EDITORIAL BOARD

Board Chairman:
Wayne Smith, CQA, Applied Information Development, Inc.

Board Members:
James C. Badger, Lomas & Nettles Information Systems
Phillip N. Bentley, CQA, Federal Reserve Bank
Pamela R. Cowger, CQA, Cowger and Associates
Ed Dreher, CQA, Pizza Hut, Inc.
Shirley Gordon, CQA, Peat, Marwick, Mitchell & Co.

Charles Hollocker, CQA, Northern Telecom, Inc.
Jan Pallenlk, American Greetings
Pat Reilly, CQA, Eastern States Bancard Association
Linda T. Taylor, Taylor & Zeno Systems, Inc.
Gary C. Vennes, CQA, Norwest Information Services, Inc.
William E. Perry, CQA, Quality Assurance Institute, Editor
Martha Platt, Quality Assurance Institute, Assistant Editor

COLUMN EDITORS

Book Reviews—Gary C. Vennes, CQA, Norwest Information Services, Inc.
Education and Chapter News—William E. Perry, CQA, Quality Assurance Institute
In-House Quality Assurance Newsletter Ideas—Ed Dreher, CQA, Pizza Hut, Inc.
The Lighter Side of Quality Assurance—William H. Walraven, CQA

Quality Assurance Ideas That Pay Off—Pamela R. Cowger, CQA, Cowger and Associates
Quality Assurance Surveys—William E. Perry, CQA, Quality Assurance Institute
Questions and Answers—Phillip N. Bentley, CQA, Federal Reserve Bank
Standards—Kenneth J. Muth, Target Stores
Testing—Harry Kalmbach, CQA, McDonnell Douglas Aerospace Information Services Co.

REGULAR CONTRIBUTORS TO QUALITY DATA PROCESSING

Robert Coull, CQA, GTE Data Services, Inc.
Jane P. Dinkins, CQA, Poling & Associates, Inc.
Mark Duncan, Federal Reserve Bank, Dallas
Victor M. Guarnera, Blue Cross/Blue Shield
Jerome B. Landsbaum, CQA, Monsanto Company
Daniel W. Martin, Tultex Corporation
Sara Messina, Manufacturers Hanover Trust Company
Leonard Muhs, New York State Department of Motor Vehicles

Robert E. Nichols, Federal Home Loan Mortgage Corporation
Ron Rizza, CQA, Ford Aerospace & Communications Corporation
W. Charles Slaven, CQA, General Electric Company
Robert J. Walsh, CQA, Avco
Donald E. Willett, RCA Automated Systems Division
Richard Zultner, Zultner and Company

Published 4 times per year by Quality Assurance Institute. Copyright © 1987, Quality Assurance Institute.

2 • Quality Data Processing
MEMO FROM: CHAIRMAN OF THE EDITORIAL BOARD, WAYNE SMITH

OVERVIEW OF SOFTWARE PRODUCTIVITY AND QUALITY MEASURES
Capers Jones, Chairman, Software Productivity Research
Article explains some misconceptions on using common quality and productivity measures, specifically cost per defect and cost per source line. Function points are emphasized for productivity measures; and defects found as a quality measure.

ORGANIZING THE QUALITY ASSURANCE FUNCTION
Author identifies three basic classes of quality assurance organization. These are library management, product verification, and process improvement.

ON THE CYCLOMATIC METRIC OF PROGRAM COMPLEXITY
Eldon Y. Li, California Polytechnic State University School of Business
Article describes the cyclomatic metric including its advantages and disadvantages. An extended version of the cyclomatic metric is introduced.

DEFECTS—TURNING A NEGATIVE INTO A POSITIVE
William E. Perry, CQA, Quality Assurance Institute
Article stresses that defects are the key to improving quality and productivity. Defects must be identified, analyzed, and summarized if they are to be understood, managed, and eradicated.

ORGANIZATIONAL BEHAVIOR IN A QUALITY AND PRODUCTIVITY IMPROVEMENT PROGRAM
David M. Siefert, CSP, CQA, NCR Corporation
Article states that the primary considerations for the success of a program are the attitudes and behavior patterns of the organization in which quality and productivity improvement programs are established.
ON THE CYCLOMATIC METRIC OF PROGRAM COMPLEXITY

By Eldon Y. Li
California Polytechnic State University School of Business

ABSTRACT
Measuring program complexity has been an intriguing topic for discussion since the early 1970s. Many complexity metrics have been proposed for the past decade. Among them, the cyclomatic metric is the easiest to understand and compute. It is also the only one that lends itself to determining the maximum set of independent paths for program testing. In this paper, the cyclomatic complexity metric and its extensions are reviewed. The strengths and weaknesses of the cyclomatic metric are identified. In order to reflect the control-flow complexity more accurately, an extension to the existing cyclomatic metrics is introduced. This extended metric can provide system developers with more information about a program and is proved to be superior to the existing cyclomatic metrics in reflecting program complexity.

INTRODUCTION
For more than a decade, program complexity has received tremendous attention from researchers in computer science. Program complexity is deemed to be a major factor that influences the quality of computer programming. Evidences to support this speculation were provided by such empirical studies as Halstead [1972b], Bulut and Halstead [1974], Funami and Halstead [1975], Cornell and Halstead [1976], Elshoff [1976], Love and Bowman [1976], Fitzsimmons [1976], Fitzsimmons and Love [1978], Schneidewind and Hoffman [1979], Sunohara, Takano, Uehara and Ohkawa [1981]. These authors applied a selected set of metrics to measure program complexity and correlated the metrics with the number of errors which occurred in the measured modules. All of them found that the occurrence of program errors correlates significantly with the complexity of the program being measured.

Measuring program complexity by a metric was first attempted by Halstead [1972a]. Since then, numerous metrics have been proposed.

Among them, the cyclomatic metric of McCabe [1976] is the easiest to understand and calculate. It is also the only one that lends itself to determining minimum test set for program testing. In this paper the cyclomatic metric and its extensions will be reviewed and compared. The strengths and weaknesses of the cyclomatic metric will be identified. Effective use of these metrics will be carefully described.

THE CYCLOMATIC COMPLEXITY METRIC
The cyclomatic complexity metric was proposed by McCabe [1976]. His metric is based on the decision structure of a program and the cyclomatic number [Berge, 1973, p. 16] (also called the cycle rank [Harary, 1969, p. 39], or the nullity [Chan, 1969, p. 65]) of the classical graph theory. The cyclomatic complexity metric, \( V(G) \), as defined by McCabe, is

\[
V(G) = E - N + 2P
\]

where \( E \) is the number of edges (or arcs), \( N \) is the number of vertices (or nodes), and \( P \) is the number of connected components. A component is a subgraph representing an external module either calling or called by another module. For example, consider a main program \( M \) and two called subroutines \( A \) and \( B \) having a control structure shown in Figure 1.
The total graph in Figure 1 is said to have three connected components and each subgraph has only one connected component (itself). Therefore, the cyclomatic complexity numbers are:

\[ V(M) = 3 - 4 + 2(1) = 1, \]
\[ V(A) = 2 - 2 + 2(1) = 2, \]
\[ V(B) = 4 - 4 + 2(1) = 2, \]

and

\[ V(M + A + B) = 9 - 10 + 2(3) = 5. \]

It can easily be shown that \( V(M + A + B) = V(M) + V(A) + V(B). \)

McCabe further demonstrated two alternate ways of finding the complexity number \( V. \) One is to count the number of both inner and outer regions for each subgraph. Notice that there should be one outer region for each subgraph. In fact, if we form a closed subgraph by drawing an imaginary arc from the exit node to the entry node for each subgraph in Figure 1 and count all the inner regions afterward it would yield the same number. We believe that the latter approach is less confusing than the former. For example, Figure 2 shows the closed subgraphs derived from Figure 1. By counting the inner regions \((I_1 \text{ through } I_5)\), we get a \( V(G) \) of 5.

The other way of calculating \( V \) is to count the number of predicate conditions in the program. Then the cyclomatic complexity is:

\[ V(G) = \text{Number of predicate conditions} + 1. \]

The attractive aspect of this method is that one can find the \( V(G) \) directly from the program text without arduously constructing a flow graph. For example, consider the following PL/1 program [Myers, 1979, p. 38]:

\begin{verbatim}
M: PROCEDURE (A, B, X);
   IF ((A>1) & (B=0)) THEN DO;
      X := X/A;
   END;
   IF ((A=2) / (X>1)) THEN DO;
      X := X+1;
   END;
END;
\end{verbatim}

Notice that each "IF" statement in procedure \( M \) has two conditions in its predicate. This type of "IF" statement will be called a compound "IF" construct. In contrast, an "IF" statement with only one condition is called a simple "IF" construct, hereafter. Since each condition in procedure \( M \) contributes one cyclomatic complexity count, the complexity number is thus \( V(M) = 4 + 1 = 5. \)

Figure 3a shows the flow graph corresponds to procedure \( M \) listed above. Notice that it reflects the compound predicate by placing an extra exit edge for the second condition on each alternation node. For the convenience of counting, we substituted a traditional decision symbol for each alternation node and created Figure 3b. It can be seen that Figure 3b is much more readable and understandable than Figure 3a. Therefore, we highly recommend adopting decision symbol in flow-graph construction since it helps not only in counting the number of predicates but also in improving the understandability of the flow graph.
STRENGTHS AND WEAKNESSES OF THE CYCLOMATIC COMPLEXITY METRIC

Although the cyclomatic complexity measure is easy to compute and understand, it has many anomalies. We have summarized all its strengths and weaknesses as follows:

Strengths
1. It is easy to compute from the program text, and flow graph as well.
2. It supports a top-down development process to control module complexity in the design phase, i.e., before actual coding takes place.
3. It lends itself to determining the maximum set of independent test paths.
4. It can be used to control the complexity of program modules. (McCabe recommended that an upper bound of 10 should be used as a guide to control the complexity of program modules. This recommendation was endorsed by Schneidewind and Hoffmann [1979], and Walsh [1979].)
"IF" predicates. Myers clearly demonstrated that this new interval metric can accurately reflect the complexity of various constructs.

Another anomaly indicated by Hansen [1978] is that the cyclomatic complexity metric reflects no "expression" complexity. In other words, "a program with more operators is simply bigger ... (and) ... more complex" and should have higher expression complexity [Hansen 1978]. He therefore proposed a pair of measures. One measures control flow complexity by one plus predicate counts (but unlike Myers' method, a "CASE" predicate is always counted as one), and the other measures expression complexity by operator counts in the program. However, Hansen’s metric is somewhat difficult to compute and only applies to program text.

Another major weakness of the cyclomatic complexity metric is its insensitivity to the level of nesting within various constructs. For example, three "WHILE" loops in succession will result in metric values similar to those for three nested "WHILE" loops. This anomaly was brought forward by Curtis, Sheppard, Milliman, Borst, and Love [1979], but they did not offer any solution to it. Dunsmore and Gannon [1979] also manifested that excessive nesting can lead to circumstances in which it is difficult for programmers to comprehend what must be true for a particular statement to be reached. He further proposed a metric of "average nesting level" [Dunsmore and Gannon, 1980]. In order to calculate this metric, every executable statement in a program must be assigned a nesting level. If statement \( S_i \) is at level \( L \) and statement \( S_j \) is in the range of a loop or a conditional transfer governed by statement \( S_i \), then the nesting level of statement \( S_j \) is \( L + 1 \). For example, the two "IF" statements in procedure \( M \) above are at level one and the two assignment statements at level two. Therefore, the average nesting level is 1.5 (or \( 1 + 2 + 1 + 2/4 \)). Dunsmore and Gannon’s metric suffers the same deficiency as that of Hansen’s. Moreover, it does not measure any control flow complexity, neither does it accurately reflect program nesting complexity.

Recently, Li [1987] proposed a solution to this problem of program nesting. He developed a nesting level metric, \( L(G) \), to accurately reflect program nesting complexity. Li’s metric is an extension to McCabe’s cyclomatic complexity metric. He derived a two-step computation procedure for \( L(G) \) as follows:

1. Find the cyclomatic complexity measure, \( V(G) \), using McCabe’s approach.
2. Identify the second, the third, the fourth, and so forth, levels of nesting construct and assign each occurrence with one additional unit of complexity. Note that a control statement should be considered to be at the first level of a new nested construct if it follows a control statement whose other branches all lead directly to program exit without any interven-
TABLE 1
COMPARISON OF THE OUTCOMES
OF FOUR COMPLEXITY METRICS

<table>
<thead>
<tr>
<th>Algorithm from Kernighan and Plauger [1974]</th>
<th>Hansen</th>
<th>Myers</th>
<th>McCabe</th>
<th>Li</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge two lists: (K&amp;P, pp. 18-19)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original version</td>
<td>(5,10)</td>
<td>(5:5)</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>Improved version</td>
<td>(3,16)</td>
<td>(3:5)</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Computer dating service: (K&amp;P, pp. 21-22)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original version</td>
<td>(7,10)</td>
<td>(7:7)</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Improved version #1</td>
<td>(3,12)</td>
<td>(3:6)</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>Improved version #2</td>
<td>(3,6)</td>
<td>(3:3)</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A checkers move generator: (K&amp;P, pp. 41-42)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original version</td>
<td>(14,60)</td>
<td>(17:17)*</td>
<td>17</td>
<td>25</td>
</tr>
<tr>
<td>Improved version</td>
<td>(10,46)</td>
<td>(10:17)*</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>Julian to Gregorian date conversion: (K&amp;P, pp. 43-46)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Original version</td>
<td>(17,85)</td>
<td>(17:19)**</td>
<td>19</td>
<td>30</td>
</tr>
<tr>
<td>Improved version #1</td>
<td>(5,25)</td>
<td>(5:10)**</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Improved version #2</td>
<td>(6,25)**</td>
<td>(6:11)**</td>
<td>11**</td>
<td>13</td>
</tr>
</tbody>
</table>

*Computed GOTO construct is regarded as CASE construct.
**One unit of complexity is introduced by an additional DO-WHILE construct.

For example, consider the six control-flow graphs depicted in Figure 4. The nesting level of each predicate is identified with a number to the right of each decision symbol. All predicates in Graphs (b), (c), and (f) are considered to be at the second or higher level of a nested construct. In contrast, the predicates following the first-level predicate in Graphs (b), (c), and (f) are considered to be at the second or higher level of a nested construct.

Notice that the new nesting metric, \( L(G) \), is now sensitive to the level of nesting but no longer able to determine the maximum number of independent test paths. Three characteristics of the \( L(G) \) metric are worth mentioning here. First, the \( L(G) \) metric assumes that nested constructs are more complex than simple construct but a nested control statement having one subsequent control statement and all other branches leading directly to the end of the program does not contribute additional complexity. Second, the \( L(G) \) penalizes the excessive use of nested construct and encourages substituting the "CASE" statement for the nested "ELSE IF" construct. Third, the \( L(G) \) converges to the cyclomatic complexity metric if there is no nested construct in the program.

**COMPARISON OF FOUR COMPLEXITY METRICS**

In order to compare the cyclomatic metric and its extensions, four algorithms from Kernighan and Plauger [1974] were measured. The outcomes of applying four complexity metrics are illustrated in Table 1. After scrutinizing the outcomes of these four metrics, four major characteristics are notable:

1. The difference between the two elements in Myers' metric indicates the number of Boolean logical operators, such as "AND," "OR," and "XOR."
2. The first element of Hansen's and Myers' metrics are the same if the program contains no "CASE" construct.
3. The only metric that does not reflect program improvement correctly is McCabe's metric. Although program improvement might introduce more operators, the control flow complexity should definitely be reduced.
4. The difference between Li's and McCabe's metrics indicates the number of nesting levels. When these two metrics are equal, the program contains no nested construct.

**RECOMMENDATIONS**

A closer look at Table 1 reveals that no single metric can reflect the entire complexity of a program. However, each metric does have its own major strengths. While Hansen's first element measures the number of predicative statements in the program, Myers' metric reflects the number of Boolean logical operators. McCabe's \( V(G) \) indicates the max-
Taking from Hansen [1978], a control-flow metric may include four different counts:

1. The number of "IF," "CASE," or other alternate execution constructs.
2. The number of iterative "DO," "DO-WHILE," "REPEAT-UNTIL," or other repetitive constructs.
3. Two less the number of alternatives in a "CASE" construct.
4. The number of Boolean logical operators such as "AND," "OR," "XOR" in a control statement.

While Hansen's first element includes only the first two numbers, Myers' first element includes the first three, and McCabe's V(G) includes all four. Hansen calls these three metrics in sequence: "CYC-MIN," "CYC-MID," and "CYC-MAX." Therefore, Hansen's metric is the "CYC-MIN," Myers' is the "CYC-MID," and McCabe's is the "CYC-MAX."

Since the differences among the former three metrics can provide useful information, we recommend that all three metrics, namely, "CYC-MIN," "CYC-MID," and "CYC-MAX" be recorded during the process of finding the L(G) metric. In addition, we suggest that the "CYC-MIN" be broken down into three numbers: the counts of "IF," "CASE," and repetitive ("WHILE" or "REPEAT") constructs. Moreover, program improvement sometimes results in decomposing a program into two modules. Therefore, it is also useful to show the number of modules encompassed by a program. As for operator count, we believe that it cannot reflect any control-flow complexity and thus shall be excluded.

In conclusion, the optimal cyclomatic complexity metric should be of two tiers and eight elements: the first tier contains the numbers of four different counts: 1) modules, 2) "IF" constructs, 3) "CASE" constructs, and 4) repetitive ("WHILE/REPEAT") constructs; the second tier contains 1) CYC-MIN, 2) CYC-MID, 3) CYC-MAX, and 4) L(G). The values of all eight elements in this metric can be readily obtained during the process of finding the L(G) value. To be more specific, the number of modules and control statements (i.e., "IF," "CASE," "WHILE," and "REPEAT" statements) must be found before CYC-MIN can be obtained. Adding to CYC-MIN by two less the number of alternatives in each "CASE" construct, we find CYC-MID. The number of logical operators plus CYC-MID yields CYC-MAX. Finally we find the value of L(G) by adding the count of second- or higher-level nested control statements to CYC-MID.

Table 2 gives the outcomes of applying this two-tier metric to the same algorithms in Table 1. Several characteristics are worth noting:

1. The first element of the second tier, CYC-MIN, equals the sum of the four elements in the first tier.
2. The first two elements of the second tier are the same if the program contains no "CASE" construct.
3. The difference between the second and the
third elements in the second tier indicates the number of Boolean logical operators, i.e., "AND," "OR," and "XOR."

(4) The last two elements of the second tier are the same if the program contains no nested construct.

(5) The two tiers can supplement each other. If the first tiers of two improved versions are alike, we can always tell the better one by examining the two second tiers. For example, refer to the second and the last examples given in Table 2. Each example gives two improved versions. It is now obvious to choose the first improved version for the second example. For the last example, we will choose the first improved version because it contains less repetitive and nested constructs (the fourth elements of the two tiers are smaller).

CONCLUSION

We have reviewed the cyclomatic complexity metric and its extensions. Strengths and weaknesses of these metrics were also discussed. It was demonstrated that no single software metric can measure the complexity of a program in its entirety. A two-tier, eight-element metric (#modules, #IF, #CASE, #WHILE/REPEAT and CYC-MIN, CYC-MID, CYC-MAX, L(G)), was therefore recommended. This practice allows software developers to understand software without looking at its program text. It can be used to compare program modules of different software projects. So can it be used to compare multiple improved versions of a program. It is better than the cyclomatic metric \( V(G) \) in measuring program complexity. It supplies more information than any other metric alone. While offering so many more advantages over the other metrics, this two-tier, eight-element metric remains easy to compute and understand. Since the information about each element in this metric must be extracted before one can obtain the \( L(G) \) value, finding the value of each element becomes a matter of retaining the readily available information. It is obvious that this extended metric is superior than any existing cyclomatic metrics in reflecting the control-flow complexity of a program.

REFERENCES


Cornell, L. and M. H. Halstead, "Predicting the Number of Bugs Expected In A Program Module," Technical Report CSD-TR-205, Computer Sciences

TABLE 2
OPTIMAL TWO-TIER EIGHT-ELEMENT CYCLOMATIC COMPLEXITY METRIC

<table>
<thead>
<tr>
<th>Algorithm from Kernighan and Plauger [1974]</th>
<th>First Tier**</th>
<th>Second Tier**</th>
</tr>
</thead>
<tbody>
<tr>
<td>Merge two lists: (K&amp;P, pp. 18-19)</td>
<td>(1,2,0,2)</td>
<td>(5,5,5,7)</td>
</tr>
<tr>
<td>Original version</td>
<td>Improved version</td>
<td></td>
</tr>
<tr>
<td>Computer dating service: (K&amp;P, pp. 21-22)</td>
<td>(1,3,0,3)</td>
<td>(7,7,7,8)</td>
</tr>
<tr>
<td>Original version</td>
<td>Improved version #1</td>
<td>(3,3,6,6)</td>
</tr>
<tr>
<td>Improved version #2</td>
<td></td>
<td>(3,3,3)</td>
</tr>
<tr>
<td>A checkers move generator: (K&amp;P, pp. 41-42)</td>
<td>(1,12,1,0)</td>
<td>(14,17,17,25)</td>
</tr>
<tr>
<td>Original version</td>
<td>Improved version</td>
<td>(2,8,0,0)</td>
</tr>
<tr>
<td>Julian to Gregorian date conversion: (K&amp;P, pp. 43-46)</td>
<td>(1,16,0,0)</td>
<td>(17,17,19,30)</td>
</tr>
<tr>
<td>Original version</td>
<td>Improved version #1</td>
<td>(1,3,0,1)</td>
</tr>
<tr>
<td>Improved version #2</td>
<td></td>
<td>(1,2,0,3)</td>
</tr>
</tbody>
</table>

**First tier contains (#modules, #IF, #CASE, #WHILE/REPEAT).

**Second tier contains (CYC-MIN, CYC-MID, CYC-MAX, L(G)).
Department, Purdue University, 1976.


Harary, F., Graph Theory, Reading, MA: Addison-Wesley, 1969.


DP HISTORY AND TRIVIA

In 1844, Samuel Morse conducted his first public demonstration of his electric telegraph... In 1951, the UNIVAC I arrived at the U.S. Census Bureau...

In 1961, Robert Noyce was awarded a patent for the integrated circuit...

In 1968, Digital Electronics Corporation announced the PDP-8, the first 8-bit minicomputer...

In 1974, a programming error caused $7.5 million in welfare checks to be issued to people no longer qualified to receive them...

In 1978, Corvus Systems, Inc. announced the first eight-inch disk drive that used the Winchester technology...

In 1979, the PC spreadsheet VisiCalc was demonstrated to the public at the National Computer Conference... 


LIST OF ADVERTISERS

We on the staff of Quality Data Processing sincerely appreciate these advertisers' confidence in our new journal.

Cover 2 On-Line Software International
Cover 3 Mathews & Co.
Cover 4 BST
Page 1 QAI
Page 3 Applied Computer Research
Page 6 Softool
Page 11 Applied Information Development
Page 18 Burrill-Ellsworth Associates, Inc.
Page 19 XA Systems

July 1987 • 23