International Journal of Discrete Mathematics
2017; 2(3): 100-106
http://www.sciencepublishinggroup.com/j/dmath
doi: 10.11648/j.dmath.20170203.17

SciencePG
Science Publishing Group

Conference Paper

Effective Approach for Code Coverage Using Monte Carlo Techniques in Test Case Selection

Varun Jasuja[1], Rajesh Kumar Singh[2]

[1]Computer Science and Engineering, Guru Nanak Institute of Technology, Ambala, India

[2]Computer Science Application, SUS Institute of Computer, Tangori, India

Email address:
varunjasuja19@gmail.com (V. Jasuja), rajeshkripal@gmail.com (R. K. Singh)

To cite this article:
Varun Jasuja, Rajesh Kumar Singh. Effective Approach for Code Coverage Using Monte Carlo Techniques in Test Case Selection. *International Journal of Discrete Mathematics.* Vol. 2, No. 3, 2017, pp. 100-106. doi: 10.11648/j.dmath.20170203.17

Received: February 20, 2017; **Accepted**: March 13, 2017; **Published**: March 29, 2017

Abstract: Source code analysis alludes to the profound examination of source code and/or gathered form of code with a specific end goal to help discover the imperfections as far as security, comprehensibility, understanding and related parameters. In a perfect world, such systems consequently discover the defects with such a high level of certainty that what's found is surely a blemish. Notwithstanding, this is past the best in class for some sorts of utilization security defects. In this manner, such devices much of the time serve as helps for an examiner to help them focus in on security pertinent segments of code so they can discover blemishes all the more productively, instead of a device that just consequently discovers imperfections. Code Coverage is a measure used to portray the extent to which the source code of a system is tried by a specific test suite. A project with high code scope has been all the more completely tried and has a lower shot of containing software bugs than a system with low code scope. A wide range of measurements can be utilized to ascertain code scope; the absolute most fundamental are the percent of system subroutines and the percent of project articulations called amid execution of the test suite. This research work focus on the quality of source code using code coverage and analysis techniques. In the proposed research work, an effective model based approach shall be developed and implemented to improve the performance of code in terms of overall code coverage time, code complexity and related metrics.

Keywords: Code Coverage, Software Testing, Automated Test Case Generation

1. Introduction

The software metrics review is organized in two ages: before 1991, where the main focus was on metrics based on the complexity of the code and after 1992, where the main focus was on metrics based on the concepts of Object Oriented (OO) systems. There is no formal concept to source code quality. According to [1], there are two ways to source code quality: technical and conceptual definition. The technical definition is related to many source code programming style guides, which stress readability and some language-specific conventions are aimed at the maintenance of the software source code, which involves debugging and updating. Therefore, the conceptual definition is related to the logical structuring of the code into manageable sections and some quality attributes like: readability, maintenance, testing, portability, complexity and others.

This work will evaluate the state-of-the-art in software metrics related to source code, which the main goal is to analyze the existents software metrics and verifies the evolution of this area and why some metrics couldn't survive. Thus, it can be understood how the source code quality evaluates throughout of the years. However, this work does not cover other software metrics related to performance [2], productivity [3], among others [4] [5] [6].

Quality is a phenomenon which involves a number of variables that depend on human behavior and cannot be controlled easily. The metrics approaches might measure and quantify this kind of variables. Some definitions for software metrics can be found on the literature [7] [8] [9]. In agreement with Daskalantonakis in [9] it is found the best motivation to measures, it is finding a numerical value for some software

product attributes or software process attributes. Then, those values can be compared against each other and with standards applicable in an organization. Through these data could be draw conclusions about quality of the product or quality of the software process used.

In the recent literature, a large number of measures have appeared for capturing software attributes in a quantitative way. However, few measures have survived and are really used on the industry. A number of problems are responsible for the metrics failure, some of them are identified in [10] [11]. We select some of these problems to analyze the set of metrics presented on this survey. The main problems are:

a) Metrics automation
b) Metrics Validation

Some work in developing tools for metrics extraction is identified. It happens because a large number of metrics is developed, but they don't have a clear definition. Normally a code metric is defined in a large context and it is validated only for a few set of programmer languages.

There are a number of problems related to theoretical and empirical validity of many measures [10] [11] [18], the most relevant of which are summarized next.

a) Measurement goal, sometimes measurers aren't defined in an explicit and well-defined context
b) Experimental hypothesis, sometimes the measure doesn't have a explicit experimental hypothesis, e.g. what do you expect to learn from the analysis?
c) Environment or context, the measure sometimes can be applied in an inappropriate context
d) Theoretical Validation, a reasonable theoretical validation of the measure is often not possible because the metrics attributes aren't well defined.
e) Empirical validation, a large number of measures have never been subject to an empirical validation.

This set of problems about validation will be used on our analysis. In next section we will be presented a survey about software metrics.

2. Problem Formulation

The main objectives of this research are

1. To implement the existing algorithm that is having the classical approach with haphazard manner of operands and comments
2. We will calculate the execution time and complexity of the existing algorithm in the base paper
3. To propose a new algorithm that will be better than in the base paper in terms of code coverage and overall integrity of the software code.
4. Multi Layered investigation of code coverage will be applied in the proposed
5. Execution time shall be measured for the proposed approach
6. Comparison of existing and proposed in terms of execution time and overall cost of the implementation

At the end, the whole research work is concluded with some future research work.

3. Proposed Work

a) To design an effective and improved model for code coverage including comments density analysis, variables and operands used.
b) To design and implement the effective model for code investigation using Monte Carlo Simulation Techniques
c) The proposed work will deliver the optimized rules and solutions so that the proportional aspects of operands, constants and comments can be used in the source code.
d) Comparison shall be done on multiple parameters in Existing and Proposed Approach
e) In the classical or base work, there is no the limitation of the implementation of the comments density and investigation.
f) In our proposed work, the proportional investigation of the comments density shall be implemented.

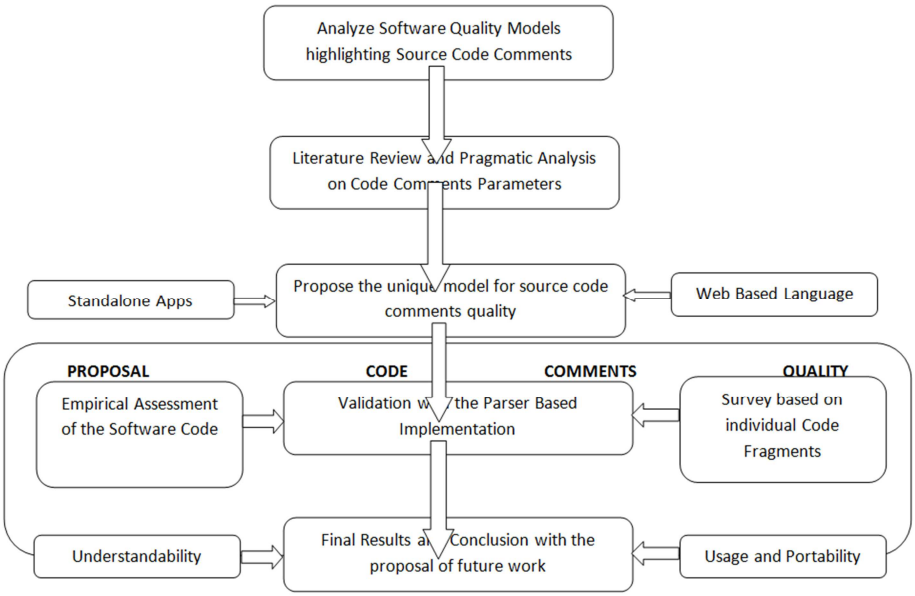

Figure 1. *Flow of the Work.*

4. Implementation Scenario

We have developed a web based simulator that read a source code. After reading the code, the software analyzes the lines of code based on various factors and parameters specified in the Code Metrics.

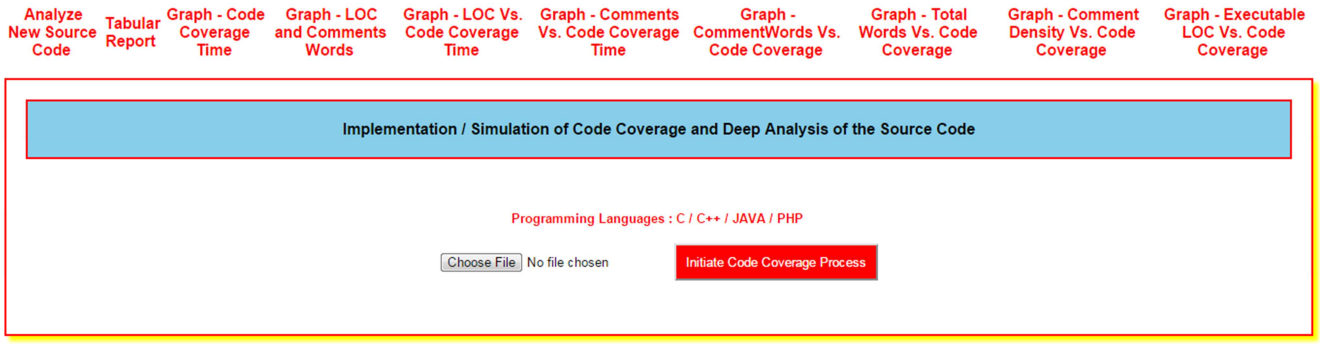

Figure 2. *Implementation Scenario of Code Coverage.*

Table 1. *Code Coverage Time of the Current Simulation Attempt.*

Filename	LOC	Comments	Words in Comments	Total Words	Comments Density	Executable LOC	Time
graph. php	57	4	14	161	8.695652173913	29	0.0052249431610107
comments. php	41	13	61	125	48.8	9	0.020942211151123
a. cpp	9	1	5	20	25	3	0.0047919750213623
a. cpp	9	1	5	20	25	3	0.004608154296875
a. cpp	9	1	5	20	25	3	0.0029840469360352
a. cpp	9	1	5	20	25	3	0.0032799243927002
a. cpp	9	1	5	20	25	3	0.0047969818115234
wp-login. php	961	248	1079	4413	24.450487196918	278	0.53639197349548
wp-comments-post. php	167	58	205	658	31.155015197568	47	0.093575954437256
index. php	98	5	71	284	25	27	0.017119884490967
data. php	18	0	0	29	0	6	0.0013720989227295
clusterpurchase. php	114	5	14	693	2.020202020202	53	0.044317007064819
jpgraph_error. php	157	52	182	459	39.651416122004	52	0.54635000228882
jpgraph_error. php	157	52	182	459	39.651416122004	52	0.60382008552551

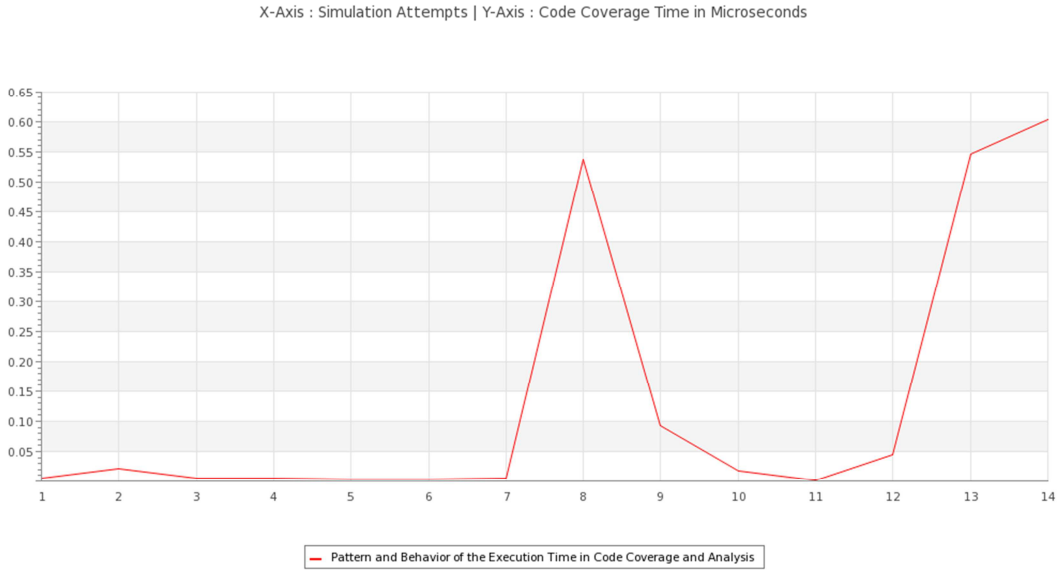

Figure 3. *Graph of Simulation Attempt with Code Coverage Time.*

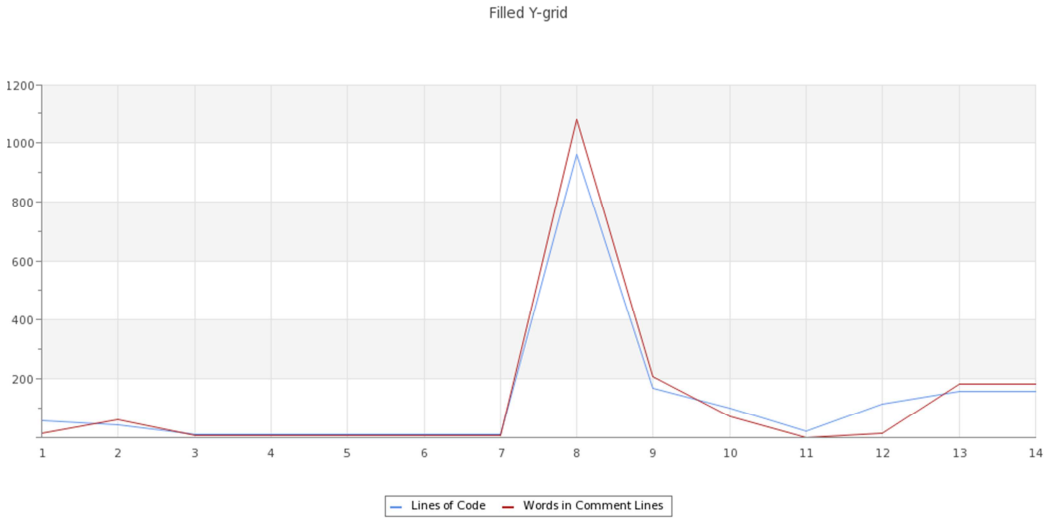

Figure 4. *Graph of Code Coverage Time for Lines of Code with Words in Comments.*

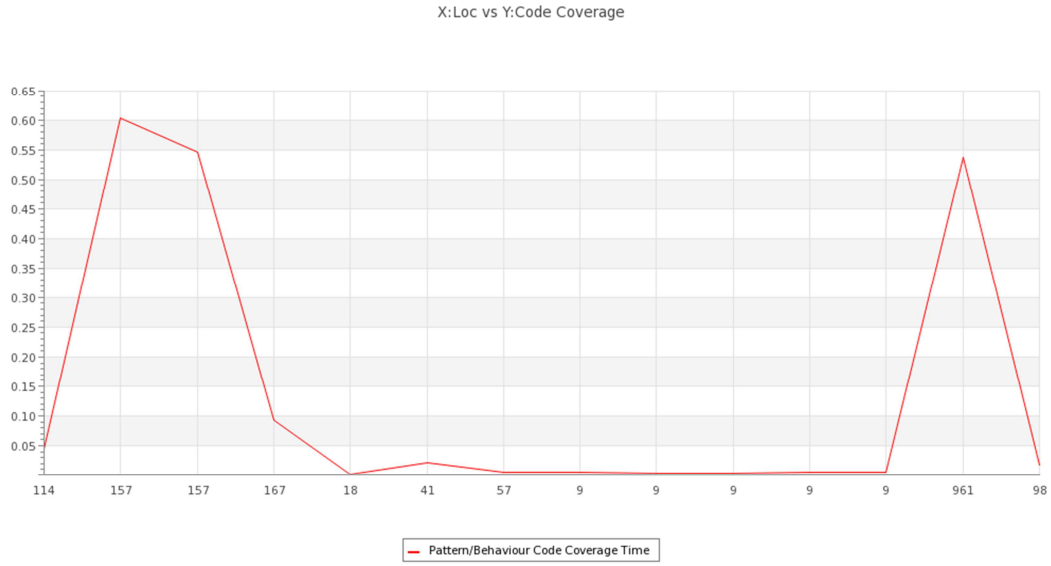

Figure 5. *Graph of Code Coverage Time with Pattern of LOC.*

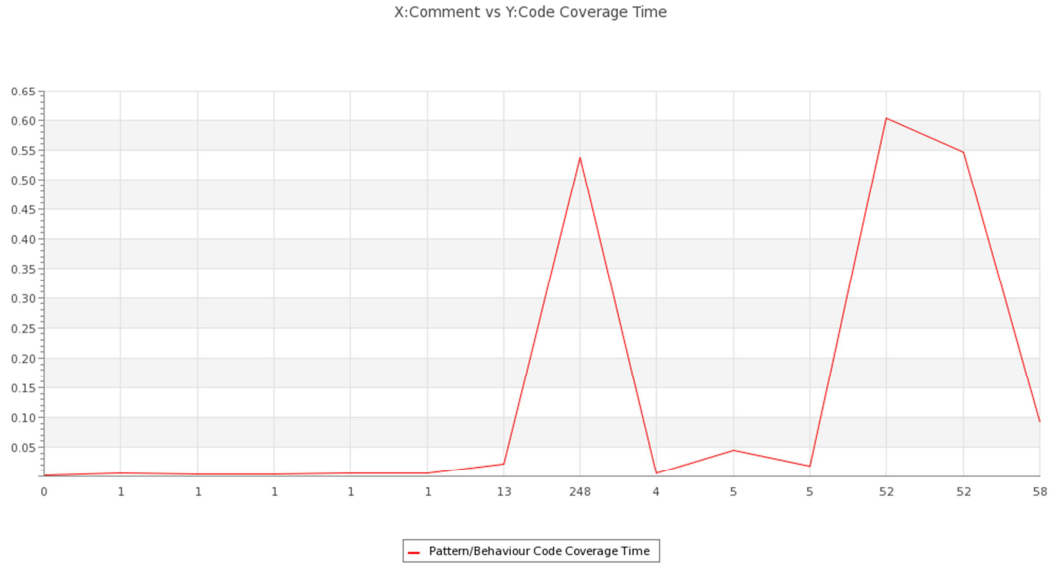

Figure 6. *Graph of Code Coverage Time with Comments Density.*

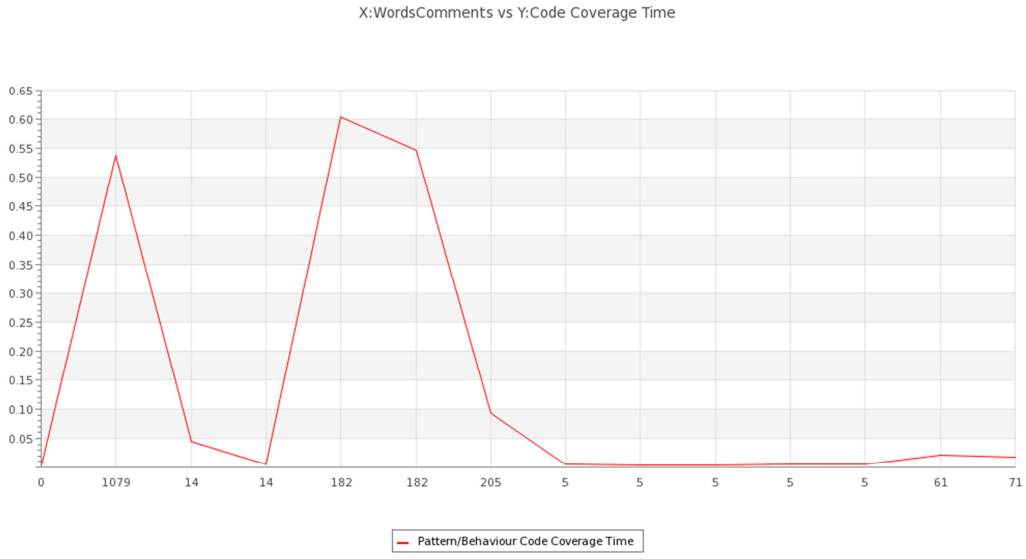

Figure 7. *Graph of Code Coverage Time with respect to Words in Comments.*

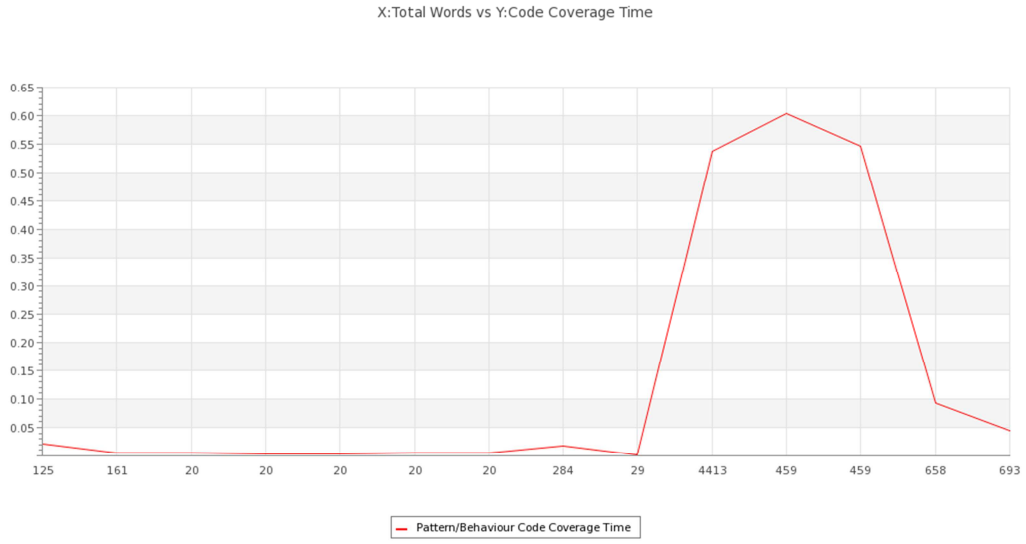

Figure 8. *Graph of Code Coverage Time with respect to the Total Words in the Source Code.*

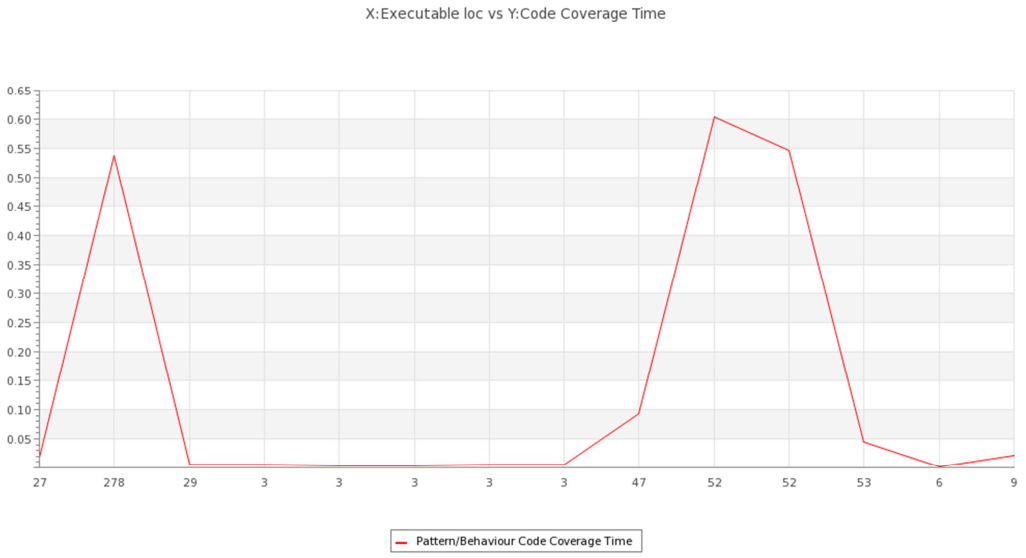

Figure 9. *Graph of Code Coverage Time with respect to the Executable Code.*

Table 2. *Overall Code Coverage Report of all the Simulation Attempts.*

Filename	LOC	Comments	Words in Comments	Total Words	Comments Density	Executable LOC	Time
comments. php	41	13	61	125	48.8	9	0.020942211151123
a. cpp	9	1	5	20	25	3	0.0047919750213623
a. cpp	9	1	5	20	25	3	0.004608154296875
a. cpp	9	1	5	20	25	3	0.0029840469360352
a. cpp	9	1	5	20	25	3	0.0032799243927002
a. cpp	9	1	5	20	25	3	0.0047969818115234
wp-login. php	961	248	1079	4413	24.450487196918	278	0.53639197349548
wp-comments-post. php	167	58	205	658	31.155015197568	47	0.093575954437256
index. php	98	5	71	284	25	27	0.017119884490967
data. php	18	0	0	29	0	6	0.0013720989227295
clusterpurchase. php	114	5	14	693	2.020202020202	53	0.044317007064819
jpgraph_error. php	157	52	182	459	39.651416122004	52	0.54635000228882
jpgraph_error. php	157	52	182	459	39.651416122004	52	0.60382008552551

5. Conclusion and Scope of Future Work

We have executed different types of source code of C++ and Java to test and measure the complexity parameters. The execution time of the proposed and classical approach is measured so that the empirical comparison can be done.

Code coverage analysis is used to measure the quality of software testing, usually using dynamic execution flow analysis. There are many different types of code coverage analysis, some very basic and others that are very rigorous and complicated to perform without advanced tool support. The proposed work can be integrated with genetic algorithm or ant colony optimization for further improvements and enhancements in the optimization task.

As the domain of software testing is much diversified, there is lots of scope of research for the scholars and practitioners. In Code Based Software Testing, the following research areas can be worked out by the research scholars -

a) Component Based Code Investigation
b) Security and Privacy Issues in Code Modules
c) Cross Platform Compatibility and Efficiency Issues
d) Functional Aspects and Scenarios
e) Analysis of Comments Density
f) Analysis of Operands and relative performance on overall code
g) Halstead Metrics Analysis

To improve the base work done in the existing algorithm having the classical approach with haphazard manner of operands and comments, we will calculate the execution time and complexity of the existing algorithm in the base paper. At the end, the whole research work will be concluded with some future research work. To design an effective and improved model for code coverage including comments density analysis, variables and operands used. To design and implement the effective model for code investigation using Monte Carlo Simulation Techniques, the proposed work will deliver the optimized rules and solutions so that the proportional aspects of operands, constants and comments can be used in the source code. Comparison shall be done on multiple parameters in Existing and Proposed Approach.

For future work, this work plan to extend my study in the following directions:

a) The metaheuristic based implementation can be performed that includes ant colony optimization, honey bee algorithm, simulated annealing and many other others. Such algorithmic approach should provide better results when we move towards metaheuristics.
b) This research work mainly discusses Halstead software complexity metrics for specific programming languages.
c) We can also plan to extend this algorithm for the processing of heterogeneous programming paradigms.

References

[1] D. Spinellis, “Code Quality: The Open Source Perspective”, Addison-Wesley, Boston - MA, 2003.

[2] B. N. Corwin, R. L. Braddock, "Operational performance metrics in a distributed system", Symposium on Applied Computing, Missouri - USA, 1992, pp. 867-872.

[3] R. Numbers, "Building Productivity Through Measurement", Software Testing and Quality Engineering Magazine, vol 1, 1999, pp. 42-47.

[4] IFPUG - International Function Point Users Group, online, last update: 03/2008, available: http://www.ifpug.org/.

[5] B. Boehm, “Cost Models for Future Software Life Cycle Processes: COCOMO 2.0”, U.S. Center for Software Engineering, Amsterdam, 1995, pp. 57-94.

[6] N. E. Fenton, M. Neil, “Software Metrics: Roadmap”, International Conference on Software Engineering, Limerick - Ireland, 2000, pp. 357–370.

[7] M. K. Daskalantonakis, “A Pratical View of Software Measurement and Implementation Experiences Within Motorola”, IEEE Transactions on Software Engineering, vol 18, 1992, pp. 998–1010.

[8] R. S. Pressman, "Software engineering a practitioner's approach", 4th. ed, McGraw-Hill, New York - USA, 1997, pp. 852.

[9] I. Sommerville, “Engenharia de Software”, Addison-Wesley, 6° Edição, São Paulo – SP, 2004.

[10] D. C. Ince, M. J. Sheppard, "System design metrics: a review and perspective", Second IEE/BCS Conference, Liverpool - UK, 1988, pp. 23-27.

[11] L. C. Briand, S. Morasca, V. R. Basili, “An Operational Process for Goal-Driven Definition of Measures”, Software Engineering - IEEE Transactions, vol 28, 2002, pp. 1106-1125.

[12] Refactorit tool, online, last update: 01/2008, available: http://www.aqris.com/display/ap/RefactorIt.

[13] O. Burn, CheckStyle, online, last update: 12/2007, available: http://eclipse-cs.sourceforge.net/index.shtml.

[14] M. G. Bocco, M. Piattini, C. Calero, "A Survey of Metrics for UML Class Diagrams", Journal of Object Technology 4, 2005, pp. 59-92.

[15] J Depend tool, online, last update: 03/2006, available: http://www.clarkware.com/software/JDepend.html.

[16] Metrics Eclipse Plugin, online, last update: 07/2005, available: http://sourceforge.net/projects/metrics.

[17] Coverlipse tool, online, last update: 07/2006, available: http://coverlipse.sourceforge.net/index.php.

[18] J Hawk Eclipse Plugin, online, last update: 03/2007, available: http://www.virtualmachinery.com/jhawkprod.htm
International Journal of Computing and Corporate Research, 3 (4).