03 Jan 1981

A Complexity Measure Based On Nesting Level

Warren A. Harrison

Kenneth I. Magel
Missouri University of Science and Technology

Recommended Citation

W. A. Harrison and K. I. Magel, "A Complexity Measure Based On Nesting Level," *ACM SIGPLAN Notices*, vol. 16, no. 3, pp. 63 - 74, Association for Computing Machinery (ACM), Jan 1981.
The definitive version is available at https://doi.org/10.1145/947825.947829

A COMPLEXITY MEASURE BASED ON NESTING LEVEL

Warren A. Harrison
Kenneth I. Magel

Computer Science Department
University of Missouri-Rolla

For the past several years an accepted method of determining the complexity of computer programs has involved developing a direct graph, G=(V,E) which represents the flow of control of the program. The directed graph, G consists of a set of nodes, V which represent "blocks" of groups of code, and a set E, of edges which corresponds to the flow of control among the various nodes. The graph is usually restricted to having one initial node which is always executed first. In addition, each block has two properties:

(1) No transfer occurs into the interior of the block from outside.

(2) If the first statement of the block is executed, all the statements of the block are executed.

McCabe [3] has published one method which has been widely accepted. His method requires the calculation of the number of basic paths within the program - the smallest set of paths that, when taken in combination may serve to generate every possible path in the graph. This is calculated as the cyclomatic number, V(G), using the following formula:

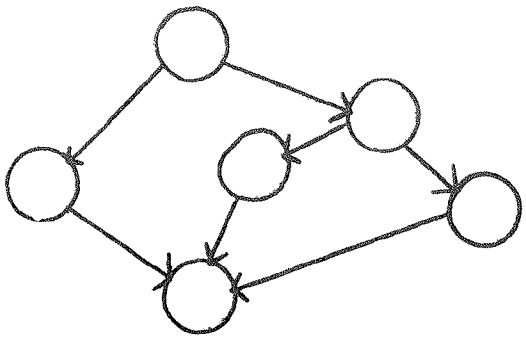
$$V(G) = e - n + 2$$

where e represents the number of edges, and n represents the number of nodes in the control flow graph.

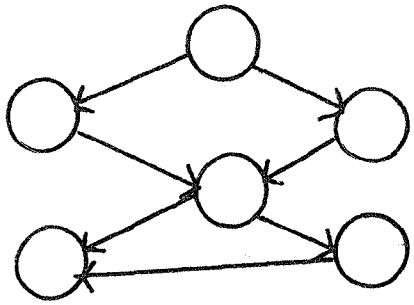
The calculation of the cyclomatic number proves to be an effective complexity measure. However, because the cyclomatic measure only counts the number of basic paths, it is incapable of recognizing the effects of two major complexity factors which can be intuitively seen to increase program complexity. These two items are the complexity of the individual blocks within the program - which we shall refer to as "program magnitude", and the program.

Halstead [1] has proposed several methods of measuring program magnitude. However, a measure of program magnitude alone sone not satisfy the need to account for the level of nesting within control structures. For example, assuming that each node or block, S_i of the following code segments is of comparable complexity,[1] the first segment of code is obviously more complex by virtue of the nesting of the if-then-else's.

```
if P1 then
      if P2 then S1;
            else S2;
      else S3;
S4;
```



```
if P1 then S1;
      else S2;
if P2 then S3;
S4;
```



However, neither the McCabe nor the Halstead metrics take into account the additional complexity of hte nesting. Using McCabe's measure, the first segment has a V(G) of 3. The second code segment also has a V(G) of 3.

Because Halstead's measure simple determines the number of operators, operands and the total use of each, and since both segments contain two selection statements, and three blocks of instructions, the complexities as determined by Halstead's meausre would alos be indistinguishable.

However, by examining the control flow graph of a computer program we can determine the nesting of each block of node. This can be done by using some concepts from the study of lattices. However, it is important to recognize that the control flow graph of a computer program is not a lattice since the relation defined by the set of edges is not necessarily a partially ordered set [2].

We may say that a node x precedes a node if there is a path from node x to node y. We may write this as x<y. If there is an edge from node x to node y, then we may say x immediately proceeds y, written x<<y.

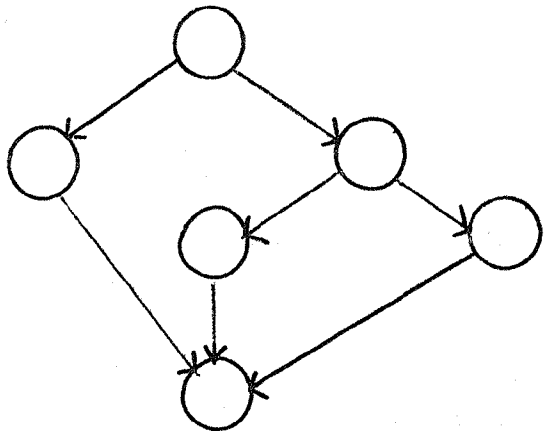
Now consider a subgraph G' of a control flow graph G. An element m in G is said to be an upper bound for G' if it precedes all nodes in G'. An element n in G is said to be a lower bound of G' if it succeeds every node in G'/ If there exists an upper bound of G', then m' is the least upper bound of G'. Likewise, if there exists a lower bound of G', n' which precedes all other lower bounds of G', n' is the greatest lower bound of G'.

Often, a node will exist in G with an outdegree of two or greater. We shall refer to such a node as a selection node.

Even though we cannot determine every path possible from a selection node since it may be an infinite number if we allow backward branches, we can determine the nodes that lay upon any possible parh from a selection node using the above concepts. If we accept the fact that the complexity of a control structure such as an if-then-else is dependent upon the statements within its range, then knowing what nodes lie upon the paths within its range can be useful.

We may determine which nodes lie within the selection node's range by forming a subgraph of all the nodes which lie between the selection node itself, and the greatest lower bound of the subgraph formed of all nodes which immediately succeed the selection node. This subgraph, G' will contain every node which lies within the range of the selection node. For example:

```
if P1 then
      if P2 then
            S1;
         else
            S2;
      else
         S3;
S4;
```



It can be said that all nodes, except for S_4 lie within the range of the node P_1 (i.e., the outer if-then-else construct). The greater lower bound for this construct is S_4. Likewise, S_1 and S_2 also lie withinthe range of node P_2. For P_2 S_4 also is the greatest lower bound. The subgraph G' for P_1 contains the nodes S_3, P_2, S_1 and S_2. The subgraph G' for P_2 contains the nodes S_1 and S_2.

We may utilize this technique to compute the complexity of a computer program by assigning each node a "raw" complexity value which would consist of the Halstead measure for that node. In addition, each node would possess an additional complexity measure shich we shall refer to as that node's "adjusted" complesity.

The adjusted complexity of a given node may be calculated using the following procedure:

First, the subgraph G' of each selection node (i.e., outdegree >1) should bedetermined.

After the subgraph G' is formed (note that it does not contain the greatest lower bound) the adjusted complexity for for selection node is computed by summing the raw complexity of every node within the subgraph G', and adding the raw complexity of the selection node itself.

For all other nodes (i.e., outdegree $\leq$1) the adjusted complexity is set equal to the node's raw complexity.

The sum of the adjusted complexities of all the nodes in the control flow graph is then used as a measure of the total programs complexity.

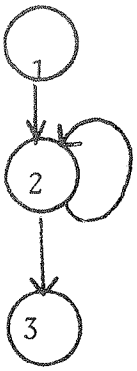
This complexity measure is capable of recognizing the effects of both the level of nesting within control structures and the program magnitude. The level of nesting is determined via the use of the control flow graph, and the progrma magnitude is taken into account by using the Halstead measure for each component node (i.e., the node's raw complexity).

This lends itself to a more intuitively satisfying complexity metric than either McCabe's or Halstead's metrics by themselves. While both McCabe's metric and the one presented here make use of control flow graphs, it should be noted that the similarity ends there. MCCabe is actually interested only in the basic paths, while the current complexity metric used the control paths fo the graph only to identify which nodes contribute to other node's conplexities.

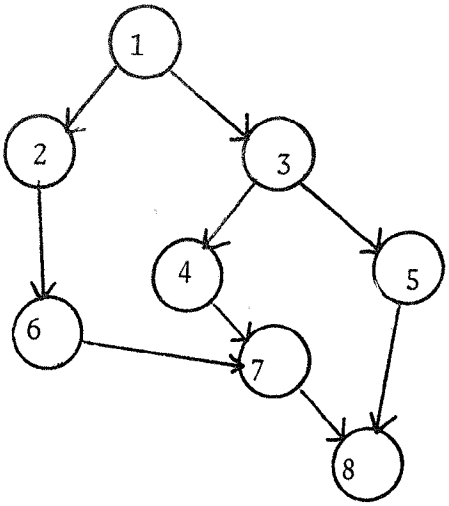
On the following pages are some comparisions of the measures calculated by both McCabe's measure and the method presented in this paper. The flow graphs used are taken from examples given in McCabe's original paper.

While the raw complexity of each node would ordinarily reflect that node's Halstead measure, raw complexities of 1 have been assigned to every node to allow comparisions between the two techniques (since McCabe does not take into account the complexity of individual nodes).

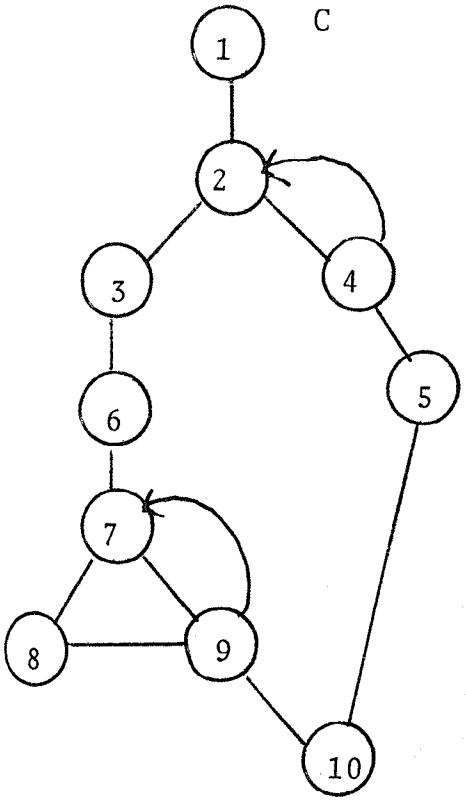
A



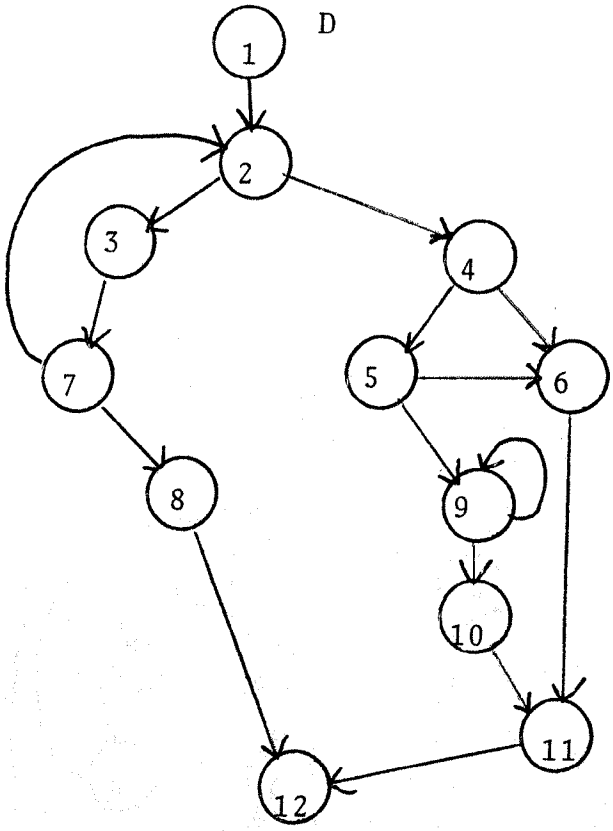
B

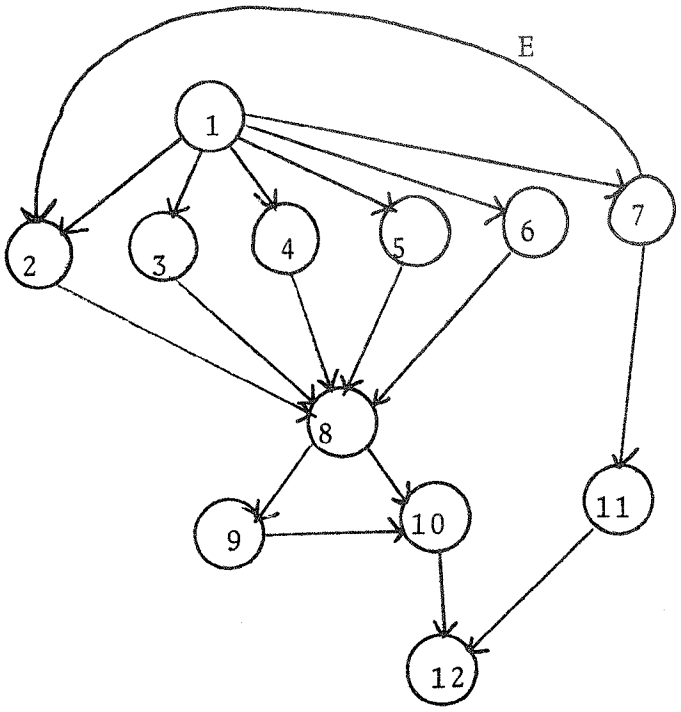


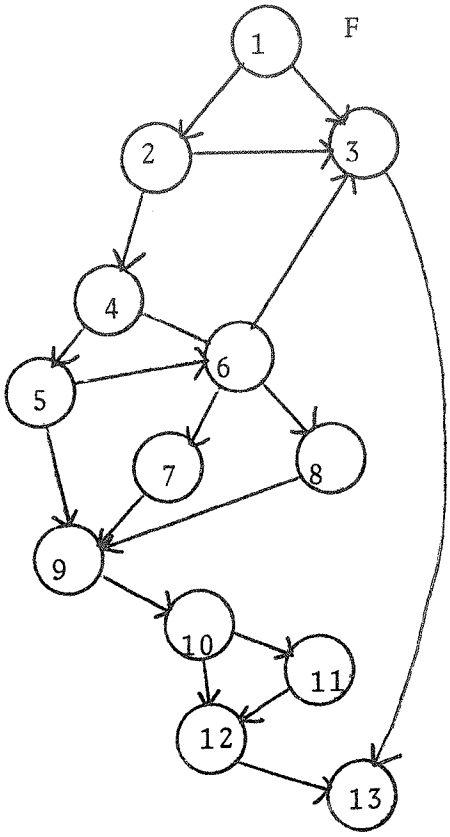
C

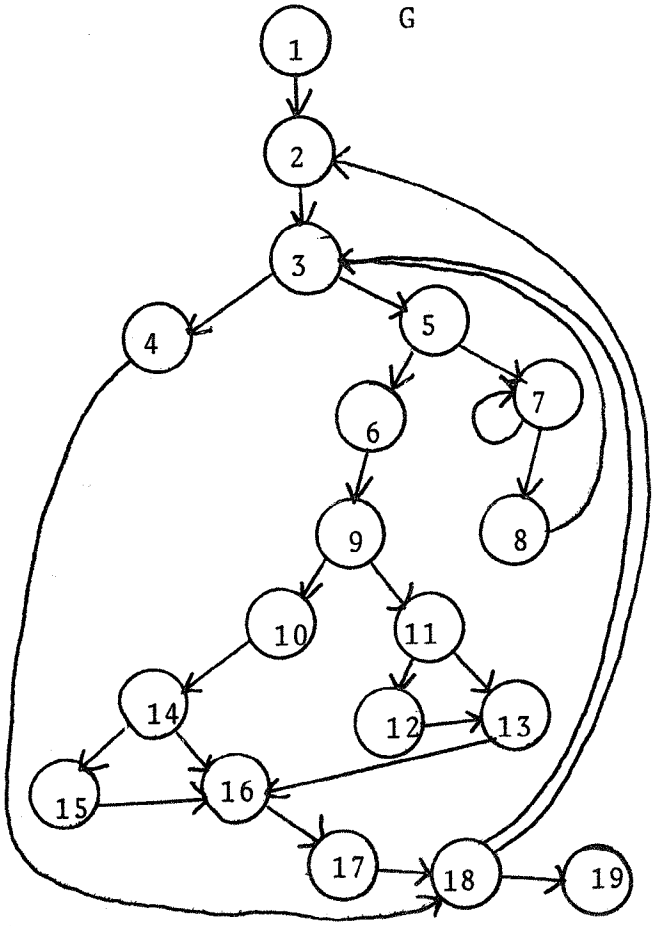


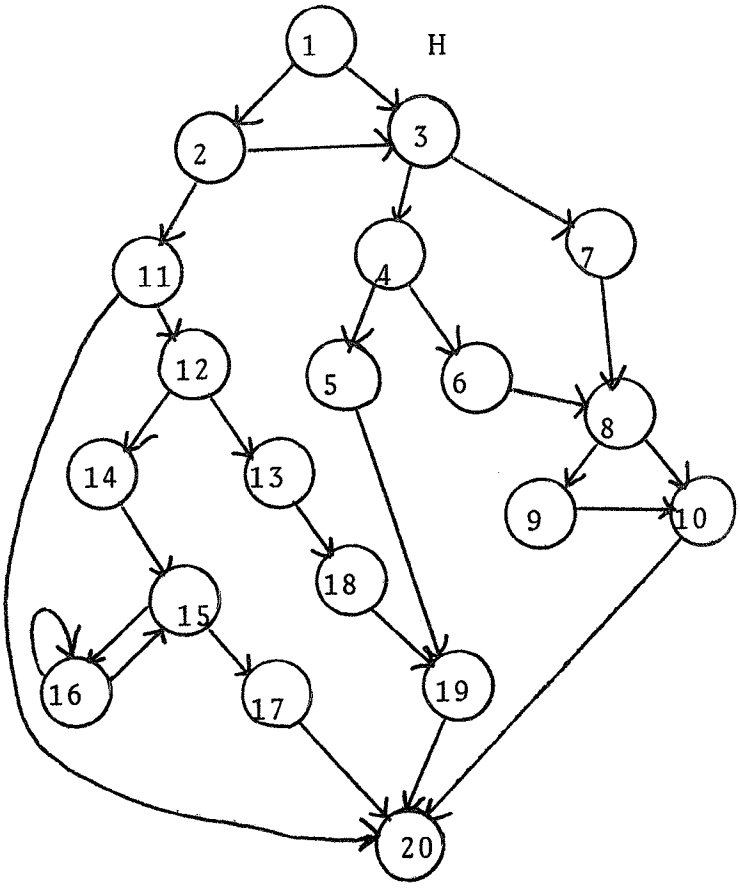
D

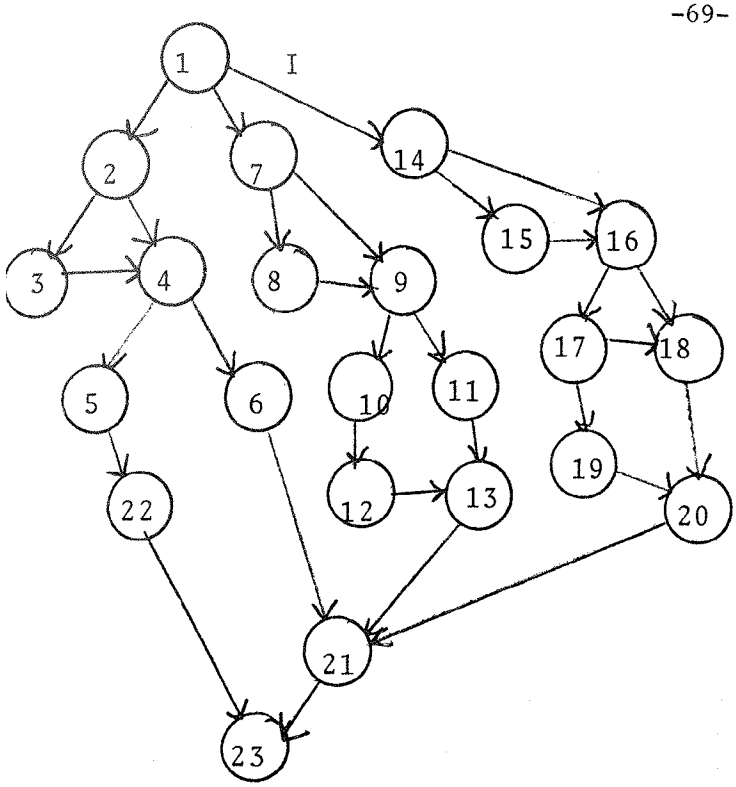


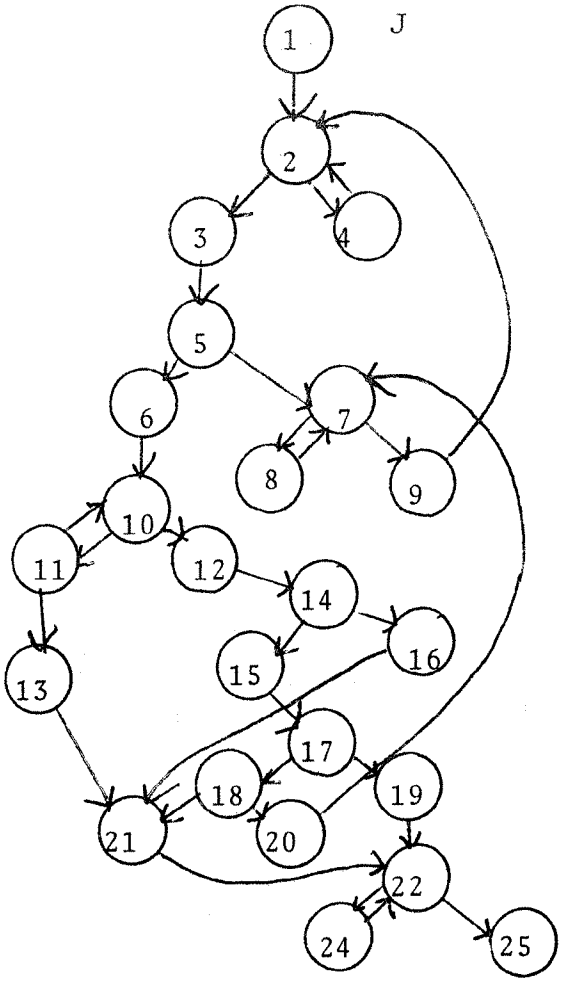

E
1
2
3
4
5
6
7
8
9
10
11
12

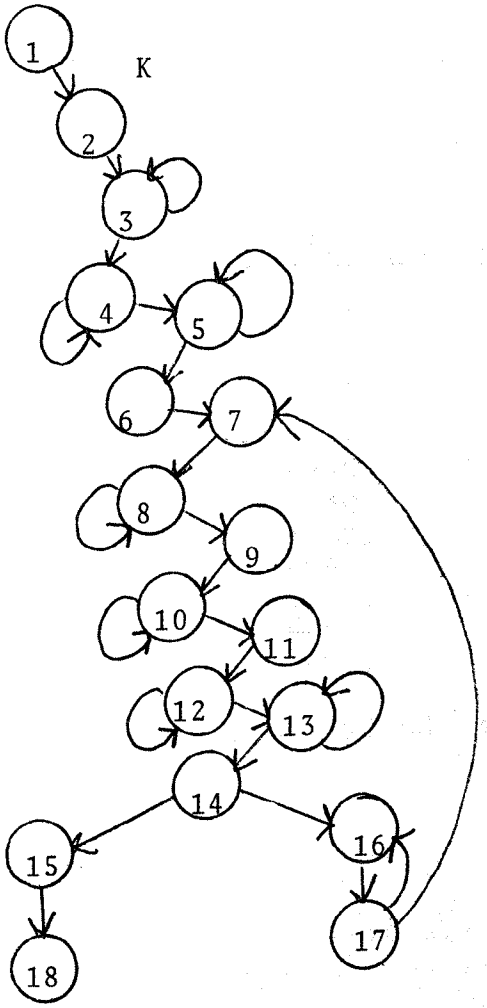

F
1
2
3
4
5
6
7
8
9
10
11
12
13

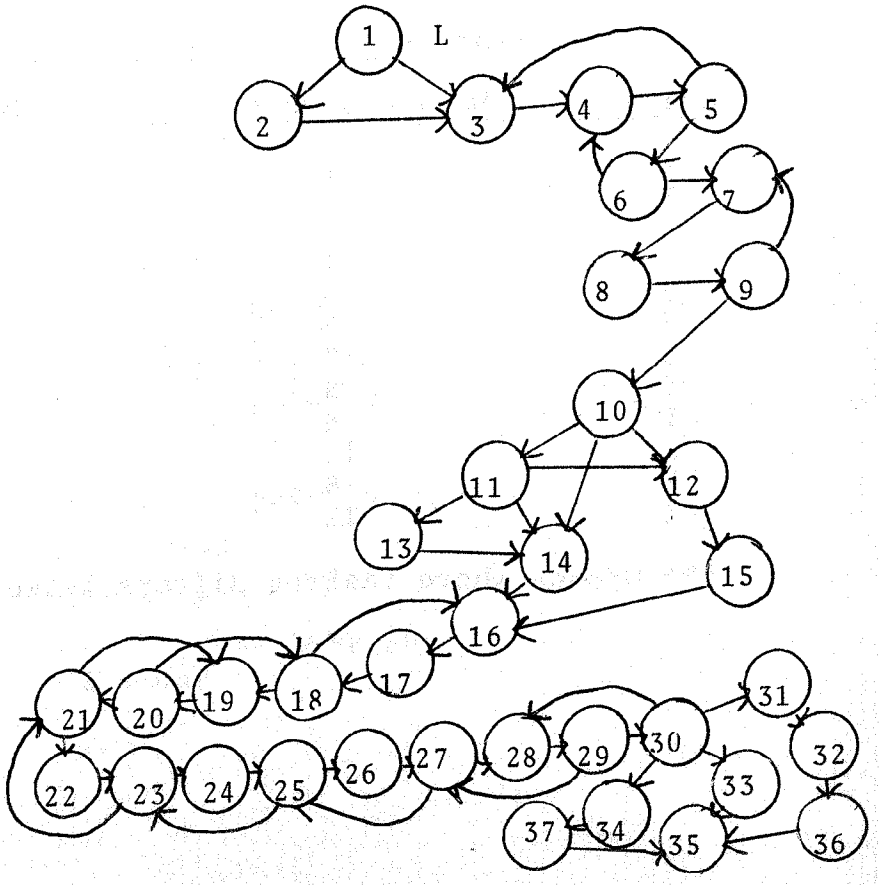

G
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19


H
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20


I
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23


J
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
24
25


K
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18


L
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37

Complexity Calculations

Graph	McCabe's Measure	New Complexity Measure*
A	2	3
B	3	15
C	5	24
D	6	26
E	8	28
F	8	59
G	9	78
H	10	81
I	10	53
J	11	186
K	10	39
L	19	126

* Whenever a graph contains two consecutive nodes where executing the first node implies executing the second and vice versa, these nodes have been merged for determination of the new complexity measure.

Table 1

Ranking of Programs by Complexity Measure

Graph	McCabe's Measure	New Complexity Measure
A	1	1
B	2	2
C	3	3
D	4	4
E	5	5
F	5	8***
G	7	9***
H	8	10***
I	8	7***
J	11	12***
K	8	6***
L	12	11***

*** Graphs where ranking differs between the two methods

Table 2

The actual values obtained for the two complexity measures are not very significant since we assume the complexity of each node was one. What is significant is the ranking of the programs in order of increasing complexity presented in Table 2. For the simple graphs A through E, the two measures agree on rankings. For the more complex graphs, the two measures rank the graphs in different orders. The rankings of graphs F and K are particularly different.

McCabe's complexity measure and the one discussed in this paper capture different notions of complexity. McCabe attempts to determine the number of execution paths in a program. This type of measure provides an indication of the difficulty which would be encountered in debugging a program using testing. The new measure attempts to determine how difficult the static program text would be to understand. This type of measure provides an indication of the difficulty which would be encountered in proving a program correct.

Consider graph F. The McCabe measure ranks this graph as simpler than any of G through L. The new measure considers K and I to be more complex. The following programs exhibit the same control structure as F, I and K respectively.

Program Corresponding to F

```
If p1
    then S3
         goto S13
    else if p2
             then goto S3
             else if p4
                     then case p6 of
                           (1): goto S3
                           (2): S7
                                goto S9
                           (3): S8
                                goto S9
                           esac
                     else if p5
                             then gotp p6
    S9
    if p10
         then s11
    S12
    S13
```

Program Corresponding to I

```
CASE p1 of
   (1): if p2
           then S3
        if p4
           then S5
                S22
                goto S23
           else S6
   (2): if p7
           then S8
        if p9
           then S10
                S12
           else S11
        S13
   (3): if p14
           then S15
        if p16
           then if p17
                   then S19
                   else;
           else S18
        S20
     esac
S21
S23
```

Program Corresponding to K

```
S1
S2
DO S3 while(condition)
DO S4 while(condition)
DO S5 while(condition)
S6
S7
DO S8 while(condition)
S9
DO S10 while(condition)
S11
DO S12 while(condition)
DO S13 while(condition)
IF p14
    then S15
    else S16
         if p17
         then goto S16
         else goto S7
```

Where the conditions for repetition of statements S3, S4, S5, S8, S10, S12, and S13 were not indicated in the original graph.

The program corresponding to K is the easiest to read. Except for the branch back to S7 at the end, it can be read from top to bottom. The program corresponding to I can be read as a Case statement followed by two simple statements. The three cases within the case statement each involve two levels of nested IFs. The first also has a goto to outside of the Case statement. The program corresponding to F has three levels of nested IF's with a case statement within the lowest level IF. Further, there are six goto's which make the program very difficult to read.

REFERENCES

1. Halstead, Maurice H., Elements of Software Science, Amsterdam, The Netherlands: North-Holland, 1977.

2. Kildall, Gary A., "A Unified Approach to Global Program Optimization", First ACM Conference on Principles of Programming Languages, Boston, October, 1973, pp. 194-206

3. McCabe, Thomas J., "A Complexity Measure", IEEE Transactions of Software Engineering, Vol. SE-2, No. 4 (Dec 1976).