A Method for Software Metrics Assessment to Heighten the Quality of Source Code

Jitendra Singh Brar [1] Dr. Vishal Goar[2] Prof. S.S. Sarangdevot[3]

[1] Research Scholar, JRN Rajasthan Vidyapeeth, Udaipur
[2]Assistant Professor, Govt. Engineering College Bikaner
[3]Vice-Chancellor, JRN Rajasthan Vidyapeeth, Udaipur
[1]jbrar2010@gmail.com, [2]dr.vishalgoar@gmail.com, [3]prof.sarangdevot@gmail.com

Abstract. The Source Code quality and understandability entirely depends on the comments specified at appropriate locations. The current paradigms do not propose any metric or methodology that is useful for checking the source comments quality. The proposed model and empirical parser based implementation emphasize the efficient and meaningful usage of code comments in the source code regardless of the language or script. In this research work, the empirical and pragmatic evaluation of the source understandability and the escalation is done using survey based analytics. The source code comments and understanding factors are taken into the consideration so that the reusability of the source code can be increased. The key parameters for evaluation of source code include cohesion, coupling and the type of source code. The type of source code is having the flavor of procedural or object oriented paradigm so that any type of source code with its inherent feature points can be analyzed and predicted.

Keywords: Source Code Understanding, Software Metrics, Software Quality, Software Reusability.

1 Introduction

The Source Code quality and understandability entirely depends on the comments specified at appropriate locations. The current paradigms do not propose any metric or methodology that is useful for checking the source comments quality. The proposed model and empirical parser based implementation emphasize the efficient and meaningful usage of code comments in the source code regardless of the language or script.

The source code written in any programming language or script should be modifiable and based on the understandability of the source code. Once the program is made modifiable based on assorted metrics and parameters, any programmer can update it. This research work focus on the improvement of software code quality based on the escalation of code metrics.

2 Motive of the Evaluation

- To identify the factors affecting the source code quality
- To associate the code comments with the code quality
- To identify the locations and reasons to associate software quality with the comments
- To investigate the factors affecting the quality of the source code itself
- To frame out an empirical model for the improvement of the source code quality regardless of the programming language to be used
- To locate the factors and points that directly or indirectly affect the code quality
- To find out the association of code understandability with the complexity as well as related parameters
- To fetch and locate multiple parameters as well as the taxonomy identification
    - Stop Words
    - Bad Comments
    - Good Comments
    - Meaningless Comments
    - Nominal Comments
    - Too Many Stop Words
    - Unnecessary
    - With Stop Words
    - No Comments
- To develop an efficient parser for code investigation
- To qualify and embed the web based parser with deep code investigation features that includes the major parameters
- To analyze the correlation between code understandability and code parameters
- To investigate the correlation between comments and stop words
- To frame out the line density as well as word density that is useful and effective for better understandability of the source code.
- To design multiple mutants of the source codes for deep investigation without biasing.
- To associate the software code quality with the meaningful parameters that affect the overall performance of the software.

Such measures are software metrics introduced by Maurice Howard Halstead in 1977 as part of the treatise on establishing an empirical science of software development. Halstead makes the observation that metrics of the software should reflect the implementation or expression of algorithms in different languages, but be independent of their execution on a specific platform. These metrics are therefore computed statically from the code.Halstead's goal was to identify measurable properties of software, and the relations between them. This is similar to the identification of measurable properties of matter (like the volume, mass, and pressure of a gas) and the relationships between them (analogous to the gas equation). Thus his metrics are actually not just complexity metrics.Halstead complexity metrics were developed by the late Maurice Halstead as a means of determining a quantitative measure of complexity directly from the operators and operands in the module to measure a program module's complexity directly from source code. Among the earliest software metrics, they are strong indicators of code complexity. Because they are applied to code, they are most often used as maintenance metric. There is evidence that Halstead measures are also useful during development, to assess code quality in computationally-dense applications. Because maintainability should be a concern during development, the Halstead measures should be considered for use during code development to follow complexity trends. Halstead measures were introduced in 1977 and have been used and experimented with extensively since that time. They are one of the oldest measures of program complexity. Halstead's metrics is based on interpreting the source code as a sequence of tokens and classifying each token to be an operator or an operand.

3 Proposed Methodology and Formulation

In this research manuscript, the evaluation of software metrics associated with the software code is done so that the new parameters for understanding can be analyzed, fetched and measured. In this work, the comments in source code with the understanding factors are taken very carefully so that the overall reusability can be increased.

SCCM = (s+r-o-g)=> ((number of attributed shared in class/no of attributes in class)) + ((number of methods returning no value/total methods in class or function returning no value to the other function(1))) – ((number of methods returning value outside the class/total methods in class or function returning value to other function(1))) - ((total number of global or public variables / total variables in the class))

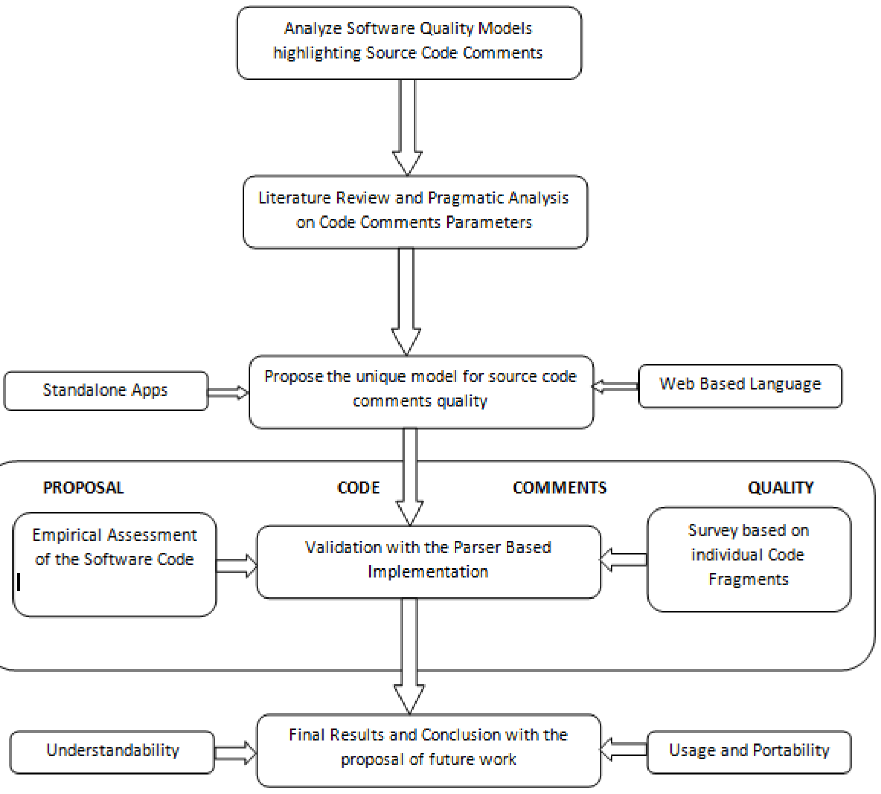
Analyze Software Quality Models highlighting Source Code Comments
Literature Review and Pragmatic Analysis on Code Comments Parameters
Standalone Apps
Propose the unique model for source code comments quality
Web Based Language
PROPOSAL
CODE
COMMENTS
QUALITY
Empirical Assessment of the Software Code
Validation with the Parser Based Implementation
Survey based on individual Code Fragments
Understandability
Final Results and Conclusion with the proposal of future work
Usage and Portability

Fig. 1. Evaluation Parameters and Flow of the Research Work

SCCM < X < N where N = Decision Parameters Used

SAMPLE SOURCE CODE : CPGM2.C

=> Main	: 1
=> Generate_armstrong :	: 0.8
=> Find_npr	: -1.6
=> Find_ncr :	: -1.6
=> gcd	: -2
=> Calculate_LCM_HCF :	: 0 :
=> Display_pascal	: 0.66
=> Display_floyd_triangle	: 0.75
=> Frequency_chars :	: 1
=> Upper_String	: 0
=> Lower_string	: 0
=> Sort_String	: 0.86
=> Substring	: 0.4
=> Concatenate_string	: 0

COUPLING METRIC=> ((total shared variable/total variable) +(total shared function including public/total function) +(if ci inherits cj then 1 else 0;)
1 < coupling metric < n

CM=(s+p+i)

CODE : CPGM2.C

=> Main	: 0
=> Generate_armstrong :	: 0.2
=> Find_npr	: 0.6
=> Find_ncr :	: 0.6
=> gcd	: 1
=> Calculate_LCM_HCF :	: 1
=> Display_pascal	: 0.33
=> Display_floyd_triangle	: 0.25
=> Frequency_chars :	: 0
=> Upper_String	: 1
=> Lower_string	: 1
=> Sort_String	: 0.14
=> Substring	: 0.6
=> Concatenate_string	: 1

4 Results and Findings from Reusability and Code Quality

In this research work and implementation aspects, more than 50 source code files of different paradigms including procedural and object oriented methodology are evaluated on multiple parameters so that different type of programs can be evaluation on the base of understanding and reusability.

Table 1. Evaluation of Source Code in Procedural Implementation

Cohe-sion	Coupl-ing	Understandability (In Minutes) After Survey	Reusability (1-10) => (Co-hesion/Coupling)*K where K=10	Class / Module Name
1	0	38	0	Main
0.8	0.2	20	40	Generate_Armstrong
-1.6	0.6	50	-26.66666667	Find_npr
-1.6	0.6	50	-26.66666667	Find_ncr
-2	1	55	-20	GCD
0	1	20	0	Calculate_LCM_HCF
0.66	0.33	15	20	Display_pascal
0.75	0.25	40	30	Display_floyd_triangle
1	0	30	0	Frequency_chars
0	1	32	0	Upper_String
0	1	34	0	Lower_string
0.86	0.14	60	61.42857143	Sort_String
0.4	0.6	20	6.666666667	Substring
0	1	37	0	Concatenate_string

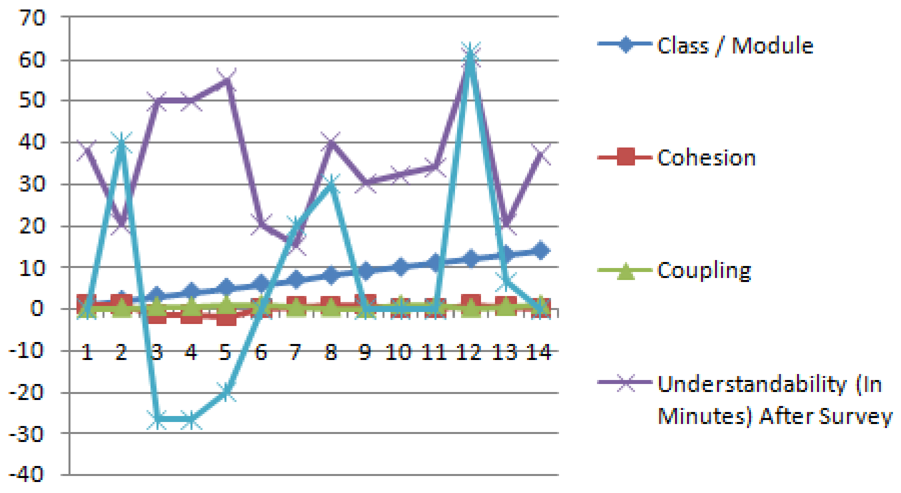

Fig. 2. Evaluation of Cohesion, Coupling and Understandability in the Source Code

Table 2. Evaluation of Statistical Parameters

Reusability (1-10) => (Cohesion/Coupling)*K where K=10	Class / Module Name	Efforts Level
0	Main	2
40	Generate_Armstrong	2
-26.66666667	Find_npr	1
-26.66666667	Find_ncr	1
-20	GCD	1
0	Calculate_LCM_HCF	2
20	Display_pascal	2
30	Display_floyd_triangle	2
0	Frequency_chars	2
0	Upper_String	2
0	Lower_string	2
61.42857143	Sort_String	3
6.666666667	Substring	2
0	Concatenate_string	2

Table 3. Evaluation of Source Code with Object Oriented Paradigm

Cohesion	Coupling	Understandability (In Minutes) After Survey	Reusability (1-10) => (Cohesion/Coupling)*K where K=10	Class / Module Name
1	2	30	5	Person
1	3	20	3.333333333	Account
1	3	20	3.333333333	Admin
1	2	20	5	Master

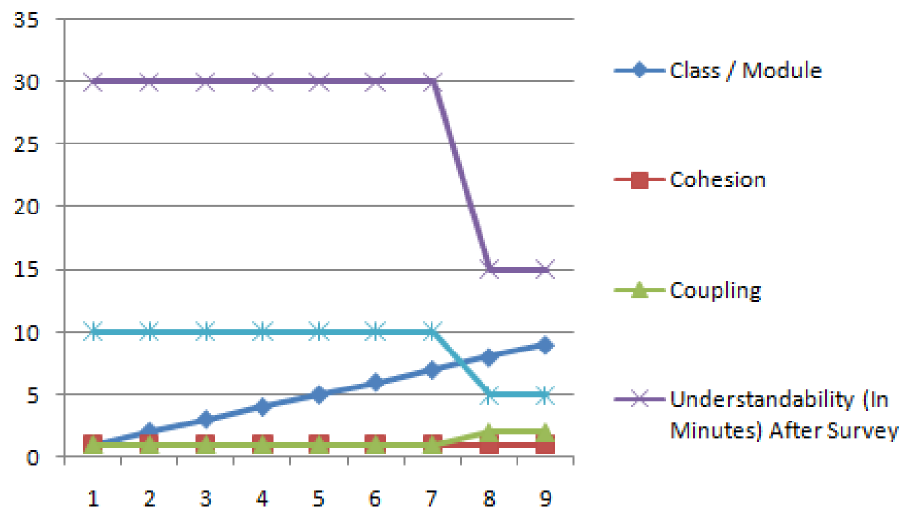

Fig. 3. Evaluation of Class in the Object Oriented Source Code with Understanding Factors

After empirical survey and metric results, it is evident that the reusability of the source code is directly dependent on the understandability. We have conducted the pragmatic survey of different source codes (procedural and object oriented) and it is analyzed that understandability is vital in finding out the reusability of the source code.

Conclusion

In this research manuscript, the evaluation of source code for understandability and related reusability is done. Once the source code is found understandable with the proper and sufficient number of comments with the indents, it is easy for new developer to evaluate the flow of code and easily new modules can be inserted. By this implementation and approach, the overall integrity of code can be increased. In this work, the source code reusability is associated because the source code and final developed software can be made reusable is that is easy to understand by the developer for new updates and insertions with new developed plugins and modules.

References

1. T. Tenny "Program Readability: Procedures Versus Comments " IEEE Trans. Softw. Eng. vol. 14 no. 9 1988.
2. S. N. Woodfield H. E. Dunsmore and V. Y. Shen "The effect of modularization and comments on program comprehension " ser. ICSE '81 1981.
3. S. C. B. de Souza N. Anquetil and K. M. de Oliveira "A Study of the Documentation Essential to Software Maintenance " ser. SIGDOC '05 2005.

4. C. S. Hartzman and C. F. Austin "Maintenance productivity: Observations based on an experience in a large system environment " ser. CASCON '93 1993.
5. B. P. Lientz "Issues in Software Maintenance " ACM Computing Surveys vol. 15 no. 3 1983.
6. M. J. B. García and J. C. G. Alvarez "Maintainability as a Key Factor in Maintenance Productivity: A Case Study " ser. ICSM '96 1996.
7. P. Oman and J. Hagemeister "Metrics for Assessing a Software System's Maintainability " ser. ICSM '92 1992.
8. I. S. Microsystems Code Conventions for the Java Programming Language 1997. [Online]. Available: http://www.oracle.com/ technetwork/java/codeconv- 138413.html
9. N. Khamis R. Witte and J. Rilling "Automatic Quality Assessment of Source Code Comments: the Java-docMiner " ser. NLDB '10 2010.
10. M.-A. Storey J. Ryall R. I. Bull D. Myers and J. Singer "TODO or To Bug: Exploring How Task Anno-tations Play a Role in the Work Practices of Software Developers " ser. ICSE '08 2008.