.1.2. Ranking of Code Snippets

The annotators were asked to evaluate the code as per their knowledge and experience. The rank was decided to be from 1 to 5 where 1 means the lowest level of readability and 5 means the highest. It is also shown in the figure 2. After the annotators finished ranking code samples, they were collected back. The results of ranking were stored in SPSS tool for further statistical analysis.

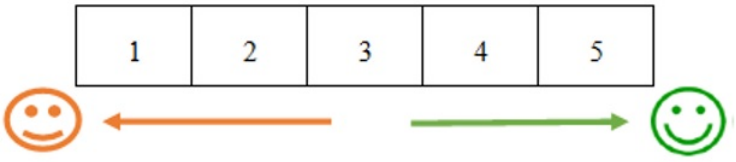

Figure 2. Ranking Criteria

3.2. Significance Measurement of Program Constructs

Source code is a highly organized set of statements written to achieve a specific purpose; logical statements, declarations, documentation in the form of comments and many more. Therefore, natural language readability and source code readability parameters are quite different. Code readability is concerned with its size, comments, naming style, keywords, and so on. Authors in [26] propose a model that contains 22 different program constructs that impact the readability of code. They are presented in Table 1 and we have also used them in this study to estimate the significance of these constructs on code readability.

Table 1. Program Constructs that Effect Code Readability

No.	Program Constructs	Description
1	Meaningful Names	Human understandable identifiers
2	Comments	Comments added for documentation
3	Spacing	White spaces
4	Indents	Indentations
5	Short Scopes	Scopes
6	Line Length Distribution	Distribution of the code
7	Identifier Name Length	Length of names used in declarations
8	Arithmetic Formulas	Complexity of arithmetic expressions
9	Identifier Frequency	Number of times an identifier is used
10 11 12	If-else Nested If Switch	Conditional Statements
13 14 15	For Loop While Loop Do-while	Iterative Statements
16	Nested Loop	Iteration(s) within Iteration
17	Recursive	Repetition through recursion
18	Arrays	Array declarations and referencing
19 20 21	Classes Distribution Inheritance Overriding	Object-oriented features
22	Consistency	Consistency in the coding style

3.2.1. Frequency of Program Constructs

To design a code readability metric, first we need to extract frequency (occurrence) of program constructs from the code snippets. But even for a single construct, just counting its occurrence in a code snippet will not give us accurate information on readability. It is because of the fact that every code snippet usually have different lines of code (LoC) and this difference can also vary from small to large. To scale the occurrences of a construct, we have used percentages and averages in most of the cases. For remaining constructs including line of code, parenthesis, and characters, we have just used their frequency by counting them. Next we present the formulas that we have used to calculate the frequency of program constructs as per our proposed technique.

- **Frequency of Identifiers:** It is calculated as the percentage of identifiers;

$$PercentageIdnt = \frac{NOI * 100}{LoC} \tag{1}$$

where NOI is total number of identifiers in the program & LoC is lines of code.

- **Length of Identifiers:** It is calculated as the average of identifiers' length;

$$AvgIdntL = \frac{LOI}{TOI} \tag{2}$$

where LOI is total length of identifiers in the program & TOI is total identifiers.

- **White Spaces:** They are calculated as the average with lines of code;

$$AvgS = \frac{TS}{LoC} \tag{3}$$

where TS is total number of white spaces.

- **Blank Lines:** They are calculated as the percentage of blank lines;

$$PercentageBL = \frac{TBL * 100}{LoC} \tag{4}$$

where TBL is total number of blank lines.

- **Frequency of Keywords:** It is calculated as the percentage of keywords;

$$AverageKW = \frac{NOK}{LoC} \tag{5}$$

where NOK is total number of keywords occurrences in the program.

- **Length of Code Line:** It is calculated as the average number of characters in each code line;

$$AvgLL = \frac{TCC}{LoC} \tag{6}$$

where TCC is total number of characters in the source code.

- **Frequency of Comments:** It is calculated as the percentage of comments;

$$PercentageNOC = \frac{NOC * 100}{LoC} \tag{7}$$

where NOC is total number of lines covered by comments in the program.

- **Frequency of Branches:** It is calculated as the percentage of branches;

$$PercentageNOB = \frac{NOB * 100}{LoC} \tag{8}$$

where NOB is total number of branches in the program.

- **Frequency of Loops:** It is calculated as the percentage of loops;

$$PercentageLoops = \frac{NOL * 100}{LoC} \tag{9}$$

where NOL is total number of loops in the program excluding recursive function calls.

- **Frequency of Operators:** It is calculated as the percentage of operators;

$$PercentageOpr = \frac{NOO * 100}{LoC} \tag{10}$$

where NOO is total number of operators in the program.

All the calculated frequencies in the form of counts, averages, and percentages from code snippets were stored in Excel sheets for further statistical analysis.

3.2.2. Rank Correlation Analysis

To compute effect of program constructs on code readability, we have performed multiple correlation analysis between percentage values of selected constructs and readability rank values of code snippets. Multiple correlation is used to find relationship between two variables and the strength of their relation. The percentage values for selected program constructs are computed using code readability tool (CRT) and readability rank values for code snippets are obtained from human judgment. We have used multiple correlation formula from the book on statistics by Richard [13]. The correlation value lies between 0 and 1 and to compute the strength of relation, we have used Cohen's standard. According to Cohen's affect size, if the coefficient value is between 0.10 and 0.29 it would represent weak relation. Coefficients, which are less than 0.49 and greater than 0.30 represent medium association. And coefficient values greater than 0.5 denotes strong correlation between both variables. After applying multiple correlation analysis, we have obtained values shown in Table 2.

Table 2. Correlation Coefficient of Program Constructs

No.	Program Construct	Correlation Cofficient
1	Lines of Code	0.720
2	Blank Lines	0.561
3	Characters	0.560
4	Parenthesis	0.549
5	Identifiers Length	0.501
6	Spaces	0.497
7	Identifiers	0.281
8	Comments	0.220
9	Branches	0.189
10	Indentations	0.158
11	Operators	0.127
12	Keywords	0.102
13	Loops	0.101
14	Line Length	0.058

3.3. Code Readability Predication

After finding correlation between program constructs and code readability, the second and the most significant contribution of our research study is the prediction of code readability. We have devised a multiple regression equation that can help us predicting readability of a given code snippet. Multiple regression analysis is a statistical technique designed to predict the value of one dependent variable based on the 2 or more independent variables. In our regression model, we use readability value obtained from human judgments as dependent variable. While independent variables are those program constructs that we find significant during correlation analysis. After analyzing the results, we have come up with following regression equation, which can predict readability of a code snippet;

$$CR = 4.317 + LoC * (-0.037) + BL * (0.015) + NOP * (-0.005) + IL * (-0.093) \quad (11)$$

where: CR is code readability, LoC is lines of code, BL is number of blank lines, NOP is number of parenthesis, and IL is length of identifiers.

3.4. Automation

The objective of our research is to automate the human readability judgments for code snippets. For this purpose we have developed a code readability tool (CRT) that can predict code readability of the code given as input to it. Figure 3 shows brief architecture of the CRT tool. As shown in the figure, it has three sub-components; *feature extractor*, *frequency scaler*, and *readability predictor*. The tool receives code snippets of C# language as input and the very first sub-component extracts program constructs from the input code as features. These features are then fed into frequency scaler that computes the frequencies of all the program constructs and scales them so they can be used to predict readability. Finally the scaled frequencies are sent to readability predictor that makes the prediction using multiple regression equation that we have proposed and presented in Section 3.3. The readability predictor generates code's readability value as its output.

4. Experiments and Results

Experiments for the evaluation of proposed technique are carried out using our implemented CRT tool and results are analyzed using statistical techniques. Total one hundred ranked code snippets of C# are used to verify the effectiveness of our proposed approach. In this regard, code readability tool (CRT) is used to extract program constructs and to predict readability value for input code snippets. Prediction accuracy is checked using two techniques where predicted and actual values are compared. For experiments and analysis, we have used CRT, SPSS, and Microsoft Excel. SPSS is used to perform multiple regression analysis, correlation analysis and many other

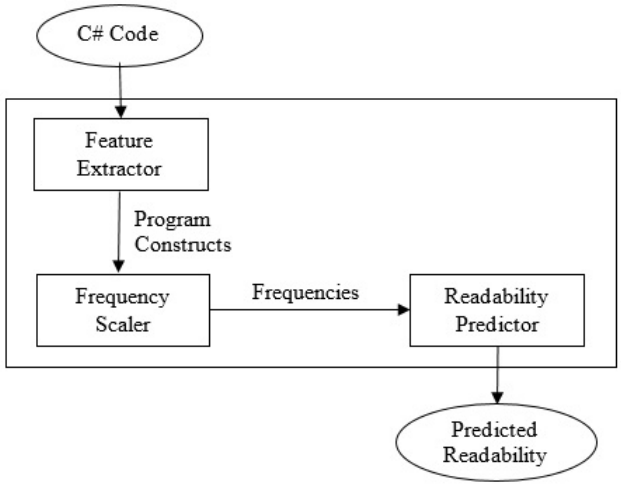

Figure 3. CRT Architecture

tests. Testing of proposed metric is performed in two steps. Firstly we have tested our proposed multiple regression model and secondly predicted values are tested for accuracy in comparison to actual values.

4.1. Testing the Proposed Model

We have used three statistical techniques to test our proposed multiple regression model and they are F-Test, t-Test, and distribution of residuals. These tests check model for model fitness as a whole, significance of each explanatory variable, and for any possible error respectively. The three tests and their results are described in upcoming sub-sections.

4.1.1. The *F-Test*

The *F-test* is used to check whether code readability depends on all considered features (program constructs) or has it happened by chance. We may also say that if the proposed regression model as a whole is significant or not? In SPSS, the output of regression analysis, the F-value, is included in output. The table containing F-value is also called ANOVA (Analysis Of Variance) table. Table 3 presents the ANOVA table that we have obtained after performing regression analysis in SPSS.

The formula used to perform the *F-test* is as follows and have been obtained from [13];

$$F = \frac{SSR/k}{SSE/(n-k-1} \tag{12}$$

Table 3. Analysis of Variance

Model	Sum of Squares	df	Mean Squares	F	Sig.
Regression	32.97	6	5.496	17.668	$.000^b$
Residual	30.485	98	.311		
Total	63.462	104			

where *SSR* is regression sum of mean squares (5.496), *SSE* is error sum of squares (0.311), k is the degree of freedom, and n is the total number of observations.

In the Table 3, we see that F-value is far greater than 1.97 that shows our regression analysis is significant. We can reach to the same conclusion by noticing that p-value is 0.000 that is less than the significant level 0.01.

4.1.2. The *t-Test*

In multiple regression analysis, we include some of the explanatory variables, which are independent variables. These variables are supposed to be significant in terms of relative change in dependent variable. The *t-Test* is a technique to check whether independent variables are significant or not. In our proposed model, we have used seven program constructs as independent or explanatory variables and one dependent variable, which is code readability. We have performed the *t-Test* for every program construct using following formula:

$$t = \frac{b_i - B_i}{S_{b_i}} \tag{13}$$

where b_i is slope of fitted regression, B_i is actual slope hypothesized for the population, & S_{b_i} is the standard error of the regression coefficient.

The t-value for each construct is calculated using SPSS, and Table 4 presents t-ratios for all of them.

4.1.3. Distribution of Residuals

Most important part of regression analysis is looking at the residuals. We can say that if residuals show any nonrandom pattern, it indicates there is something wrong in the model. Our proposed model shows a random pattern in residuals as shown in the Figure 4.

4.2. Measuring the Prediction Accuracy

It is not enough to have an accurate regression model. We also need to test the accuracy of the prediction, which is, whether the actual and the predicted values

Table 4. Analysis of Variance

No.	Program Construct	t-ratio	Significance
1	Lines of code	-9.684	0.000
2	Blank lines	5.704	0.000
3	Characters	-6.423	0.000
4	Parenthesis	-6.480	0.000
5	Identifiers length	-5.912	0.000
6	Spaces	-5.587	0.000
7	Identifiers	2.989	0.004
8	Comments	-1.876	0.064
9	Branches	-0.439	0.662
10	Indentations	-1.623	0.108
11	Operators	-0.652	0.516
12	Keywords	1.267	0.208
13	Loops	-0.031	0.751
14	Line length	1.047	0.297

are similar or not. For prediction accuracy, we calculate two measures; mean absolute percentage error and mean absolute deviation. Both of these measures show percentage and average errors in the predicted values by a prediction model.

4.2.1. The Mean Absolute Percentage Error (MAPE)

Mean absolute percentage error (MAPE) [10] is a statistical measure, which is used to evaluate a prediction model. This measure explains prediction error in the form of percentage. It is also known as means absolute percentage deviation. Percentage error of prediction is calculated using the following formula:

$$MAPE = \frac{\sum_{t=1}^{n} |A_t - F_t|}{F_t} * 100 \tag{14}$$

where A_t is actual values of code readability and F_t is predicted values of code readability.

The percentage prediction error for our proposed technique is 12.08%, which shows that 87.02% values are accurately predicted.

4.2.2. The Mean Absolute Deviation (MAD)

The mean absolute deviation (MAD) [15] is another statistical measure to evaluate prediction models. This measure shows an average error of prediction. For mean deviation error (MAD), we use following formula:

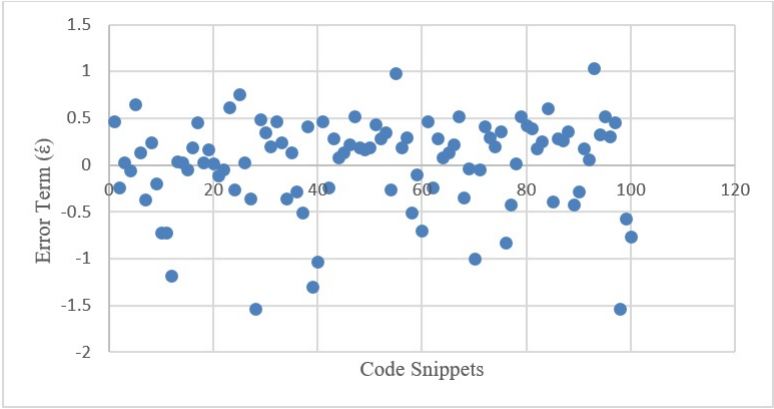

Figure 4. Random Pattern in Residuals

$$MAD = \frac{\sum_{i=1}^{n} |A_i - F_i|}{n} \tag{15}$$

where A_i is actual values of code readability and F_i is predicted values of code readability, and n is total number of samples (code snippets).

The resultant value of mean absolute deviation (MAD) is calculated as 0.414, which shows a minor error of prediction in the proposed model.

4.3. Comparative Analysis & Discussion

Proposed model is evaluated for fitness of model and prediction accuracy. Percentage of prediction accuracy is 87.02% and average deviation from actual values is 0.414 that is too less. Original and predicted values are compared and shown in Figure 5 visually, where similar patterns commit high accuracy of prediction. Moreover we also take descriptive statistics of predicted, original and error term values to show the validity of proposed technique. The comparison of actual and predicated values clearly show that our proposed regression model has performed well in predicting the code readability and can help programmers to judge how readable is their code.

Threats to Validity Although a great care has been taken to ensure that the research study is conducted without bias but there are some threats that can pose challenge to this research. We have listed them below.

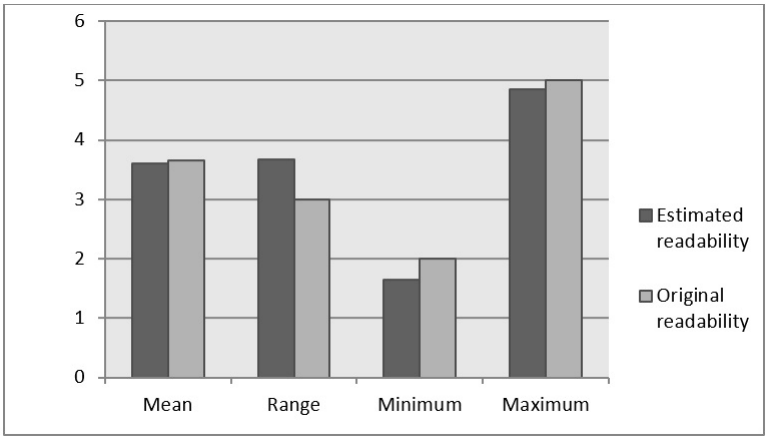

Figure 5. Comparative Analysis of Actual and Predicted Values

4.3.1. Code Snippets Instances

The number of instances used in experiments and analysis can cause a validity threat. For more confidence in the results, more code snippets should be added.

4.3.2. Number of Practitioners

The number of practitioners involved in the research to collect their judgment on code readability can cause the results to variate.

4.3.3. Practitioners Skills

Mostly the students working in C# programming language were involved in this research. The working experience and programming skill may cause the results to variate.

5. Conclusion

The code readability is the ability of source code that makes it readable and understandable to everyone. In this article, a new metric for source code readability prediction is proposed based on statistical modeling. The aim is to calculate the effectiveness of each program constructs along with the code readability prediction and automating this prediction process. Firstly, 100 snippets are examined by the programmers and the obtained readability values are given as input to the SPSS for

statistical analysis. Correlation between readability and program constructs is calculated using SPSS. Multiple regression models are tested for its fitness and significance of explanatory variables using the F-Test, t-Test and Residual's randomness. Results show around 90% prediction accuracy. It shows that we can use multiple regression equation to predict readability of any chunk of code. Moreover, a code readability tool is developed that can estimate code readability of code snippets written in C# language. Finally, predicted values and original values are compared using MAPE and MAD measures, which highlights the effectiveness of the proposed prediction model with minimum mean error.

References

[1] Buse R. P. L., Weimer W.R., A metric for software readability. *Proceedings of the 2008 international symposium on Software testing and analysis*. ACM, 2008.

[2] Buse R. P. L., Weimer W. R., Learning a Metric for Code Readability, *In IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546-558, July-Aug. 2010.

[3] Collar E. , Ricardo V., Role of software readability on software development cost, *In the proceedings of the 21st Forum on COCOMO and Software Cost Modeling*, Herndon, VA, 2006.

[4] Daryl P., Hindle A., Devanbu P., A simpler model of software readability, *Proceedings of the 8th working conference on mining software repositories*, ACM, 2011.

[5] Dhabhai D., Dua A.K., Saroliya A., Review paper: A Study on Metric For Code Readability, *International Journal of Advanced Research in Computer Science and Software Engineering*, Volume 5, Issue 6, June 2015.

[6] Duaa A., An empirical study of the relationships between code readability and software complexity, 2018.

[7] Gunning R., The Technique of Clear Writing. McGraw-Hill. pp. 36–37. 1952.

[8] Harry Mc.G., SMOG grading-a new readability formula, *Journal of reading*, vol. 12, no. 8, 1969, pp. 639-646.

[9] Hofmeister J., Siegmund J., Holt D. V., Shorter identifier names take longer to comprehend, *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Klagenfurt, 2017, pp. 217-227.

[10] Hyndman R.J., Koehler A.B., Another Look at Measures of Forecast Accuracy, *In International Journal of Forecasting*, pp. 679-688, 2006.

[11] Kate R.J., Luo X., Patwardhan S., Franz M., Florian R., Joseph R., Roukos S., Welty C., Learning to predict readability using diverse linguistic features, *In the proceedings of the 23rd International Conference on Computational Linguistics*, August 2010, pp. 546–554.

[12] Kincaid J.P., Fishburne R.P., Rogers R.L., Chissom B.S., Derivation of new readability formulas (automated readability index, fog count, and flesch reading ease formula) for Navy enlisted personnel. *Research Branch Report* 8–75. Chief of Naval Technical Training: Naval Air Station Memphis. 1975.

[13] Levin R. I. Statistics for Management, Pearson Education India, 2008.

[14] Lin J., Wu K., A Model for Measuring Software Understandability, *The Sixth IEEE International Conference on Computer and Information Technology (CIT'06)*, Seoul, 2006, pp. 192-192.

[15] Mark T., Mean Absolute Deviation. *Dynamic Portfolio Theory and Management*, 2004.

[16] Mccarthy P. , Dufty D., McNamara D., Toward a new readability: A mixed model approach, *In the proceedings of the 29th Annual Conference of the Cognitive Science Society*, 2007.

[17] McNamara D.S., Ozuru Y., Graesser A., Louwerse M., Validating coh-metrix, *In the proceedings of the 28th annual conference of the cognitive science society.* Mahwah, NJ: Erlbaum, 2006.

[18] Namani R., Kumar J., A New Metric for Code Readability, *IOSR Journal of Computer Engineering*, vol. 6, Issue 6, (2012) November-December.

[19] Nielebock S., Krolikowski D., Krüger J., Leich T., Ortmeier F., Commenting source code: is it worth it for small programming tasks?, *In Empirical Software Engineering*, vol. 24, 2019, pp. 1418–1457.

[20] Relf P. A., Tool assisted identifier naming for improved software readability: an empirical study, *2005 International Symposium on Empirical Software Engineering*, 2005.

[21] Scalabrino S., Linares-Vásquez M., Poshyvanyk D., Oliveto R., Improving code readability models with textual features, *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, Austin, TX, 2016, pp. 1-10.

[22] Scalabrino S., Linares-Vásquez M., Oliveto R., Poshyvanyk V, A comprehensive model for code readability, *Journal of Software: Evolution and Process*, vol. 30, no. 6, p. e1958, 2018.

[23] Senter R.J., Smith E.A., Automated Readability Index. Wright-Patterson Air Force Base: iii. AMRL-TR-6620. Retrieved March 18, 2012.

[24] Si L., Callan J., A statistical model for scientific readability, *In proceedings of the 10th international conference on Information and knowledge management*, October, 2001, pp. 574–576.

[25] Taek L.E.E., Jung-Been L.E.E., Effect analysis of coding convention violations on readability of post-delivered code. *IEICE TRANSACTIONS on Information and Systems*, vol. 98, no .7, 2015, pp. 1286-1296.

[26] Tashtoush Y., Odat Z., Alsmadi I., Yatim M., Impact of Programming Features on Code Readability, *In International Journal of Software Engineering and Its Applications*, vol. 7, 2013, pp. 441-458.

[27] Tufano M., Pantiuchina J., Watson C., Bavota G., Poshyvanyk D., On learning meaningful code changes via neural machine translation, *In the proceedings of the* 41^st *International Conference on Software Engineering*, IEEE Press, 2019.

[28] White M., Tufano M., Martinez M., Monperrus M., Poshyvanyk D., Sorting and Transforming Program Repair Ingredients via Deep Learning Code Similarities, *In the proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2019.

[29] Wu Y., Wang S., Bezemer C., Inoue K., How do developers utilize source code from stack overflow?. *Empirical Software Engineering*, vol. 24, no. 2, 2019, pp. 637-673.

Received 21.01.2020, Accepted 27.04.2021