CYCLOMATIC COMPLEXITY METRICS REVISITED: AN EMPIRICAL STUDY OF SOFTWARE DEVELOPMENT AND MAINTENANCE; OCTOBER 1990. CISR WP NO. 218, SLOAN WP NO. 3222-90

Published @ 2017 Trieste Publishing Pty Ltd

ISBN 9780649739998

Cyclomatic Complexity Metrics Revisited: An Empirical Study of Software Development and Maintenance; October 1990. CISR WP No. 218, Sloan WP No. 3222-90 by Geoffrey K. Gill & Chris F. Kemerer

Edited by Trieste Publishing Pty Ltd.
Cover @ 2017

GEOFFREY K. GILL & CHRIS F. KEMERER

CYCLOMATIC COMPLEXITY METRICS REVISITED: AN EMPIRICAL STUDY OF SOFTWARE DEVELOPMENT AND MAINTENANCE; OCTOBER 1990. CISR WP NO. 218, SLOAN WP NO. 3222-90

Trieste

CYCLOMATIC COMPLEXITY METRICS REVISITED: AN EMPIRICAL STUDY OF SOFTWARE DEVELOPMENT AND MAINTENANCE

Geoffrey K. Gill
Chris F. Kemerer

October 1990

CISR WP No. 218
Sloan WP No. 3222-90

Center for Information Systems Research
Sloan School of Management
Massachusetts Institute of Technology

Abstract

While the need for software metrics to aid in the assessment of software complexity for both development and maintenance has been widely argued, little agreement has been reached on the appropriateness and value of any single metric. McCabe's cyclomatic complexity metric, a measure of the maximum number of linearly independent circuits in a program control graph has been widely used in research. However, despite the widespread interest and popularity of this metric, it has not been without criticism, both analytical (the Myers and Hansen variants) and empirical (the high correlation of cyclomatic complexity with size measures). The current research tested both types of critiques on a newly collected dataset of real world software development and maintenance projects. The analytical research questions were tested on a set of 834 software modules from a number of existing real-time systems. Neither the Myers nor Hansen variants were found to be significantly different from the original value as computed by McCabe. Therefore, these particular theoretical criticism seem to have little or no practical impact, as represented by the data collected in this study. In regard to the empirical research questions, previous concerns were validated on this new dataset. However, the current research proposes a simple transformation of the metric whereby the cyclomatic complexity is divided by the size of system in source statements, thereby determining a "complexity density" ratio. This complexity density ratio is demonstrated to be a useful predictor of software maintenance productivity on a small pilot sample of actual maintenance projects.

CR Categories and Subject Descriptors: **D.2.7 [Software Engineering]**: Distribution and Maintenance; **D.2.8 [Software Engineering]**: Metrics; **D.2.9 [Software Engineering]**: Management; **F.2.3 [Analysis of Algorithms and Problem Complexity]**: Tradeoffs among Complexity Measures; **K.6.0 [Management of Computing and Information Systems]**: General - *Economics*; **K.6.3 [Management of Computing and Information Systems]**: Software Management

General Terms: Management, Measurement, Performance.

Additional Key Words and Phrases: Software Productivity, Software Maintenance, Software Complexity, McCabe Metrics, Cyclomatic Complexity.

Research support for the second author from the Center for Information Systems Research and the International Financial Services Research Center is gratefully acknowledged. Helpful comments were received from Bill Curtis on an earlier draft.

1. INTRODUCTION

A critical distinction between software engineering and other, more well-established branches of engineering is the shortage of well-accepted measures, or metrics of software. Without such metrics, the tasks of planning and controlling software development and maintenance will remain stagnant in a "craft"-type mode, wherein greater skill is acquired only through greater experience, and such experience cannot be easily communicated to the next system for study, adoption, and further improvement. With such metrics, software projects can be quantitatively described, and the methods and tools used on the projects to improve productivity and quality can be evaluated. These evaluations will help the discipline grow and mature, as progress is made at adopting those innovations that work well, and discarding or revising those that do not.

Of particular concern is the need to improve the software maintenance process, given that maintenance is estimated to consume 40-75% of the software effort [Vessey and Weber, 1986]. What differentiates maintenance from other aspects of software engineering is, of course, the constraint of the existing system. The existing system constrains the maintenance work in two ways, the first through the imposition of a general design structure that circumscribes the possible designs of the system modifications, and second, and more specifically, through the complexity of the existing code which must be modified.

A measure, or metric of this latter, more specific type of system influence that has been proposed is the McCabe's cyclomatic complexity metric, a measure of the maximum number of linearly independent circuits in a program control graph [McCabe, 1976]. As described by McCabe, a primary purpose of the metric is to "...identify software modules that will be difficult to test or maintain" [McCabe, 1976, p.308], and therefore is of particular interest to researchers and practitioners concerned with maintenance. The McCabe metric has been widely used in research [Zuse and Bollmann, 1989], and a recent article by Shepperd cites some 63 articles that are directly or indirectly related to the McCabe metric [Shepperd, 1988]. However, despite the widespread interest and popularity of this metric, it has not been without criticism, with an early critique appearing in 1982 [Evangelist], and most recently the comprehensive aforementioned critique by Shepperd [1988].

Shepperd's review highlights two types of criticisms. The first, which he labels "theoretical criticisms," have to do with analytical criticisms of the algorithm by which the cyclomatic complexity metric is computed for software. Most prominent among these are the refinements proposed by Myers [Myers, 1977] and Hansen [Hansen, 1978]. Shepperd also 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.[1] This "empirical criticism" suggests that the additional effort required for computing and understanding the McCabe cyclomatic complexity metric may not be worthwhile in practice.

Therefore, given the need for a complexity metric such as McCabe's in general, but also the well-formed criticism of the McCabe metric in particular, the need for additional research in this area is clear. The purpose of the current research is to empirically test the two main criticisms summarized or raised by Shepperd, and to suggest needed improvements. In regard to the theoretical criticisms of Myers and Hansen, Shepperd does note that it is "arguable whether [they] represent much of an improvement over that of McCabe" (p. 32), since "The majority of modifications to McCabe's original metric remain untested." (p.35) [Shepperd, 1988] The current research directly addresses this call for an empirical test of these variations.

The second research question revolves around the practical usefulness of the metric, given the high correlations noted by some previous research. As noted by Shepperd, concerns about the

[1] See [Basili and Perricone, 1984], [Feuer and Fowlkes, 1980], [Paige, 1980], and [Woodward, Hennell, and Hedley, 1979].

external validity of the data and analyses in some of the previous studies can be used raised to mitigate some of the results, particularly those using data on small programs, often using student subjects. Therefore, these results bear validation on data from actual systems, and, in particular, from data on maintenance projects, since use of the metric for testing and maintenance was one of its author's main stated purposes. In particular, this research seeks not to determine whether Cyclomatic complexity captures all aspects of complexity in one figure of merit, but rather to answer the question raised by Shepperd, as to whether cyclomatic complexity can serve as a "useful engineering approximation" [Shepperd, 1988].

Briefly summarized, the results of this research were as follows. In a test on 834 software modules from a number of existing real-time systems, neither the Myers nor Hansen variants were found to be significantly different from the original value as computed by McCabe. However, in regard to the second set of research questions, the empirical criticism of Shepperd and others was validated, in that the correlation between Cyclomatic complexity and SLOC was found to be .95 for these real world systems. However, a simple transformation involving the cyclomatic complexity metric is proposed, whereby the cyclomatic complexity is divided by the size of system in source statements, thereby determining a "complexity density" ratio. This complexity density ratio is demonstrated to be a useful predictor of software maintenance productivity. on a small sample of actual maintenance projects.

The rest of this paper is organized as follows: Section 2 details the main research questions, and provides references to previous research literature where appropriate. Section 3 outlines the data collection procedures and summarizes the data set used in the research. Results are presented in Section 4, and a concluding remarks are presented in Section 5.

2. CYCLOMATIC COMPLEXITY METRICS

2.1 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.

Analytically, V(G) is the number of predicates plus 1 for each connected single-entry single-exit graph or subgraph.

From the beginning, a number of researchers have proposed variations on the original metric. The two most prominent variations are those of Hansen (1978) and Myers (1977) [Shepperd, 1988]. Hansen [1978] gave four simple rules for calculating McCabe's Complexity:

1. Increment one for every IF, CASE or other alternate execution construct.
2. Increment one for every Iterative DO, DO-WHILE or other repetitive construct.
3. Add two less than the number of logical alternatives in a CASE.
4. Add one for each logical operator (AND, OR) in an IF.

Myers [1977] demonstrated what he believed to be a flaw in McCabe's metric. Myers pointed out that an IF statement with logical operators was in some sense less complicated than two IF