International Journal of Computer Science and Informatics

Volume 1 | Issue 3 Article 13

January 2012

Developing A Code Readability Model To Improve Software Quality

Venkatesh Podugu
Department of CSE, JNTUA College of Engineering, Anantapur, Andhra Pradesh, India,
venkatesh.podugu@gmail.com

Recommended Citation

Podugu, Venkatesh (2012) "Developing A Code Readability Model To Improve Software Quality," *International Journal of Computer Science and Informatics*: Vol. 1 : Iss. 3 , Article 13.
DOI: 10.47893/IJCSI.2012.1039
Available at: https://www.interscience.in/ijcsi/vol1/iss3/13

Available online at www.interscience.in

Developing A Code Readability Model To Improve Software Quality

Venkatesh Podugu
Department of CSE, JNTUA College of Engineering, Anantapur, Andhra Pradesh, India
E-mail : venkatesh.podugu@gmail.com

Abstract **-** Software maintenance is one of the main phase in software evaluation. This paper presents the relation between software metrics and maintainability. This paper explains about the concept of Software code readability and its relation to software quality. The quality of code is very essential for the future and for the reuse purpose. Here generated a code readability model to calculate the readability of the code by selecting the snippets and these snippets are to be given to the expert to rate them. Collecting the features of code and combing the judgments generated the readability model.

This paper focus on providing the graphical user interface (GUI),to the code readability model to improve the understanding of software code readability. By providing the readability of code to the many open source projects, automatically informing the existed code quality to improve the quality of code. It show that this readability model developed is correlates strongly with three measures of software quality: code changes in software, defect log messages and automated defect reports. It measures correlations over many releases of selected projects.

Keywords— *Software Readability, Software maintenance, Code Readability, Software Quality, Code Metrics, Software Evaluation.*

.

I. INTRODUCTION

In software systems maintenance activities are important. Many reports are estimated that nearly 50% to 60% of development life cycle spent on maintenance, because evolving software by inducing changes[1]. The changes are made to the many software projects to improve the Quality, maintainability and for the readability. If the software contains the above mentioned attributes then the software automatically have long life. The readability of software code or document is related to maintainability.Readability is simply the ease of reading and understanding a code.

Program readability is actually a judgment about the understanding of a program [6]. In an academic environ-ment where students use example programs as a learning tool, readability of the examples must be good and helpful in understanding. The research on readability of ordinary text shows that a text with low readability is difficult to unde-rstand as well. The same principle can be applied on computer programs as, a program that is hard to read, is likely to be hard to understand.

A computer program or software, in terms of a set of instructions, is a special type of text with its own semantics and syntactic rules. The program semantics and syntactic rules are defined by a particular programming language. A hypothesis behind program readability is that if we treat a program code as a plain text then we can apply the text readability measures on computer programs as well. This means the idea of ordinary text readability can be extended to program readability, and we can use the well established and validated readability formulas to measure program readability as well. The results obtained from program readability measures then can be used to predict difficulty or complexity index of a program, i.e., how much effort or time will be required to read or understand a program. It will be useful in academic perspective as well. It may help the teachers and students to evaluate example programs quoted in text books or lecture notes to match their level of readability.

Here, presented a descriptive model of software readability based on simple features that can be extracted automatically from programs. This model of software reada-bility correlates strongly with human annotators and also with external (widely available) notions of software quality, such as defect detectors and software changes.

To understanding the usefulness of the objective mo-del of software readability, we have to consider the reada-bility metrics in natural languages. A number of readability measure and formulas were defined, but only few succeeded to conform validation standards. Few of the most popular readability formulas include: Flesch's Reading Ease Score [3], Dale-Chall's Readability Formula [4], SPACHE Readability Formula, FryGraph Readability Formula, SMOG Grading, Cloze Procedure, Lively-Pressey's Formula and Gunning's Fog Index (or FOG).

The Flesch formula is one of the most successful readability formulas designed to measure readability for paragraphs [2]. It was proposed by Rudolph Flesch [3], a writing consultant and well known supporter of the Plain English Movement. Simplicity and accuracy are the two main characteristics of the Flesch reading ease score. Unlike other formulas, it is easy to calculate and is regarded as more accurate readability index. Total number of words, syllables and sentences are the basic counts of the formula. Then it uses average sentence length and average number of syllables per word to compute a final readability score for a given text. The original Flesch Reading Ease Formula is as below:

R:E: = 206.835 - (0.846 *wl) - (1.015 * sl)

Here:

R.E. = Reading Ease

wl = Word Length (The number of syllables in a 100 word sample).

sl = Average Sentence Length (the number of words divided by the number of sentences, in a 100 word sample).

Below is the modified form of the formula in case of text having more than 100 words:

R:E: = 206.835 - (84.6 * ASW) - (1.015 *ASL)

Here:

ASW = Average Number of Syllables per Word (total number of syllables divided by the total number of words).

ASL = Average Sentence Length (the number of words divided by the number of sentences).

Constants in the formula are selected by Flesch after years of observation and trial [8].

The R.E. value ranges from 0 to 100 and higher value implies easier the text is to read. Abram and Dowling [8] use following interpretations for FRES, originally specified by Klare and Campbell.

The above mentioned is one example for the natural language readability metrics. These metrics can help organizations gain some confidence that their documents meet goals for readability very cheaply, and have become ubiquitous for that reason. We believe that similar metrics, targeted specifically at source code and backed with empirical evidence for effectiveness, can serve an analogous purpose in the software domain.

Most of the classical readability formulas, including FRES, are based on the count of lexical tokens or entities, e.g., total number of words, unique words, sentences, syllables, paragraphs. In order to apply readability formulas to computer programs, one have to find the equivalents of these lexical entities for a program text. Programming languages at present are not exactly same as natural languages are, however the basic lexical units are similar. They have their own set of characters equivalent to alphabets, keywords and user defined identifiers equivalent to words, statements equivalent to sentences, block structures equivalent to paragraphs or sections, and modules equivalent to chapters. Static code analysis techniques and tools can be used to count these static code elements.

Program complexity and readability are closely interrelated, however they are not exactly same. Complexity is an intrinsic or essential code property based on the problem domain and it can not be avoided completely in all scenarios. Whereas readability is an accidental property that can be avoided independent of the problem domain or problem complexity [6]. Readability is a static measure based on independent individual elements, such as identifiers, statements, comments, indentation, and program style. Whereas complexity depends on both static com-ponents as well dynamic interactions among program components.

In this paper, concentrated on the two readability models for the software code readability and their comparisons among them for the judging the software code readability.

II. SOFTWARE READABILITY EASE SCORE - SRES

As the name implies, Software Readability Ease Score (SRES) is a measure of program readability proposed by Borstler et al.. The basic idea behind SRES is Flesch's reading ease score [3], and we can say SRES is a software oriented extension of FRES. SRES works by interpreting program's identifiers, keywords and other lexical tokens as words, its statements as sentences, and word's length as syllables. A program readability formula can be defined on the basis of number of lexemes, statements, and modules declared in a program. At present we use Average Sentence Length (ASL) and Average Word Length (AWL) as program readability index. Lower values for ASL and AWL imply the program is easier to read, because of shorter sentences and words length; and higher values indicate the program is difficult to read and understand.

A. SRES Measurement Tool:

In this explains an alpha version of the SRES measurement tool, developed using a parser generator tool, ANTLR, and Java platform. The current version implements Halstead's measures of software science and measures of software readability ease scores[5]. Main objective of this study is to determine and evaluate the quality of commonly used java example programs. SRES measurement tool developed is a standalone java source code analyzer. It reads java programs and displays

results for the implemented metrics of SRES and Halstead. At present it supports Java 1.5. It follows static code analysis approach performed with the help of Java Lexer, Parser, and Tree Parser components, automatically generated using ANTLR, parser generator tool.

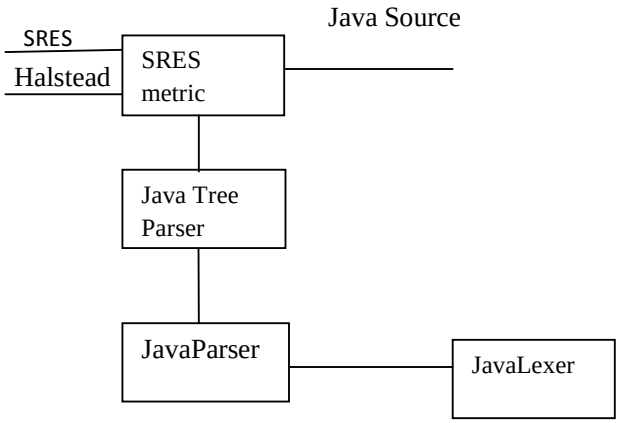


Figure 1: SRES – Diagram SRES Metrics:

This component is responsible to implement software readability ease score, halstead measure; and any other software metrics added in future. It also provides user interface to the SRES tool. As shown in the figure 1, it requires source java file as input and provides Halstead and SRES metrics results as output. The package edu.cs.umu.sres" in the source code contains all the implementation classes for this component, where Main.java provides user interface, while SRES.java and Halstead.java implements corresponding software metrics.

Java Lexer:

Lexer also known as a scanner or lexical analyzer is a program component that recognizes input stream of characters and breaks it into a set of vocabulary symbols or tokens. A token object returned by the lexer may represent more than one characters of same type, for example INT token represent integers. Each tokens consist of at least two pieces of information: the token type (lexical structure) and the text matched by the lexer [7]. These tokens are then pulled and used by the parser component. It implementation in source code is provided by JavaLexer.java class that is automatically generated by ANTLR against the lexical rules defined in the Java.g grammar file. Lexer rules match characters on the input stream and return a token object automatically. Below is an excerpt from lexer grammar, that defines lexer rules for hexadecimal, decimal and octal literals.

Java Parser:

Parser is simply a language recognizer that. It applies grammatical structure defined as parser rules to a stream of tokens(vocabulary symbols)received from the lexer com-ponent. Both Lexer and Parser perform similar task, the difference is that the parser recognizes grammatical structure in a stream of tokens while the lexer recognizes structure in a stream of characters [7]. Along with language recognition, parser has the ability to act as a translator or interpreter, and can generate appropriate output or an intermediate data structure, usually a parse tree or abstract syntax tree (AST).

Java Tree Parser:

ANTLR has the ability to generate a tree parser automatically from a tree grammar. Tree grammar actually describe the structure of the tree (AST built by the JavaParser component). In order to evaluate program statements and compute the corresponding metrics values, actions are embedded as in-line code segments surrounded by curly braces f...g, as shown in the tree grammar excerpt below. This component is implemented by JavaTreeParser.java class automatically generated by ANTLR using a tree grammar file JavaTreeParser.g. It traverses a two-dimensional abstract syntax tree and computes the metrics values for each node matched in the AST. Below is an excerpt from tree parser grammar JavaTreeParser. grammar file.

III. DEVELOPING READABILITY MODEL AND PROVIDING "GUI" TO IT

This is the work paper dealing with generating a reada-bility model, and to generated model incorporated into integrated development environment such as Netbeans and Eclipse.

A. Selecting the suitable snippets

In the generation of readability model, first coll-ected the 100 snippets from the some open source java projects which are available at SourceForge, which is an open source software repository. Snippet is small part of the code. A snippet does include preceding or in-between lines that are not simple statements, such as comments, function

Headers, blank lines, or headers of compound statements like if-else, try-catch, while, switch, and for. These snippets must be too short to aid feature discrimination. However, if snippets are too short, then they may obscure important readability considerations. Second, snippets should be logically coherent to allow annotators the context to appreciate their readability. These snippets are given to the annotators, these are the people who can write the functionality of the code.

B. *Scoring the readability*

The annotators were asked to give ratings to the snippets in given order from 1 to 5. The participants were given ratings according to if the code is "more readable" they given near to 5, if less they given 1or 2, if in the average case then they given as the 3. According to given instructions they are gave ratings for the 100 snippets in the given order.

C. Study on snippts

The study was taken on computer science Courses at The University of JNTU, Students are participated on judging the readability. Participants had different experience on reading and writing code: 27 were taking first-year courses, 33 were taking second-year courses, 50 were taking third or fourth-year courses, and40 were graduate students. In total, 150 students participated. These results are to be Our 150 annotators each annotator scored 100 snippets for a total of 15,000 different judgments. After applied various correlation statistics on the obtained results to measure the relation on them. Later we applied several correlation statistics on these results to measure the relationship among them.

D. Model Generation

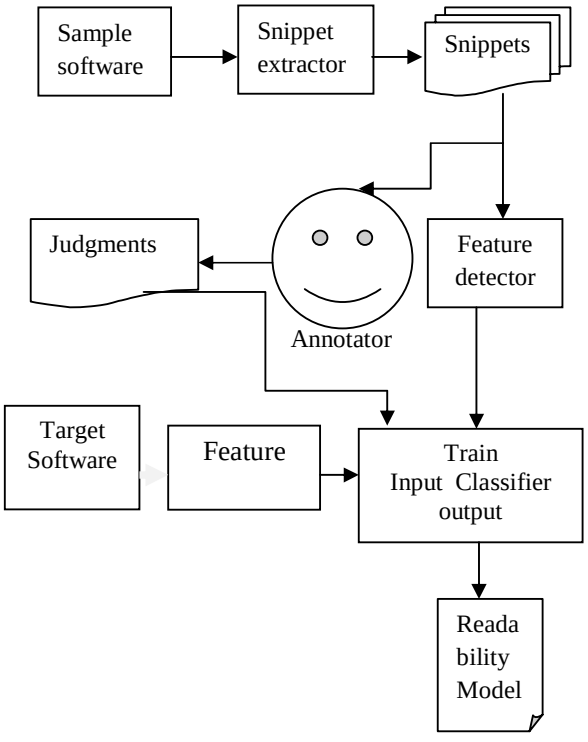


Fig 2: Readability model

First, forms a set of features that can be detected statically from a snippet or other block of code. For any code it contains some of local code features those are to be Line length (# character), identifiers, identifier length, Indentation (preceding whitespace), Keywords, Parenthesis, Numbers, Comments, Periods, branches, loops like wise nearly 18 features are there. Each feature can be applied to an arbitrary sized block of Java source code, and each represents either an average value per line, or a maximum value for all lines. For example, we have a feature that represents the average number of identifiers in each line and another that represents the maximum number in any one line. There are several machine learning algorithms are available for this situation. Such algorithms typically take the form of a classifier which operates on instances. For our Purposes, an instance is a feature vector extracted from a single snippet. In the training phase, we give a classifier a set of instances along with a labeled "correct answer" based on the readability data from our annotators. The labeled correct answer is a binary judgment partitioning the snippets into "more readable" and "less readable" based on the hu-man annotator data. We group the remaining snippets and consider them to be "more readable." Furthermore, the use of binary classifications also allows us to take advantage of a wider variety of learning algorithms{[9]. After making the training and testing phases we generated a readability model. Using this readability the readability of the code is calculated. The readability is to be comes between 0-1, means a fractional value[10].

The readability model which is to be developed is to be incorporated into the graphical user inter phase such as to be NetBeans or Eclipse we can easily understand the readability and we can be generate graphs to the readability of the code which is to be taken to calculate the readability.

The graphical representation is to be for the better understanding purpose. NetBeans and Eclipse are to be the IDEs (Integrated Development Environment), and if we incorporate this model into the IDEs, we can make more friendliness to the users to use the readability model in nature. Many organizations can be use this to check their code readability. If code readability is less then automatically the quality of the code also to be less. Readability and quality both are to be inter related in nature. If readability is less then they try to increase the readability of the code by changing the code. Then automatically quality of the code also increases.

Any one can automatically judge readability about as well as the "average" human can. This notion of readability shows significant correlation with:

1. Version Change
2. The output of a bug finder.
3. Self-reported program maturity.

These metrics can help organizations gain some confidence that their documents meet goals for readability very cheaply.

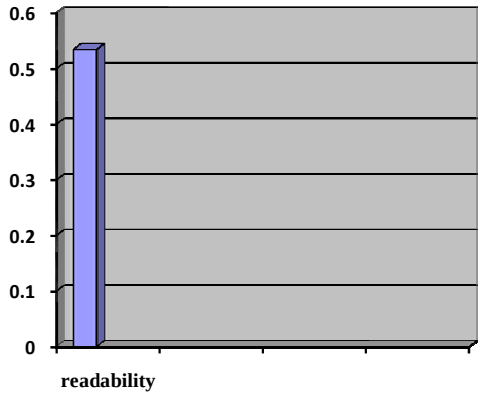
IV. EXPETIMENTAL RESULTS

An experiment is to be conducted on to the a small part of the java code called snippet. This experiment is conducted using the IDE as Netbeans to calculate the readability of the code. For this experiment given the snippet as

```
Class clas = object.getClass();

Field field = Reflect.resolveJavaField

( clas, name, false/*onlyStatic*/ );

 if ( field != null )

return new Variable(

name, field.getType(), new LHS(  object,
field ) );
```

The above used a snippet from the java code and this is used as a input to my model and the output generated is the readability score. The readability score is

0.5342345566



The above ment1oned graph is to be the calculated readability of the given snippet. Using the model like above it can be calculated for any snippets

V. CONCLUSIONS AND FURTHER WORK

In this paper explained about the different metrics available for the both natural languages and for the software code which is to be created. For natural languages there are already existed many readability metrics and those metrics are already incorporated into the word processors. Here explained about the readability metrics are to be SRES and code readability model which is generated The code readability model is to be incorporated in to the IDES for the more useful purpose for the users and for the future purpose. The model mentioned above is not exactly correct one, why because ii is constructed by taking the opinions of the some people. So in future taking the opinions of experts and the increasing some more features can construct the improved and more efficient model.

REFERENCES

[1] B. Boehm and V.R. Basili, "Software Defect Reduction Top 10 List," Computer, vol. 34, no. 1, pp. 135-137, Jan. 2001.

[2] M.J. Abram and W.D. Dowling. How Readable are Parenting Books ? Family Coordinator, pages 365 {368, 1979.

[3] R. Flesch. A New Readability Yardstick. The Journal of Applied Psychology, 32(3):221 1948

[4] E. Dale and J.S. Chall. A Formula for Predicting Readability. Educational Research Bulletin, pages 11{28, 1948.

[5] J.Borstler,M.E.Caspersen, and M. Nordstrom. Beauty and the Beast{Toward a Measurement Framework for Example Program Quality. (UMINF-07.23), 2007.

[6] Raymond P.L. Buse and Westley R. Weimer. Learning a metric for code readability. Ieee Transactions On Software Engineering, Vol. 36, No. 4, July/August 2010

[7] T.Parr. The Definitive ANTLR Reference. The Pragmatic Programmers (May 2007)

[8]. M.J.Abram and W.D. Dowling. How Readable are Parenting Books? Family Coordi-nator, pages 365{368, 1979.

[9] G.R. Klare. The measurement of readability: useful information for communicators.ACM Journal of Computer Documentation (JCD), 24(3):107{121, 2000

[10] W.H.DuBay. The Principles of Readability. Impact Information, pages 1{76, 2004}

❑❑❑