International Journal of Computing and Business Research (IJCBR)
ISSN (Online) : 2229-6166
International Manuscript ID : 22296166V7I3201705
Volume 8 Issue 1 January 2018

Evaluation of Software Design Complexity using Halstead Metrics

Ashwani Sethi[1], Surinder Kumar Sharma[2],

Professor, Guru Kashi Unversity[1]

Research Scholar, Guru Kashi University[2]

Abstract

The evaluation of software quality and audit from multiple perspectives is an important task before actual deployment so that the bugs, complexity and overheads can be evaluated in prior. The successful running of source code is always not sufficient because the code complexity and related performance issues are also required to be integrated for cumulative results. In this research manuscript, the complexity measures associated with the software are analyzed with the empirical results using Halstead's metrics used for complexity. In traditional Halstead metrics, the use of program vocabulary, length and difficulty levels are processed which are not sufficient as per the current paradigms of the programming using advance tools. Now days, most of the work is done using object oriented programming languages and therefore the improvements are proposed in the traditional Halstead metrics with object oriented paradigms. The projected results are found effectual as compared to the classical approach on multiple parameters.

Keywords: Code Coverage, Code Evaluation, Halstead Metrics, Object Oriented Enabled Halstead Metrics, Software Complexity

Introduction

Software Design and Code Metrics is one of the prominent areas of research in the segment of software engineering [1]. In this domain, the deep perspectives of the source code written for a specific software tool are analyzed so that the resource consumption and finally optimization can be done [2]. The execution of source code consumes enormous system resources including memory, processor and time which degrade the overall performance if not taken care [3].

Following are the key measurements and indexes used while analyzing the quality of code and overall software design

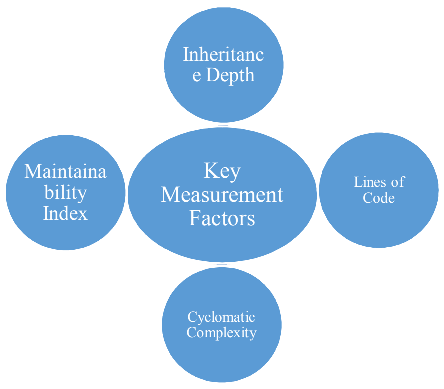

Figure 1: Key Measurement Factors for Evaluation of Software Code and Design

Maintainability Index

It ensures the understandability and reusability of the source code. This value ranges from 0 to 100 in terms of the index value. Higher value signifies the higher degree of maintainability [4].

Cyclomatic Complexity

It evaluates the structural complexity of the source code so that the different constituents of source code can be measured with the flow of code [5].

Inheritance Depth

This aspect evaluates the depth of the inheritance in the functions and classes of the source code so that overall dependency of modules can be evaluated [6].

Lines of Code

It signifies the lines of code which are executed by the compiler or interpreter. It is always desired to write the optimized code with less number of lines so that overall overhead can be reduced [7].

Halstead Metricsfor Evaluation of Complexity

The original performance of a software design is associated with the assessment of complexity measures and metrics. Simply development of code and testing using automation tools are not sufficient because only these perspectives can increase the overall overhead on different resource of the system. The system resources which are directly affected by the software design and code are Memory, Processor, Execution Time, Dependent Libraries and many others.

In year 1977, M. H. Halstead devised the metrics for the measurement and evaluation of software complexity using different code components and categories [8]. This metrics is more focused towards the implementation of program code based on the classical components including Operators, Operands and their relative occurrences. That was the time when the Object Oriented Programming (OOP) was not prominent.

Key elements and constituents of the Halstead metrics include the following

Table 1. Indicators and Elements of Halstead Metrics

Element or Indicator	Description
n1	Number of unique operators
n2	Number of unique operands
N1	Number of total occurrence of operators
N2	Number of total occurrence of operands

Table 2. Metrics Report from the Viewer of Source Code

Parameter	Metric Element	Notation
Vocabulary	n	n1 + n2
Size	N	N1 + N2
Volume	V	Length * Log2 Vocabulary
Difficulty	D	(n1/2) * (N1/n2)
Efforts	E	Difficulty * Volume
Errors	B	Volume / 3000
Testing time	T	Time = Efforts / S, where S=18 seconds.

Simula is considered as the first programming language that was object oriented programming language but its popularity escalated in far ahead decades. The Halstead Metrics was lacking on the perspectives of including the OOP based components including virtual functions, friend functions, pointers, classes, constructors, destructors and many others.

Table 3. Elements in Improved Halstead Complexity Metrics

Element or Indicator in Improved Halstead	Programming Paradigm
distinct operators	Hybrid (Procedural, OOP)
total operators	Hybrid (Procedural, OOP)
distinct unique operands variables constants	Hybrid (Procedural, OOP)
number of operands variables constants	Hybrid (Procedural, OOP)
number of struct used	Hybrid (Procedural, OOP)
number of classes	OOP
number of constructors destructors	OOP
lines of code	Hybrid
comment lines	Hybrid
friend functions	OOP
virtual functions	OOP
file pointers	Hybrid (Procedural, OOP)

In this work, the integration of OOP based ingredients to the classical Halstead Metrics is proposed and implemented along with the prevalent objects of Halstead metrics. Following is the log of results obtained from the simulation scenario created and found that the proposed approach is effectual as compared to the classical approach of Halstead metrics.

```
Objects Evaluation
(
distinct operators  (DO) => 5
operators  (O) => 8
distinct unique operands (UO) => 5
operands variables constants  (OV) => 9
struct(S) => 0
classes  (C) => 1
constructors destructors  (CD) => 2
lines of code  (LOC) => 23
comment lines  (CL) => 7
friend functions  (FF) => 2
virtual functions  (VF) => 0
file pointers  (FP) => 0
)
Program Vocabulary (n) => 18
Program Length (N) => 12
Program Difficulty (D) => 8.5
Calculated Program Length (N) => 78.43
Volume (V) => 401.323
Effort (E)=> 124.743
Volume (V) in Improved Halstead Metrics
=> 420.921
Effort (E) in Improved Halstead Metrics
=> 130.383
Execution Time in Classical Approach
(Microseconds) : 0.02312 ms
Execution Time Proposed Approach
(Microseconds) : 0.0182
```

As per the results, the execution time and complexity is found less in the proposed approach and integrity aware results are projected in terms of program length and the efforts.

Conclusion

Quality of the source code is an important task for the software developers rather than simply generating the compiled code. The optimization factors are always considered by the test evaluation of source code and overall design so that the dependency factors and related complexities can be evaluated. The evaluation and optimization of design is important so that the resource consuming perspectives can be reduced and taking care of the important constituents which are required to execute the code towards final product. In this work, the specific focus on the Halstead metrics is given with the proposed elements of object oriented paradigms in the software code evaluations rather than traditional components. The results found in the simulation are effectual as compared to the traditional perspectives.

References

[1] Jiang Y, Cuki B, Menzies T, Bartlow N. Comparing design and code metrics for software quality prediction. InProceedings of the 4th international workshop on Predictor models in software engineering 2008 May 12 (pp. 11-18). ACM.

[2] Mahmoud SS, Ahmad I. A green model for sustainable software engineering. International Journal of Software Engineering and Its Applications. 2013 Jul;7(4):55-74.

[3] Abdelzaher TF, Stankovic JA, Lu C, Zhang R, Lu Y. Feedback performance control in software services. IEEE Control Systems. 2003 Jun;23(3):74-90.

[4] Coleman D, Ash D, Lowther B, Oman P. Using metrics to evaluate software system maintainability. Computer. 1994 Aug;27(8):44-9.

[5] Gill GK, Kemerer CF. Cyclomatic complexity density and software maintenance productivity. IEEE transactions on software engineering. 1991 Dec;17(12):1284-8.

[6] Daly J, Brooks A, Miller J, Roper M, Wood M. Evaluating inheritance depth on the maintainability of object-oriented software. Empirical Software Engineering. 1996 Jan 1;1(2):109-32.

[7] Binkley AB, Schach SR. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. InSoftware Engineering, 1998. Proceedings of the 1998 International Conference on 1998 Apr 19 (pp. 452-455). IEEE.

[8] Kearney JP, Sedlmeyer RL, Thompson WB, Gray MA, Adler MA. Software complexity measurement. Communications of the ACM. 1986 Nov 1;29(11):1044-50.

[9] Bowes, D., Hall, T., & Petrić, J. (2017). Software defect prediction: do different classifiers find the same defects?. Software Quality Journal, 1-28.

[10] Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., Galster, M., & Avgeriou, P. (2017). A mapping study on design-time quality attributes and metrics. Journal of Systems and Software, 127, 52-77.

[11] Czech, G., Dorninger, B., Pfeiffer, M., Moser, M., & Pichler, J. (2017). Software Analytics and Evolution Team Report 2016.