Journal of Nature Inspired Computing (JNIC)
Vol. 1, No. 2, 2013

www.worldsciencepublisher.org

Implementation of Cyclomatic Complexity Matrix

[1]Ambuj Kumar Agarwal, [2]Dr. Vinodini katiyar

[1]Teerthanker Mahaveer University, Moradabad, India
[*2]SRM University, Lucknow, India

Email: ambuj4u@gmail.com

Abstract – Cyclomatic complexity (or conditional complexity) is a software metric (measurement). It directly measures the number of linearly independent paths through a program's source code. The concept, although not the method, is somewhat similar to that of general text complexity measured by the Flesch-Kincaid Readability Test. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

Keywords – Cyclomatic Complexity, Readability Test

1. Introduction

Cyclomatic complexity (or conditional complexity) is a software metric (measurement). It directly measures the number of linearly independent paths through a program's source code. The concept, although not the method, is somewhat similar to that of general text complexity measured by the Flesch-Kincaid Readability Test. Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

One testing strategy, called Basis Path Testing by McCabe who first proposed it, is to test each linearly independent path through the program; in this case, the number of test cases will equal the cyclomatic complexity of the program.

1.1 Analytic Critiques

McCabe [1976] proposed that a measure of the complexity is the number of possible paths through which the software could execute. Since the number of paths in a program with a backward branch is infinite, he proposed that a reasonable measure would be the number of independent paths.
After defining the program graph for a given program, the complexity calculation would be:

$V(G) = e - n + 2.$
where $V(G)$ = the cyclomatic complexity,
e = the number of edges,
n = the number of nodes.

1.2 Empirical Tests

In Shepperd's comprehensive review of previous research on the metric, he notes that, among the numerous empirical validations or uses of the metric, that the most consistent single result is the high degree of correlation between McCabe's metric and a count of source lines of code (SLOC). As evidence of this, four studies cited by Shepperd had Pearson correlation coefficients of .9 or greater for these two metrics. This "empirical criticism" suggest that the additional effort required for computing and understanding the McCabe cyclomatic complexity metric may not be practically worthwhile. However, as noted by Shepperd, concerns about external vahdity of the data and analyses in some of the previous studies can be raised to mitigate some of the results, particularly those using data on small programs from student subjects. Therefore, these results bear validation on data from actual systems. In particular, applied research in this area does not seek to determine whether cyclomatic complexity captures all aspects of complexity in one figure of merit, but rather looks to answer the overriding question raised by Shepperd, as to whether cyclomatic complexity can serve as a "useful engineering approximation".

1.3 Terminology and Notation

In many of the related software metrics the authors have given mathematical notation to concretely describe their metrics fully. However, many of these notations are merely created just for their specific metrics. Fortunately Briand proposed a standard notation that is used to describe software metrics so that all could readily understand the terminology.

2. A Unified Framework for Coupling

Briand introduced a unified framework for defining coupling measurement in object-oriented systems. They review three other previous attempts at defining such a framework and attempt to improve and unify the terminology. The previous frameworks have been proposed by Eder, Ritz and Montazeri and an earlier attempt by Briand. The framework utilizes mathematical notation to specifically define the different types of relationships. There are many definitions which are stated within the framework, for brevity, the definitions necessary for understanding the proposed metrics will be introduced.

2.1 Methods

A class has a set of methods. A method can be either virtual or non-virtual and either inherited, overridden, or newly defined, all of which have implications for measuring coupling.

Methods of a Class: For each class $c \in C$ let M(c) be the set of methods of class c.

The Set of all Methods: M(C) is the set of all methods in the system and is represented as

$$M(C) = \bigcup_{c \in C} M(c)$$

The Set of Methods Implemented in a Class: $M1(c) \subseteq M(c)$ be the set of methods implemented in c, i.e., methods that c inherits but overrides or non virtual non inherited methods of c.

Polymorphic Identification: P(m) is the function to identify which class the method m is dynamically bound to. $P(m) = C \in c$ where $m \in M(c)$

2.2 Method Invocations

To measure coupling of a class c, it is necessary to define the set of methods that $m \in M(c)$ invokes and the frequency of these invocations. Method invocations can be either static or dynamic. For static invocations, the invoked method is determined by the type of the variable that references the object. For dynamic invocations, the invoked method is determined by a late-binding at run-time to the polymorphic type. One definition which is needed here but not defined in the unified framework is the notion of a transitive relation upon method invocations. A method invocation may possibly invoke another method and so on. A proposed addition to the framework will be defined to account for this behavior.

The Set of Statically Invoked Methods of m: Let $c \in C$, $m \in M1(c)$, and $m' \in M(C)$. Then $m' \in S1M(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' is invoked for an object of static type class d.

The Set of Polymorphically Invoked Methods of m: P1M(m) is the set of all polymorphically invoked methods on m. Let $c \in C$, $m \in M1(c)$, and $m' \in M(C)$. Then $m' \in P1M(m) \Leftrightarrow \exists d \in C$ such that $m' \in M(d)$ and the body of m has a method invocation where m' may, because of polymorphism and dynamic binding, be invoked for an object of dynamic type d.

The Transitive Closure on a Set of Invoked Methods m: *T(*m) is the transitive closure on a set of invoked methods. Let m be a method, whether it be statically or polymorphically invoked. Let m be defined to be *mo*, where mo can invoke ml, ml can invoke *m2*, and so on.

2.2 Attributes

Classes have attributes which are either inherited or newly defined.

The Set of all Attributes: A(C) is the set of all attributes in the system and is represented as

$A(C) = \bigcup A(c)$ *where* $c \in C$

2.3 Attribute References

Methods may reference attributes. These attributes may not be part of the encompassing class, therefore coupling it to the referenced encompassing class.

The Set of Attributes referenced by the method m: For each $m \in M(C)$ let *AR(*m) be the set of attributes reference by method m.

2.4 Predicates

To ensure proper usage between terms, a *uses* predicate must be defined.

Uses: Let $c \in C$, $d \in C$. $uses(c, d) \Leftrightarrow (\exists m \in M_I(c): \nexists m' \in M_I(d): m' \in PIM(m)) \vee (\exists m \in M_I(c): \exists a \in A_I(d) : a \in AR(m))$

A class c uses a class *d* if a method implemented in class c references a method or an attribute implemented in class *d.*

3. Proposed Matrix

Ten metrics are proposed that measure different dimensions of a system when compared to CC and CBO. The metrics are grouped in a suite which we call Gray/ Janzen Coupling Complexity Metrics Suite. Some of the variations on CC look more in depth on a procedure's possible execution path rather then solely focusing on a procedure's static nature. The variations on CBO are upon CBO's coupling weight to other classes. This weight can be different depending on exactly *how* a class is coupled to another. The different method complexities and possible method execution paths will be explored.

3.1 Transitive Cyclomatic Complexity

CC computes a complexity value over a procedure. By McCabe's definition the number of connected components (method calls) or *P* is included within his complexity metric. It is defined as *M = E -N +2P* where the number of method calls is merely multiplied by two. Transitive Cyclomatic Complexity attempts to further this value. Instead of only using *2P*, TCC will inject the summation of all Cyclomatic.

Complexities computed on all methods that can possibly be executed on the static types invoked.

Class C_1, has methods $m_1, \ldots, m_n$ that are defined in the class:

$$McCabe(m_i) = E - N + 2P$$

$$CC(m_i) = Decisions + 1$$

$$TCC(C) = \sum_{i=1}^{n} CC(T(SIM(m_i)))$$

$$TWMC(C) = \sum_{i=1}^{n} McCabe(T(SIM(m_i)))$$

E is the number of edges and *N* is the number of nodes in the control flow graph of a method m. The plus one is used for when the CC of a method that has no decision points still maintains some complexity or more specifically a value of one.

4. Conclusions

Metrics are never fully validated until they have been tried and tested for many projects. These metrics have only been used on two industry projects totalling four different analyses across four versions. This is hardly a conclusive result for absolute proof of their accuracy. Future experiments can further investigate the accuracy of these metrics with other projects.

An experimental group could volunteer to use these metrics in developing software and supply feedback of their use. Analyses could be performed on the amount of work that is generated from the results of the metrics as well as another measurement judging the impact of this work on the overall system. Some examples could include showing the CC values of overall methods within the system and comparing them at a later date to see if the CC has decreased in particular classes. If this is the case it could be classified that this decrease was in response to a value that surpassed a particular threshold for a metric.

Threshold values for metrics are a hard goal. As discussed previously it is thought to be the case that the entire range of the metric's values must be taken into account when creating a threshold value. An algorithm for figuring out the threshold value of a metric depending on its current range of values would be an interesting piece of research.

Acknowledgements

I would like to express my sincere gratitude to my advisor Prof. Dr. Vinodini Katiyar for the continuous support of my PhD study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research. I could not have imagined having a better advisor and mentor for my research.

References

[1] A. B. Binkley and S. R. Schach. Validation of the coupling dependency metric as a predictor of run-time failures and maintenance measures. In Proceedings of the 1998 (20th) International Conference on Software Engineering, pages 452-455, Apr 1998.

[2] A. Beszedes, T. Gergely, S. Farago, T. Gyimothy, and F. Fischer. The dynamic function coupling metric and its use in software evolution. In CSMR '07. 11th European Conference on Software Maintenance and Reengineering, pages 103-112, Mar 2007.

[3] A. Dunsmore, M. Roper, and M. Wood. Practical code inspection for object oriented systems: an experimental comparison. IEEE Software, 20(4):21-29, July1Aug 2003.

[4] Albert L. Baker, Stuart H. Zweben (1980), "A Comparison of Measures of Control Flow Complexity", IEEE Transactions On Software Engineering, Vol. SE-6.

[5] Albrecht, A. J. and J. E. Gaffney. Jr. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Trans. Software Eng. SE-9, 6, Nov. 1983, pp. 639-648.

[6] Arifoglu, A.: A Methodology for Software Cost Estimation, ACM Sigsoft, vol.18 no.2, 1993.

[7] Arthur, L. J. "Measuring Programmer Productivity and Software Quality", New York, John Wiley, 1985.

[8] Austin, Robert D., Lister, Timothy R., Demarco, T. "Measuring & Managing Performance in Organizations", New York, Dorset House, June, 1996.

[9] B. Henderson-Sellers. Object-Oriented Metrics: Measures of Complexity. Object-Oriented Series. Prentice Hall, Upper Saddle River, New Jersey, 07458, 1996.

[10] Banker, R.D., Datar, S.M., Zweig, D.: Software Complexity and Maintainability CiteSeer Scientific Literature Digital Library and Search Engine.

[11] Barbacci, M.R., Klein, M.H., Longstaff, T.A., Weinstock, C.B.: Quality Attributes of a Software Architecture (last accessed 02.04.2010)

[12] Basci, D., Misra, S.: Measuring and Evaluating a Design Complexity Metric for XML Schema Documents' Code. Journal of Information Science and Engineering. Sep. 2009, pp.1415-1425.

[13] Basci, D., Misra, S: 'Data Complexity Metrics for Web-Services' Advances in Electrical and Computer Engineering, Volume 9, Number 2, 2009, pp.9-15.

[14] Basili, V.R.: Qualitative Software Complexity Models: A Summary. In Tutorial on Models and Methods for Software Management and Engineering. IEEE Computer Society Press, Los Alamitos, California, 1980.

[15] Boehm, B. W. "Software Engineering Economics", Englewood Cliffs, New Jersey, Prentice-Hall, 1981.

[16] Boehm, Barry W.; "Improving Software Productivity"; Computer, September 1987; pp. 43-57.

[17] Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering – Using UML, Patterns, and Java, 2nd International Edition, Prentice Hall, 2004.

[18] C. Rajaraman and M. R. Lyu. Reliability and maintainability related software coupling metrics in c++ programs. In Proceedings of the Third International Symposium on Software Reliability Engineering, pages 303-311, Oct 1992.

[19] Card, D. N. and W. W. Agresti. “Measuring Soft ware Design Complexity.” J. Syst. and Software 8, 3 (June 1988), 185-197.

[20] Cerino, D. A. “Software Quality Measurement Tools And Techniques.” Proc. COMPSAC 86.Washington, D. C.: IEEE Computer Society, Oct. 1986, 160-167.

[21] Chidamber S.R., Kemerer, C.F.: A Metric Suite for object oriented esign. IEEE Transactions Software Engineering, SE-6(1994)476-493. 86

[22] Conte, S. D., H. E. Dunsmore, and V. Y. Shen. Software Engineering Metrics and Models.Menlo Park, Calif.: Benjamin/Cummings, 1986.

[23] Conte, S. D.; Dunsmore, H. E.; and Shen, V. Y.; Software Engineering Metrics and Models; Benjamin/Cummings Publishing Qarapany, Inc., 1986.

[24] Cote, V., P. Bourque, S. Oligny, and N. Rivard. “Software Metrics: An Overview of Recent Results.” J. Syst. and Software 8, 2 (March 1988), 121-131.

[25] Curtis, B., S. B. Sheppard, P. Milliman, M. A. Borst, and T. Love. “Measuring the Psychological Complexity of Software Maintenance tasks with the Halstead and McCabe Metrics”, IEEE Trans. Software Eng. SE-5, 2 (March 1979), pp. 96-104.

[26] Tarun Kumar Sharma, Millie Pant, V.P. Singh, Improved Local Search in Artificial Bee Colony using Golden Section Search, Journal of Engineering, 1:1(2012) 14-19.

[27] Tarun Kumar Sharma, Millie Pant, V.P. Singh, Adaptive Bee Colony in an Artificial Bee Colony for Solving Engineering Design Problems, Advances in Computational Mathematics and its Applications, 1:4(2012) 213-221.

[28] Avanaksh Singh Sambyal and Prikhshayat SinghRouting Misbehavior in MANets and How it Impact QoS!, Advances in Computer Science and its Applications, 1:1(2012) 84-88.

[29] S.Balaji, Lakshmi.A, V.Revanth, M.Saragini and V.Venkateswara Reddy, Authentication techniques for engendering Session passwords with colors and text, Advances in Information Technology and Management, 1:2(2012) 71-78.

[30] C.Narasimha and B.Jalaja Kumari, Secured Multicasting over MANET’s through EGMP, Advances in Information Technology and Management, 1:2(2012) 90-96.

[31] M. Sreedevi,.C.Narasimha and R.Seshadri, Efficient Data Delivery Over MANET’s through Secured EGMP, Advances in Asian Social Science, 2:3(2012) 512-516.