Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences, 1985.

EXPERIMENTAL EVALUATION OF THE DATA DEPENDENCY GRAPH FOR USE IN MEASURING SOFTWARE CLARITY

JAMES M. BIEMAN
Department of Computer Science
Iowa State University
Ames, Iowa 50011
(515)294-9939

WILLIAM R. EDWARDS
Computer Science Department
University of Southwestern Louisiana
Lafayette, Louisiana 70504
(318)231-6284

Abstract

A data dependency graph (DDG) was constructed from each member of a testbed of program segments. The rooted spanning tree complexity (RSTC) and cyclomatic complexity of each DDG was computed. The DDG based complexity measures were compared to the software science effort measure, the control flow cyclomatic complexity measure, and subjective judgements of program clarity. The measures appear to be sensitive to clarity except for the control flow cyclomatic complexity measure. One of the test shows that the RSTC measure appears to be most sensitive to the spaghetti code resulting from unstructured branches.

1. INTRODUCTION

The data dependency graph (DDG) has been proposed as a model of software complexity that can be used as a basis for complexity measures [Bieman84]. The DDG is a directed graph representation of the possible data dependencies in a program. A DDG node is used to represent each variable definition and the edges represent possible direct dependencies between definitions. The cyclomatic complexity and the rooted spanning tree complexity (RSTC) of the DDG are possible program complexity measures. The cyclomatic complexity (CC) is well known in application to the control flow graph [McCabe76]. The cyclomatic complexity is computed from the number of nodes (N), edges (E), and connected components (C); CC = E - N + C. The RSTC is the number of distinct spanning trees with a specified root contained in a directed graph. It is known in graph theory [Berge73] that the RSTC using node x_1 as the root can be calculated as the determinant of A where A is a form of an adjacency matrix:

$$A = \begin{vmatrix} \sum_{i \neq 2} a_{i,2} & -a_{2,3} & \cdots & -a_{2,n} \\ -a_{3,2} & \sum_{i \neq 3} & \cdots & -a_{3,n} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ -a_{n,2} & -a_{n,3} & \cdots & \sum_{i \neq n} a_{i,n} \end{vmatrix}$$

and $a_{i,j}$ is the number of edges directed from node x_i to node x_j.

Empirical results suggest that programmers trace backwards through programs when debugging [Weiser82]. Therefore,by reversing edge directions and letting program outputs be roots, the resulting tree count provides a meaningful measure of tracing effort and logical complexity.

The data dependency model and associated complexity

measures have been applied in a set of tests in order to evaluate the model and metrics. Each test compares measures applied to two or more program segments that represent solutions to the same problem. The objectives of the experimentation were to

(1) evaluate the difficulty of applying the DDG model and metrics,

(2) compare results of applying the DDG metrics with the subjective human judgement of program clarity, and

(3) compare the DDG based metrics with other measures.

The difficulty of constructing the model and calculating the values of the metrics may be considered to be the *automatability* of the metrics. The compiler should be able to compute the measurements at the coding stage. However, the measures should also be usable at the earliest stage in the software development cycle. Ideally, a scanner could be implemented that would compute values for the metrics from the earliest specification documents or from a formal design. For the DDG construction and metric evaluation to be automatable, a computationally tractable algorithmic procedure for construction and evaluation must exist. The methodology will not be practical for large scale application unless the DDG construction and metric evaluation is automatable.

The metrics, to be useful, must distinguish between implementations that vary in psychological complexity. The metrics should enhance the unquantifiable and often contradictory subjective judgment of human experts. Measures of software complexity should provide the tools to discover quantifiable characteristics of software clarity.

The DDG based metrics are compared with other metrics that have been described in the literature. The comparison is made to determine if the DDG metrics describe anything new about software complexity. We use the software science effort measure [Halstead77] and the flow of control cyclomatic complexity [McCabe76] as metrics for comparison.

The application of the model to the testbed programs is not expected to provide a formal proof of the models usefulness. The experiment is designed to provide initial experimental results, a basis for future empirical research, and intuitive support.

2. TESTBED PROGRAMS

The development of software complexity measures should lead to increased knowledge of how to quantify previously unquantifiable intuition concerning software clarity. A program that is clear to one individual may be confusing to another person. Programs were sought that were subjectively judged as to relative clarity with the clarity judgements supported in the literature.

Gordon compared implementations of a collection of algorithms that were subjectively evaluated as to relative clarity in an attempt to validate software science measures [Gordon79]. He selected two or more implementations of 46 problems from the literature. In each case, one of the implementations is judged subjectively to be clearer than the other. Gordon's paper provides a convenient set of program segments from which to evaluate the data dependency model. In addition to reporting the clarity rankings between program segments, Gordon has taken measurements of software science metrics and calculated the software science effort value for each program segment. The programs and program segments in Gordon's testbed were taken from sources that are often used for classroom examples of proper and improper programming style. For this validation effort, nineteen program segments that are solutions to nine problems from Gordon's testbed were selected. Table 1 lists the sources of the program segments.

Program Segment No	Gordon's Reference No. [Gordon79] & Orig. Source of Program Segment
1a	6A and 7A; [Kernighan74] p.306
1b	6B and 8A; [Kernighan74] p.306
1c	7B and 8B; [Kernighan74] p.307
2a	9A; [Kernighan74] p.307
2b	9B; [Kernighan74] p.307
3a	11A; [Kernighan74] p.311
3b	11B; [Kernighan74] p.311
4a	16A; [Kernighan74] p.315
4b	16B; [Kernighan74] p.316
5a	28A; [Wirth74] p.254, fig.20
5b	28B; [Wirth74] p.254, fig.22
6a	27A; [Wirth74] p.253, fig.17
6b	27B; [Wirth74] p.253, fig.16
7a	29A; [Wirth74] p.256, fig.34
7b	29B; [Wirth74] p.257, fig.37
8a	13A; [Kernighan74], p.312
8b	13B; [Kernighan74], p.312
9a	32A; [Knuth74], p.271, ex.4a
9b	32B; [Knuth74], p.271, ex.4

Table 1. Source of Testbed Programs

The selected program segments were chosen to illustrate more than one solution of the same problem with a significant variation in clarity and a persuasive subjective judgement as to the clearest. In addition, the program segments had to be short so that the DDG could be constructed manually and the features of the DDG could be visible.

3. EXPERIMENTAL PROCEDURE

For each testbed program segment, the following procedure was followed:

(1) The Halstead effort measure calculated by Gordon was recorded.

(2) The flow of control cyclomatic complexity was calculated by counting the number of decisions.

(3) The data dependency graph was constructed.

(4) The cyclomatic complexity and RSTC of each DDG was computed.

Calculating the cyclomatic number is straightforward - only the number of edges and nodes are necessary for the calculation.

The RSTC was calculated using the following procedure. First, a new node labeled OUTPUT was added to the

DDG and edges were drawn to the OUTPUT node from each node representing definitions that may be output. Included as output definitions were

(1) all output parameters and definitions that are found in output statements, and

(2) all DDG sink nodes.

Using the OUTPUT node as the root, the direction of the edges was reversed, the appropriate adjacency matrix was built, and the determinant of the matrix was calculated.

To illustrate the procedure, the process, as performed for Program Segment 1a is presented. Program Segment 1a, which follows, is a segment of a FORTRAN program designed to calculate the smallest value of three variables:

```
      IF(X .LT. Y) GO TO 30
      IF(Y .LT. Z) GO TO 50
      SMALL = Z
      GO TO 70
30    IF (X .LT. Z) GO TO 60
      SMALL = Z
      GO TO 70
50    SMALL = Y
      GO TO 70
60    SMALL = X
70    --
```

Gordon calculated the Halstead E measure of Program Segment 1a to be 2008 and, as McCabe notes, the control flow cyclomatic complexity is the number of decisions plus one. Using DDG algorithms [Bieman84], the DDG for the program segment was constructed.

To calculate the RSTC of Program Segment 1a, the code was examined to determine which of the definitions represent possible outputs of the program segment. The variable SMALL is the only output variable and at the end of the code segment there are three live definitions of SMALL. An output node was added to the DDG and edges constructed from the nodes representing the three live definitions of SMALL to the output node.

The DDG of Program Segment 1a with the output node and edges added is illustrated in Figure 1 below.

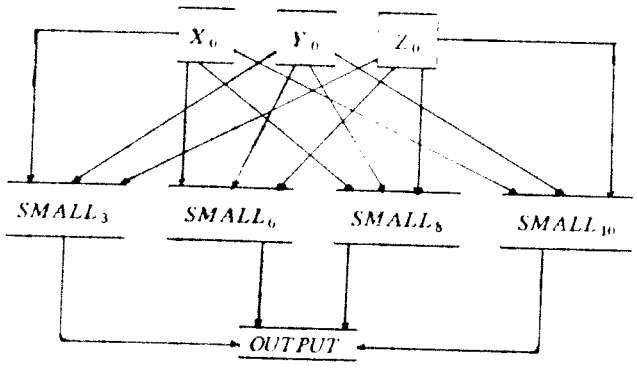

Figure 1. DDG of Program Segment 1a With Output Node

The DDG cyclomatic number is the number of edges minus the number of nodes plus one. For Program Segment 1a the result is six. Note that the DDG cyclomatic complexity is computed before the addition of the output node and edges.

The calculation of the RSTC of Program Segment 1a continues by reversing the direction of the edges. Then the adjacency matrix was constructed using the output node as the root. The determinant of the matrix is the RSTC of the graph. The matrix and determinant for Program Segment 1a is shown in Figure 2 below.

$SMALL_3$	1	0	0	0	-1	-1	-1
$SMALL_6$	0	1	0	0	-1	-1	-1
$SMALL_8$	0	0	1	0	-1	-1	-1
$SMALL_{10}$	0	0	0	1	-1	-1	-1
X_0	0	0	0	0	4	0	0
Y_0	0	0	0	0	0	4	0
Z_0	0	0	0	0	0	0	4

$= 64$

Figure 2. Matrix for Calculating the RSTC of Program Segment 1a

The experimental procedure described in this section was applied to each of the program segments included in the experiment. The results are presented in the following section.

4. RESULTS

Table 2 summarizes the results of the experiment. For each testbed program segment, the table shows

(1) the Halstead Effort measure,

(2) the flow of control cyclomatic number,

(3) the DDG cyclomatic number, and

(4) the DDG rooted spanning tree complexity.

Program Segment	Halstead Effort Number	Control Cyclo. Comp.	DDG Cyclo. Comp.	RSTC
1a	2008	4	6	64
1b	300	3	1	6
1c	54	1	0	1
2a	1869	5	26	77,760
2b	1020	3	9	384
3a	6263	4	25	2100
3b	4833	4	14	240
4a	4507	3	10	96
4b	3485	3	10	162
5a	4445	4	17	300
5b	2931	4	16	480
6a	4275	5	26	1260
6b	1974	4	20	320
7a	3447	3	9	32
7b	2821	3	12	80
8a	952	3	7	9
8b	620	3	6	4
9a	2605	5	9	360
9b	2160	4	5	40

Table 2. Metrics From Testbed Program Segments

In each test version A is described as less clear than the version B. For Test 1, which has three program versions, version B is described as less clear than version C. Relative clarity is the reported subjective judgment of Kernighan, Wirth, or Knuth as indicated in Table 1.

The results were examined to compare and contrast the four measures and nineteen program segments. First, the sensitivity of the measures to subjective program clarity is analyzed. Then the cases were examined to find explanations for instances when specific metrics fail to distinguish between implementations with different degrees of clarity. Next, rank ordering of the program segments by different measures were compared. Finally the specific cases where the measures vary widely were studied.

Table 3 summarizes the success of each metric in supporting the subjective clarity judgements.

Metric	% of Clear/Unclear Program Segments That the Metric Identified Correctly
Halstead E	100 %
Rooted Spanning Tree Complexity	66.7 %
DDG Cyclomatic Complexity	77.8 %
Control Cyclomatic Complexity	44.4 %

Table 3. Success of Each Metric in Supporting Subjective Clarity Ranking of Testbed Program Segments

In each case the Halstead E measure distinguished between clear and less clear implementations. However, one must keep in mind that the purpose of Gordon's study was to support the software science metrics.

The RSTC was unable to match the reported subjective clarity ranking in Test 4, Test 5, and Test 7. The DDG cyclomatic complexity measure was unable to match the clarity ranking in Test 4 and Test 7. A close look at the reasons given for the clarity judgements in the experiment reveals the subjective nature of "clarity" and provides explanations for the inability of the DDG based metrics to match the human ratings.

In the three cases of disagreement the reported rankings are a result of semantic differences rather than clarity or readability differences. Program 4a, which is designed to calculate the sine function, fails to initialize three of the variables. Program Segments 5a and 5b are designed to input a sequence of numbers, select the first n distinct numbers, and assign the numbers to an array with n elements. Program Segment 5b is described as "more economical" than Program Segment 5a because, in Program Segment 5a, a loop exit test is not always necessary [Wirth74]. Program Segments 7a and 7b are designed to multiply two non-negative integers using only addition, doubling, and halving. Program Segment 7a is criticized because the program segment may cause a semantic error on some machines in the form of numerical overflow [Wirth74].

The Halstead E measure agreed with the subjective rankings in Test 4, Test 5, and Test 7. The Halstead E measure is computed from simple counts of the number of operators (N1), operands (N2), distinct operators (n1), and distinct operands (n2). Three of the four counts are lower in Program Segment 4b than Program 4a. Two of the counts are lower in Program Segment 5b than Program Segment 5a and one of the counts is lower in Program Segment 7b than Program Segment 7a while the remaining counts are identical. The Halstead E measure tends to be lower when there are fewer operators and operands.

The flow of control cyclomatic complexity fails to distinguish relative complexity in half of the cases, probably because of its small range of values. Recall that the value depends only on decision count. Out of all nineteen testbed programs and program segments the flow of control cyclomatic complexity only had four different values.

The relative sensitivity to clarity of each measure is illustrated by Table 4. Table 4 shows the ratio of each of the measures of unclear to clear versions. The most notable feature of the table is the sensitivity of the RSTC measure. In each of the cases where RSTC can distinguish relative clarity, the ratio of the RSTC of the unclear version to the clear version is significantly higher than the ratio for the E measure. We may note that the "explosive" nature of the RSTC (at worst case it may be exponential in the number of nodes) may imply that the logarithm of the RSTC should be used for some applications.

Program Segments Compared	Halstead Effort	Control Cyclo. Comp.	DDG Cyclo. Comp.	RSTC
1a/1b	6.69	1.33	6.00	10.67
1b/1c	5.56	3.00	undefined	6.00
2a/2b	1.83	1.67	2.89	202.50
3a/3b	1.30	1.00	1.79	8.75
4a/4b	1.29	1.00	1.00	.59
5a/5b	1.51	1.00	1.06	.63
6a/6b	2.16	1.25	1.30	3.94
7a/7b	1.22	1.00	.75	.40
8a/8b	1.53	1.00	1.17	2.25
9a/9b	1.21	1.25	1.80	9.00

Table 4. Ratio for Each Measure of Unclear/Clear Program Versions

The rank order of each testbed program segment was computed using the value of each of the measures. The rank ordering shows whether programs that have comparatively high values for one measure have high values for the other measures. Table 5 presents the rank of each program segment using each measure. The number seven in the entry for Halstead E rank for Program Segment 6b means that Program Segment 6b had the seventh lowest E value of the 19 programs. The rank ordering shows how the measures differ. Many of the rankings of control flow cyclomatic complexity are identical because of the small number of values of the measure. The results

indicate that the control cyclomatic complexity measure is relatively insensitive to subtle differences in program complexity. The flow of control cyclomatic complexity measure was the measure that was least able to differentiate between implementations with different clarity.

Program Segment Number	Halstead Effort Rank	Control Cyclo. Comp. Rank	DDG Cyclo. Comp. Rank	RSTC Rank
1a	8	10	4	7
1b	2	2	2	3
1c	1	1	1	1
2a	6	17	19	19
2b	5	2	17	15
3a	19	10	17	18
3b	18	10	13	11
4a	17	2	10	9
4b	14	2	10	10
5a	16	10	15	12
5b	12	10	14	16
6a	15	17	18	17
6b	7	10	16	13
7a	13	2	7	5
7b	11	2	12	8
8a	4	2	6	4
8b	3	2	4	2
9a	10	17	7	14
9b	9	10	3	6

Table 5. Testbed Program Segments Ranked by the Metrics

The most significant discrepancy in the rank order of measures occurs in Program Segment 2a. For Program Segment 2a the data dependency complexity is very high while the E measure is relatively low; the program segment ranks sixth for E measure and nineteenth for both data dependency measures. A investigation of Program Segment 2a may explain the discrepancy:

```
      DO 10 I=1,M
      IF(BP(I)+1.0)19,11,10
11    IBN1(I) = BLNK
      IBN2(I) = BLNK
      GO TO 10
19    BP(I) = -1.0
      IBN1(I) = BLNK
      IBN2(I) = BLNK
10    CONTINUE
```

Because of the computed go to statement in the program segment, the data dependency complexity is high. Each variable definition is dependent on many other definitions because of the complex flow of control. Since the E measure is based only on the counts of operators and operands it does not account for the added complexity resulting from the complex interactions among variables.

5. CONCLUSIONS

A cursory evaluation of the results suggest that E is the measure most closely related to clarity. However, the DDG based measures appear sensitive to some aspect(s) of complexity that E is not. A short program with few operators and operands can be complex. Program Segment 2a shows one case where E is relatively low and RSTC and DDG cyclomatic complexity are extremely high for a short nine line program. The flow of control cyclomatic complexity measure appears to be relatively insensitive to clarity.

While performing the experiment, some deficiencies were observed in the DDG based measures. The value of both the DDG cyclomatic complexity and the RSTC measures are not affected by the addition of a single node connected to the rest of the DDG by a single edge. Also, the RSTC is insensitive to loops from a node to itself.

The experiment demonstrated that data dependency graphs from actual programs could be built and the DDG based metrics could be evaluated. The output variable of a complete program could be determined algorithmically by the DDG sink nodes, actual output statements, assignments to global data structures, and the output parameters.

Perhaps the most serious problem with the experimental evaluation is the subjectivity of human judgement of program clarity. In each case that the DDG based measures fail to match the human clarity judgement, one can argue that the human was not really judging clarity. Program clarity must be defined objectively in order to conclusively validate complexity metrics. The development of objective measures of clarity will be of significant value in validating software complexity measures.

The application of the model and metrics to a the set of testbed programs represents initial prototype experimentation. The prototype experiment will aid in the design of more comprehensive future empirical work. Future empirical work would hopefully examine a relatively large set of industry programs. A standard testbed of software examples would be useful in order to compare different metrics and techniques.

One would need to compare the DDG based metrics to additional measures related to clarity or complexity such as the number of bugs found. The subjective human judgements of clarity used in the experiment are still opinion and opinions are subject to debate.

Automated measurement tools would be required to analyze a significant set of industry programs. Since DDG construction requires modification to a parser, the tool development phase of a large scale empirical project is not insignificant. Since "empirical evidence typically takes years to collect" [McCabe82], an empirical evaluation of a large sample of industry programs must be an ongoing data collection effort requiring substantial support in both dollars and manpower. Another more formal approach to validation would be a controlled experiment involving human subjects. Because of the need for specialized psychological, statistical, and experimental

design expertise, rigorous empirical studies involving human subjects suggests the use of interdisciplinary research teams [Curtis83].

The experimentation indicates a direction for future theoretical work. The measurement of graph complexity in the abstract should be studied. Each measure of graph complexity that was used fails to measure some aspects of complexity. None of the metrics, except Halstead's E, measure the addition of one node and edge; RSTC fails to measure the complexity of self loops. Further work on measuring abstract graph complexity appears to be promising.

REFERENCES

[Berge73] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, The Netherlands, 1973.

[Bieman84] Bieman, J.M., *Measuring Software Data Dependency Complexity*, Ph.D. Dissertation, Computer Science Department, University of Louisiana, Lafayette, Louisiana, 1984.

[Curtis83] Curtis, Bill, "Software Metrics: Guest Editor's Introduction," *IEEE Transactions on Software Engineering*, SE-9, 6 (1983), 637-638.

[Gordon79] Gordon, R.D., "Measuring Improvements in Program Clarity," *IEEE Transactions on Software Engineering*, SE-5,2(1979), 79-90.

[Halstead77] Halstead, Maurice H., *Elements of Software Science*, Elsevier, New York (1977).

[Kernighan74] Kernighan, Brian W. and P.J. Plauger, "Programming Style: Examples and Counterexamples," *ACM Computing Surveys*, 6, 4, December, (1974), 303-319.

[Knuth74] Knuth, Donald E., "Structured Programming With GO TO-statements," *ACM Computing Surveys*, 6, 4, December (1974), 261-301.

[McCabe76] McCabe, Thomas, "A Complexity Measure," *IEEE Transactions on Software Engineering*, SE-2, 4 (1976), 308-320.

[McCabe82] McCabe, Thomas, *Structured Testing: A Testing Methodology Using the McCabe Complexity Metric*, NBS Special Publication, Contract NB82NAAK5518, U.S. Department of Commerce, National Bureau of Standards, (1982).

[Weiser82] Weiser, Mark, "Programmers Use Slices When Debugging," *Communications of the ACM*, 25, 7, July 1982,446-452.

[Wirth74] Wirth, Niklaus, "On the Composition of Well-Structured Programs," *ACM Computing Surveys*, 6, 4, December (1974), 247-260.