Available Online at www.ijcsmc.com

International Journal of Computer Science and Mobile Computing

A Monthly Journal of Computer Science and Information Technology

ISSN 2320–088X

IJCSMC, Vol. 3, Issue. 8, August 2014, pg.146 – 161

RESEARCH ARTICLE

An Effective Implementation of Improved Halstead Metrics for Software Parameters Analysis

Prachi Chhabra[1]
M.Tech Scholar, CSE, GNI Mullana, Kurukshetra University
Lalit Bansal[2]
Assistant Professor, CSE, GNI Mullana, Kurukshetra University

ABSTRACT - The software metrics area has a lack of reports about metrics usage and real applicability of those metrics. In this sense, the main goal of this section is briefing review research in software metrics regarding source code in order to guide the research in software quality measurement. The 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. This paper focus on the specific improvements or enhancements in the halstead metrics in terms of object oriented paradigm.
Keywords – Software Metrics, Halstead Metrics, Software Quality

I. INTRODUCTION

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]. 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]. In agreement with Daskalantonakis in [9] it

© 2014, IJCSMC All Rights Reserved

is found the best motivation to measures, it is finding a numerical value for some software product attributes or software process attributes. Tshen, 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]. We select some of these problems to analyze the set of metrics presented on this survey. The main problems are:

- Metrics automation
- Metrics Validation

II. METRICS VALIDATION

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

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

This set of problems about validation will be used for analysis. In next section a survey about software metrics presented by different researchers is discussed.

III. HALSTEAD COMPLEXITY MEASURES

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.

© 2014, IJCSMC All Rights Reserved

A Program Length (N)

The program length (N) is the sum of the total number of operators and operands in the program:

$$N1 = N1 + N2 \quad \text{… .(1.1)}$$

B Vocabulary Size (N)

The vocabulary size (n) is the sum of the number of unique operators and operands:

$$n = n1 + n2 \quad \text{… (1.2)}$$

C Program Volume (V)

The program volume (V) is the information contents of the program, measured in mathematical bits. It is calculated as the program length times the 2-base logarithm of the vocabulary size (n) :

$$V = N \times log\,2\ \ (n) \quad \text{. …(1.3)}$$

D Difficulty Level (D)

The difficulty level or error proneness (D) of the program is proportional to the number of number of unique operators in the program D is also proportional to the ration between the total number of operands and the number of unique operands (i.e. if the same operands are used many times in the program, it is more prone to errors).

$$D = (n1 \div 2) \times (N2\ \div n2) \quad \text{… (1.4)}$$

E. Program Level (L)

The program level (L) is the inverse of the error proneness of the program. i.e. a low level program is more prone to errors than a high level program.

$$L\ =\ 1\,/\,D \quad \text{... (1.5)}$$

F Effort To Implement (E)

The effort to implement (E) or understand a program is proportional to the volume and to the difficulty level of the program.

$$E\ = V\ \times\ D \quad \text{… (. 1.6)}$$

© 2014, IJCSMC All Rights Reserved

G. Time To Implement (T)

The time to implement or understand a program (T) is proportional to the effort. Empirical experiments can be used for calibrating this quantity. Halstead has found that dividing the effort by 18 give an approximation for the $time\ in\ seconds$.

$$T = E / 18 \qquad \dots (1.7)$$

H. NUMBER OF DELIVERED BUGS (B)

The number of delivered bugs (B) correlates with the overall complexity of the software. Halstead gives the following formula for B:

$$B = \left(E\frac{2}{3}\right) \div 3000 \qquad \dots (1.8)$$

IV. LITERATURE REVIEW

Sommerville [9] classifies metrics in two categories:

(i) Control Metrics generally associated with software process;

(ii) Predict Metrics, normally associated with software product.

In this work the focus is Predict Metrics, because the predict metrics measures the static and dynamic characteristics of the software [9]. According to [17] the first key metric used to measure programming productivity and effort was Lines of Code (LOC or KLOC for thousands of lines of code) metric. It still is used routinely as the basis for measuring programmer productivity.

Zuse and Fenton [21] agree that in the mid-1970, there was a need for more discriminating measures rather than only LOC measure, especially with the increasing diversity of programming languages. After all, a LOC in an assembly language is not comparable in effort, functionality, or complexity to a LOC in a high level language. Also, there are easy identify the main problems in this measure, Environment or context and Measurement goal, the ruler not specify what kind of context the metric can be used.

Nowadays, the LOC metric is implemented in many used metrics tools [12] and it can be used to calculate other metrics. The 1970's started with an explosion of interest in measures of software complexity. Many works about software complexity can be found in literature [23]. The most referenced program complexity metric is the Cyclomatic Complexity, v(G), [23]. The Cyclomatic Complexity is derived from a flow graph and is mathematically computed using graph

© 2014, IJCSMC All Rights Reserved

theory (i.e. it is found by determining the number of decision statements in a program). The cyclomatic complexity can be applied in several areas including [27]

- Code development risk analysis, which measures code under development to assess inherent risk or risk buildup.
- Change risk analysis in maintenance, where code complexity tends to increase as it is maintained over time.
- Test Planning, mathematical analysis has shown that cyclomatic complexity gives the exact number of tests needed to test every decision point in a program for each outcome.

This measure is based upon the premise that software complexity is strongly related to various measurable properties of program code. Nowadays, this measure is strongly used for measure complexity in industry and academy, because it has a clear measurement goal, McCabe specify clearly what is complexity and how to quantify complexity using Cyclomatic Complexity metric. This metric measure complexity in a structural context, it is great because the measure is not dependent of technology or program language used. This metric have been implemented in many metrics tools [12] and it had been validated in many industrial works [24].

Another program complexity metric found on literature is Halstead metric [30], it was created in 1977 and it was determined by various calculations involving the number of operators and the number of operands in a program. Halstead metric [30] which is different of the McCabe metrics [23], because the McCabe metric determines code complexity based on the number of control paths created by the code and this one is based on mathematical relationships among the number of variables, the complexity of the code and the type of programming language statements. Nowadays, Halstead metric isn't used frequently, because in your measurement goals are clearly related to the program language used, it doesn't have a large validation by industrial works [24]. We find some tools implementing this metric [18].

In the age 1, before 1991, we identify few works in system design metrics area. Yin and Winchester, [31] created two metric groups called: primary metrics and secondary metrics. The primary metric are expressed through extracted values of the specification of design. These metrics are based on two design attributes: coupling and simplicity. These metrics have been used in some organizations [26] and all reports indicate their success in pinpointing error-prone areas in the design.The validation of this metric is poor, because this measure ignores the use of modules on the system design. Some researches obtained a high correlation between values of the metric and error counts, but only when the analyzed system has small number of modules. One aspect to note about this work is that it gave rise to the first reported example of a software tool used for design [32].

Another complexity metric was defined by McClure [33]. This work focuses on the complexity associated with the control structures and control variables used to direct procedure invocation in a program. In this metric a small invocation complexity is assigned to a component which, for example, is invoked unconditionally by only one other component. A higher complexity is assigned to a component which is invoked conditionally and where the variables in the condition are modified by remote ancestors or descendents of the component. We don't find reports about tools that implements this metric and we found some researches about this metric application [24].

After sometime, Woodfield [36] publish another complexity system metric. He observes that a given component must be understood in each context where it is called by another component or affects a value used in another component. In each new context the given component must be reviewed. Due to the learning from previous reviews, each review takes less effort than the previous ones. Accordingly, a decreasing function is used to weight the complexity of each review. The total of all of these weights is the measure assigned to the component. Woodfield applied this measure successfully in a study of multiprocedure student programs. We don't find reports about tools that implements this metric. We found some reports about this metric application .

In 1981, Henry and Kafura [35] created another system complexity metric. Henry and Kafura´s metric determine the complexity of a procedure, which depends on two factors: the complexity of the procedure code and the complexity of the procedure's connections to its environment. Henry and Kafura´s approach is more detailed than Yin and Winchester, [31] metric, because it observes all information flow rather than just flow across level boundaries. It has another major advantage in that this information flow method can be completely automated.

However, some definitions, like flows definition and modules definition, are confusing. Consequently different researches have interpreted the metric in different ways thus disturb the comparison of results [24]. According to [10] another problem in Henry and Kafura´s approach is the validation, because the algebraic expression on the metric definition is seems arbitrary and the application of parametric tests to data which is skewed is questionable. We don't find metrics tools implementing this metric. The first suites of OO design metrics was proposed by Chidamber and Kemerer [36], which proposed six class-based design metrics for OO system (CK Metrics). However, the CK metrics can be used to analyse coupling, cohesion and complexity very well, but the CK metrics suffer from unclear definition and a failure to capture OO specifics attributes. The attributes of data-hiding, polymorphism and abstraction not measured all and the attributes of inheritance and encapsulation are only partially measured.

The CK metrics are the most referenced [37] and most commercial metrics collection tools available at the time of writing also collect these metrics [12]. The CK metrics validation catch our

© 2014, IJCSMC All Rights Reserved

attention because is a complete work if we compare to other metrics. We could find researches in industry and academy [39][40][41], using many programmer languages.

Sometimes ago, Lorenz and Kidd created a set of OO design metrics [38]. They divided the classes-based metric in 4 categories [8]: size, inheritance, internals and externals. Size-oriented metrics for the OO classes focus on counts of attributes and operations. Inheritance-based metrics focus on the manner in which operations are reused in hierarchy class. Metric for internal class look at cohesion and code-oriented issues, and the external metrics examine coupling and reuses. Probably CK metrics [36] are more known and complete then Lorenz and Kidd metrics [42] because include more OO attributes in its analysis. To our knowledge no worked related to the theoretical validation of this metric has been published. According to [13], a tool called OO Metric was developed to collect these metrics, applied to code written in Smalltalk and C++.

It was defined to measure the use of OO design mechanisms such as inheritance metrics, information hiding, polymorphism and the consequent relation with software quality and development productivity [43]. The validation for this set of metrics is questionable for Polymorphism Factor metric (PF), because it is not valid, in a system without inheritance the value of PF is not defined, being discontinuous. The MOODKIT is a tool for metrics extraction from source code, which supports the collection for C++, Smalltalk and Eiffel code [14].

The metrics are the measurement of the coupling between classes [44]. Their empirical validation conclude that if one intends to built quality models of OO design, coupling will very likely be an important structural dimension to consider. We could find a tool for this for metrics extraction. The research in software metrics continue intense in 90´s decade. Some other OO metrics were created like [45][46], many works analyzing the metrics and about validating metrics were published. The software metrics scenario, after 2000, present little reports about new metrics. The proposed metric in [46] is not based in the classical metrics framework. The Chatzigeorgiou´s work is innovative because apply a web algorithmic from verify the relation between design classes and not use the traditional and existents metrics. Chatzigeorgiou validate your metric comparing it with classics OO software metrics. In the first analysis was verified the ability to account for the significance of the related classes and the ability to consider both incoming and outgoing flows of messages. The Lorenz and Kidd [38] these metrics not fulfilled to the ability to account for the significance of the related classes, but, although it fulfils ability to consider both incoming and outgoing flows of messages.

© 2014, IJCSMC All Rights Reserved

V. Research Methodology

The existing Halstead Metrics can be improved by devising new parameters for measurement of the metrics. There are numerous imitations of existing Halstead Metrics. The difficulties to differentiate operands and operators were a major problem in Halsted's model. This work explains that code complexity increases as volume increases without specifying which value of program level makes the program complex. And another issue in this technique is that Halstead method not capable of measuring the structure of a code, inheritance, interactions between modules. As it is difficult to estimate, the information will be needed to calculate the mentioned metrics (lines of code- number of operands/operators) early in the analysis and design phases these metrics are calculated after the implementation level which makes this kind of metrics not suitable for our objectives. Moreover, it is not possible to consider these metrics as a measure of complexity at all. A function with negligible length and contains many nesting loops can be more complex than a large function without conditional statements.

A Proposed and Implemented Objectives

1. To study various software metrics for analysis of features and parameters of programs.
2. To investigate various software metrics for software complexity.
3. To analyze Halstead's metrics specifically for attributes measurement of software.
4. To propose a new model and measurement class that should be added in the Halstead's metrics.
5. To implement the proposed class or parameter in a suitable tool for results analysis.
6. To implement the classical approach as well as proposed approach using simulation tool.
7. To perform a comparative analysis between classical and proposed approach in terms of graphs and tables.

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 Halstead Metrics.

© 2014, IJCSMC All Rights Reserved

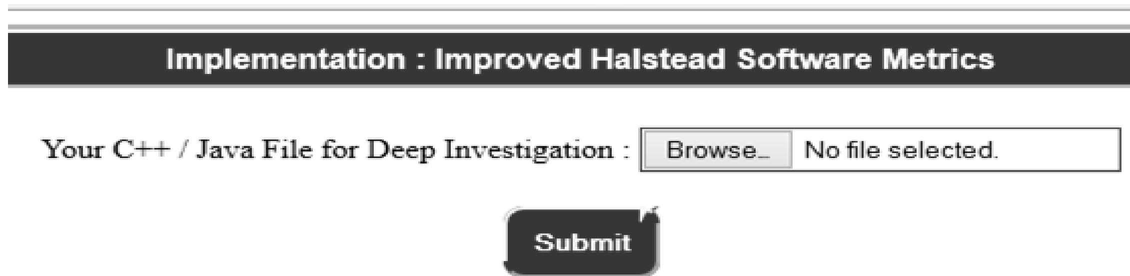

Figure 1 – Implementation Screenshot of the Simulator

OUTPUT FETCHED

Stats Object

(

total distinct operators => 6

total operators => 10

total distinct unique operands variables constants => 4

total number of operands variables constants => 10

total number of struct used => 0

total number of classes => 0

total number of constructors destructors => 0

total lines of code => 21

total comment lines => 5

total friend functions => 0

total virtual functions => 0

total file pointers => 0

)

Proposed Approach : Halstead Software Metrics

Program Vocabulary (n) => 23

Program Length (N) => 45

Program Difficulty (D) => 17.5

Calculated Program Length (N) => 234.4125953201

Volume (V) => 342.23523

Effort (E)=> 411.235253

Execution Time Proposed Approach (Microseconds) : 0.002324343

Classical Approach : Halstead Software Metrics

Program Vocabulary (n) => 22

Program Length (N) => 12

Program Difficulty (D) => 21.5

Calculated Program Length (N) => 18.729055953201

Volume (V) => 21.9162238128

Effort (E)=> 324.74867143841

Execution Time Classical Approach (Microseconds) : 0.023523532

VI. RESULT ANALYSIS

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.

Table 1.IMPLEMENTATION / SIMULATION SCENARIO – 1

Attempt ID	Classical Approach	Proposed Approach
1	1.00519800186	0.200445890427
2	1.00735902786	0.37758398056
3	1.00400495529	0.0761110782623
4	1.0077021122	0.00262784957886
5	1.47337388992	0.0465440750122
6	1.01530909538	0.0499310493469

© 2014, IJCSMC All Rights Reserved

7	1.10068583488	0.0308630466461
8	1.01806807518	0.0550210475922
9	1.08409404755	0.0505809783936

A. INTRPRETATION OF THE TABLE

In this simulation scenario, we have taken the set of source code for execution using classical and proposed improved approach. The implementation has been performed to test and analyse the results from the Live Server Based Deployment. The results very clearly show that the proposed algorithmic system or approach is providing effective as well as efficient results as compared to the classical approach.The Live web based implementation that is based on Halstead Metrics analysis has been tested and simulated with PHP Scripts and Graphical Implementation. It is found without any specific qualm that the proposed system approach of optimization is efficient to justify and prove the research work. In the proposed as well as implemented research work, it is apparent that the code metrics optimization time of the proposed algorithmic approach is rapid and acceptable. In the upcoming graphical representation and analysis, we will explain the results so that the proposed approach can be proved better than the classical approach. We have taken the Live Web Server response time and generated the different types of graphs and all graphs are proving that we are getting better results in the proposed approach.

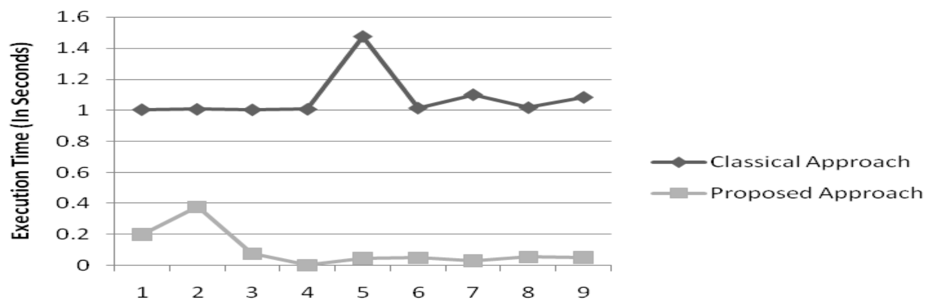

Figure 2 - Execution Time of Classical and Proposed Approach

© 2014, IJCSMC All Rights Reserved

B. INTERPRETATION OF THE GRAPH

The above drawn graph has been generated from the data fetched from the live simulator and our implementation performed on the data set. It can be seen from the graph that the query execution time in the proposed approach is far lesser than that classical approach.

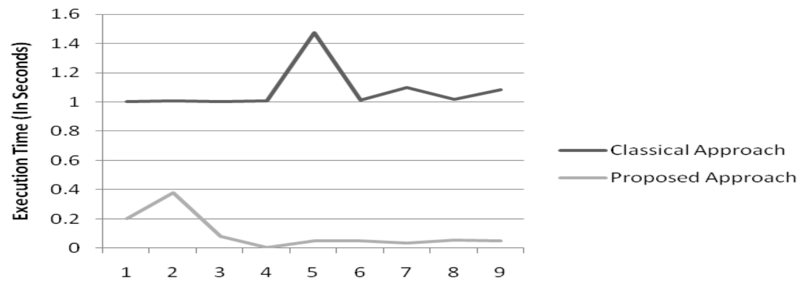

Figure 3 - Execution Time of Classical and Proposed Approach

C INTERPRETATION OF THE GRAPH

The graph above mentioned shows that that demarcation line of the classical versus proposed approach is having huge difference. The graphical representation and analysis of the figure demonstrates the fact as well as conclusion that the proposed improved implemented is better as compared to the classical Ha

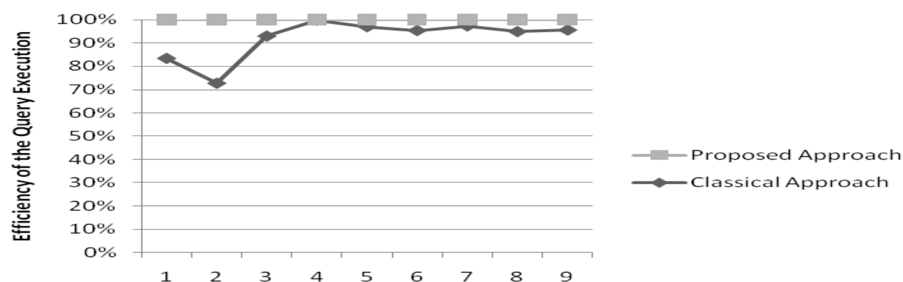

Figure 4 - Execution Time of Classical and Proposed Approach with Parameter based efficiency measurement

D. INTERPRETATION OF THE GRAPH

In the above drawn graphical representation, it has been found that the proposed approach as the parameter of performance and made to the 100% efficiency level. Once this value of 100% set to the proposed level, we have analyzed the classical approach. It has been found that if proposed approach is to the point of 100%, then the classical approach is very far from that performance point. It is clear from the graphical representation showing the performance level of both approaches, the proposed approach is still on the upper part of the performance line as compared to the classical approach.

© 2014, IJCSMC All Rights Reserved

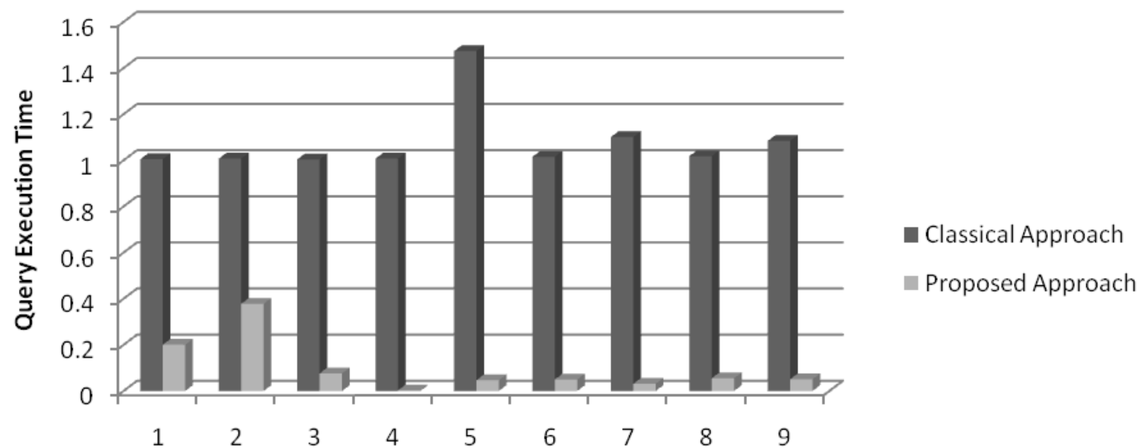

Figure 5 Execution Time Analysis from the Classical and Proposed Approach (Bar Chart Based Comparison)

VII. CONCLUSION AND FUTURE WORK

For future work, this work plan to extend the study in the following directions: 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 one move towards metaheuristics. This research work mainly discusses Halstead software complexity metrics for specific programming languages. We can also plan to extend this algorithm for the processing of heterogeneous programming paradigms.

REFERENCES

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

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

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

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

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

[6] 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.

© 2014, IJCSMC All Rights Reserved

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

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

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

[10] 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.

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

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

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

[14] JHawk Eclipse Plugin, online, last update: 03/2007, available: http://www.virtualmachinery.com/jhawkprod.htm

[15] S. Morasca, L. C. Briand, V. R. Basili, E. J. Weyuker, M. V. Zelkowitz, B. Kitchenham, S. Lawrence Pfleeger, N. Fenton, "Towards a framework for software measurementvalidation", Software Engineering, IEEE Transactions, vol 23, 1995, pp. 187-189.

[16] H. F. Li, W. K. Cheung, “An Empirical Study of Software Metrics”, IEEE Transactions on Software Engineering, vol 13, 1987, pp. 697-708.

[17] H. Zuse, “History of Software Measurement”, online, last update: 09/1995, lavailable: http://irb.cs.tu-berlin.de/~zuse/metrics/History_00.html

[18] N. E. Fenton, M. Neil, “Software Metrics: Roadmap”, International Conference on Software Engineering, Limerick - Ireland, 2005, pp. 357–370.

[19] T. J. McCabe, “A Complexity Measure”. IEEE Transactions of Software Engineering, vol SE-2, 1976, pp. 308-320.

[20] D. Kafura, G. Reddy, “The Use of Software Complexity Metrics in Software Maintenance”, IEEE Transactions on Software Engineering archive, vol 13 , New Jersey USA, 1987, pp. 335-343.

[21] B. Ramamurty, A. Melton, “A Syntheses of Software Science Measure and The Cyclomatic IEEE Transactions on Software Engineering, vol 14, New Jersey - . USA, 1988, pp. 1116-1121

© 2014, IJCSMC All Rights Reserved

[22] J. K. Navlakha, “A Survey of System Complexity Metrics”, The Computer Journal, vol 30, Oxford - UK, 1987, pp. 233-238.

[23] E. VanDoren, K. Sciences, C. Springs, “Cyclomatic Complexity”, online, last update: 01/2007, available: http://www.sei.cmu.edu/str/descriptions/cyclomatic_body.html

[24] R. K. Lind, K. Vairavan, “An Experimental Investigation of Software Metrics and Their to Software Development Effort”, IEEE Transactions on Software Engineering, New Jersey - USA, 1989, pp. 649-653.

[25] M. H. Halstead, Elements of Software Science, Operating, and Programming Systems, vol 7, New York - USA, 1977, page(s): 128.

[26] B. H. Yin, J. W. Winchester, "The establishment and use of measures to evaluate the quality of software designs", Software quality assurance workshop on Functional and performance, New York - USA, 1978, pp. 45-52.

[27] R. R. Willis, "DAS - an automated system to support design analysis", 3rd international Software engineering, Georgia - USA, 1978, pp. 109-115.

[28] C. L. McClure, “A Model for Program Complexity Analysis”, 3rd International Software Engineering, New Jersey - USA, 1978, pp. 149-157.

[29] S. Henry, D. Kafura, "Software Structure Metrics Based on Information Flow", Software Engineering, IEEE Transactions, 1981, pp. 510-518.

[28] S. R. Chidamber, C. F. Kemerer, “A Metrics Suite for Object Oriented Design”, IEEE Transactions on Software Engineering, vol 20, Piscataway - USA, 1994, pp. 476-493.

[29] M. Alshayeb, M. Li, "An Empirical Validation of Object-Oriented Metrics in Two Different Iterative Software Processes", IEEE Transactions on Software Engineering archive, vol 29, 2003, pp. 1043–1049.

[30] R. Subramanya, M. S. Krishnan, “Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implication for Software Defects”, IEEE Transactions on Software Engineering, vol 29, 2003, pp. 297-310.

[31] L. C. Briand, S. Morasca, V. R. Basili, "Property-based software engineering measurement", Software Engineering, IEEE Transactions, vol 22, 1996, pp. 68 - 86.

[32] S. R. Chidamber, D. P. Darcy, C. F. Kemerer, "Managerial use of metrics for object-oriented software: anexploratory analysis", Software Engineering, IEEE Transactions, vol 24, 1998, pp. :629–639.

[33] N. Woodfield, "Enhanced effort estimation by extending basic programming models to include modularity factors", West-Lafayette, USA, 1980.

© 2014, IJCSMC All Rights Reserved

[34] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen, "An empirical study on object-oriented metrics", Software Metrics Symposium, 1999, pp. 242-249.

[35] M. Lorenz, J. Kidd, “Object-Oriented Software Metrics: A Practical Guide”, Englewood Cliffs, New Jersey - USA, 1994.

[36] A. F. Brito, R. Carapuça, "Object-Oriented Software Engineering: Measuring and controlling the development process", 4th Interntional Conference on Software Quality, USA, 1994.

[37] L. Briand, W. Devanbu, W. Melo, "An investigation into coupling measures for C++", 19th International Conference on Software Engineering, Boston - USA, 1997, pp. 412-421.

[38] R. Harrison, S Counsell, R. Nithi, "Coupling Metrics for Object-Oriented Design", 5th International Software Metrics Symposium Metrics, 1998, pp. 150-156.

[39] M. Marchesi, "OOA metrics for the Unified Modeling Language", Second Euromicro Conference, 1998, pp. 67-73.

[40] N. F. Schneidewind, "Measuring and evaluating maintenance process using reliability, risk, and test metrics", Software Engineering, IEEE Transactions, vol 25, 1999, pp. 769-781.

© 2014, IJCSMC All Rights Reserved