IEEE Region 10 Conference on Computer and Communication Systems, September 1990, Hong Kong

Software Development Techniques -- Combining Testing and Metrics

Chi-Ming Chung

Tamkang University

Tamshui, Taiwan, R.O.C.

Abstract

Software metrics and testing methodologies are widely applied in evaluating and assuring software quality. However, the existed metrics and testing methodologies are developed independently. Since most testing strategies and metrics are derived from similar program factors, such as control flow or data flow factors, software development methodologies can be developed by considering both testing and metrics factors to be be utilized in testing and measurement of software to form a more effective software development tool. This will significantly reduce the costs in constructing automated tools for testing and metrics by removing the redundant work to construct both. Path Complexity Techniques (PCT) are proposed for guiding testing and measuring software complexity. Two test critera : intra level first and inter level first are presented. The idea of the most complicated path is also illustrated.

1. Introduction

The improvement of software development techniques has received great attention due to the significant increasing software demand. One of the primary goals of these techniques is to improve software software quality and reliability. Two important approaches to assure the reliability and software quality are considered. One is through software testing to minimize errors, and the other is utilizing software metrics to monitor the software development process.

Static testing and dynamic testing are two major approaches in software testing. Dynamic testing is divided into functional testing and structural testing. Functional testing, also called black box testing, tests are constructed based upon the program's functional properties, ignoring its internal structure. Structural testing, also called white box testing, the internal control structure or data dependencies are used to develop testing methodologies to conduct testing. McCabe's structured testing [1] and Woodward's test effectiveness hierarchy [2] are examples of the control structure testing, and Ntafos's required k-tuples criteria [3] and Rapps's testing criteria (all-defs, all-p-uses, all-uses, etc.) [7], are examples of the data dependencies testing.

Software metrics is generally classfied into size metrics, control flow metrics, and data flow metrics [8]. Lines of code and Halstead's software science [4] are examples of size metrics. McCabe's cyclomatic complexity [5] and Conte's average nesting level [6] are examples of control flow metrics. Dunsmore's live variables and Chung's live definitions [8][9] are examples of data flow metrics. The utilization of software metrics and testing methodologies to control software quality is in Figure 1.

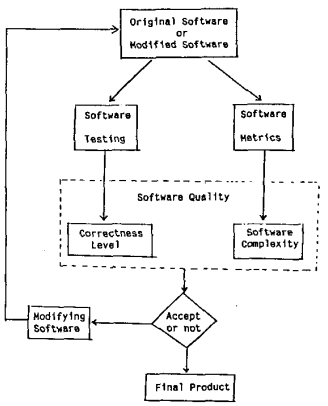

Figure 1.

It is well known that to determine whether a test data set will exercise all possible paths in a program or not is, in general, computationally undecidable. Because of the difficulty to automatically exercise all possible paths, Path Complexity Techniques(PCT) are proposed. PCT combines the idea of metrics and testing methodologies that can be applied in testing to guide test paths selection as well as software complexity measurement. This significantly reduce the costs for developing CASE (Computer Aided Software Engineering) tools, since the development of CASE tools is very expensive.

In the next Section, software metrics and testing methodologies are illustrated. Section 3 introduces the idea of path complexity which is the basis of Path complexity techniques. In Section 4, Path complexity techniques and two test criteria are expressed, and algorithms for finding the most complicated path are proposed. At last, the conclusion of this research is in Section 5.

2. Testing and Metrics

Softeware metrics and testing methodologies are expressed in this Section. Since most of these techniques are derived from similar program factors, such as control flow and data flow factors, techniques derived from different program factors are selected and introduced.

2.1 Testing Methodologies

2.1.1 Control Flow Oriented Testing

A. McCabe's Structured Testing

McCabe [1] proposed a structured testing methodology based on the control flow graph and the idea of cyclomatic complexity [5]. To accomplish a structured testing of a program P, the following criteria need to be met.

1. Every branch of each decision in P must be exercised at least once.

2. The least number of distinct paths needs be exercised is v. Where the value of v is the cyclomatic compelxity of P.

The first rule implies that every reachable statement in the program will be exercised at least once, and the second rule tells the minimum number of paths to be tested. Two drawbacks arise in structured testing; One is that there is no justification why v paths need to be tested, the other is that the number of paths can be tested are higher than v in most cases.

B. Test Effectiveness Hierarchy

The test effectiveness hierarchy proposed by Woodward [2] is an extention of general hierarchy based on the notation of of Linear Code Sequence And Jump (LCSAJ). A LCSAJ consists of a list of statements which can be executed sequentially and is terminated at a jump (e.g. goto, if then else, while and so on). For example, in program 1, there are four LCSAJ's as shown in Figure 2.

Program 1

```
1. I = 0
2. READ *,J
3. DO 20 K = 1,J
4.   I = I + J
5.   CONTINUE
6.   M = I * J
7.   STOP
8.   END
```

LCSAJ'S of Program 1

1. 1-2-3-4-5-6-7-8.
2. 1-2-3-4-5.
3. 3-4-5.
4. 3-4-5-6-7.

Figure 2.

This hierarchy is defined in terms of Test Effectiveness Ratio (TER). TER(1) and TER(2) are the same as CO and C1 in Miller's hierarchy [14]. TER(1) requires every statement in a program to be exercised at least once; TER(2) requires every branches in a program to be exercised at least once. Note that both of them are independent of LCSAJ.

$$TER(3) = \frac{\text{number of LCSAJ's exercised at least nce}}{\text{total number of LCSAJ's}}$$

$$\vdots$$

$$TER(n+2) = \frac{\text{(number of distinct subpaths of length nLCSAJ's exercised at least once plus the number of distinct complete paths of length less than or equal to n LCSAJ's exer cised at least once)}}{\text{(total number of distinct subpaths of lengthn LCSAJ's plus the total number of distinctcomplete paths of length less than or equal to n LCSAJ's)}}$$

TER(3) is the first level of LCSAJ's hierarchy based on the idea of LCSAJ. In program 1, four LCSAJ's need to be tested to obtain full coverage of TER(3). TER(4) is based on the subpath with no more than two contiguous LCSAJ's. For example, 1-2-3-4-5-3-4-5 and 3-4-5-3-4-5-6-7 are two subpaths containing two contiguous LCSAJ's in program 1. Similarly, TER(n+2) is based on the subpath with no more than n contiguous LCSAJ's.

2.1.2 Data Flow Oriented Testing

It is well recognized that data dependency is an important factor of testing complexity, it can be used to provide program testing. Several testing methodologies based on the data dependency of a program have been proposed and discussed [3][16][17].

2.2 Software Metrics

2.2.1 Size Metrics

Halstead proposed a family of metrics called software science [Halstead 77]. In software science, a program is treated as a group of tokens. Tokens are divided into two classes, one is operands, the other is operators. Operands represent data such as variable or value; otherwise is operarors, such as +,-,*,\. Halstead's metrics are based on the counts of operators and operands. The basic counts are n1, n2, N1, and N2. n1 (n2) are the number of unique operators (operands) in a program. N1(N2) are the total number of operators (operands) in a program.

A family of metrics are derived from the basic counts: vocabulary, size, length, volume, and effort.

2.2.2 Control flow based metrics

Control flow complexity metrics are derived from the control structural of a program. A number of empirical results indicate that control flow complexity has related to the complexity of a program [1][20][21].

A. McCabe's Cyclomatic Complexity

McCabe's cyclomatic complexity [1] is a well known metric. It has been widely accepted and applied in measuring software complexity as well as in conducting testing [McCabe 82]. The cyclomatic complexity can be derived from control flow graph

PG, which is defined as:

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

where e is the number of edges and n is the number of nodes in G. The cyclomatic complexity can be also derived from the number of decisions in a program.

B. Average Nesting Level

In structured languages, the control flow could be highly nested. For instance, nested loops or nested if structures. Intutively, the higher nested level will yield higher complexity. The average nested level measure is to assign more weights in higher nested level. That is, a statement not in a nested structure has weight 1; a statement with one outer nested structure has

weight 2. In general a statement with n outer nested structures has weight n+1. The average nested level is to sum up the weights in each statement and then divided by total number of statements [6].

2.2.3 Data flow based metrics

Data flow based metrics are concerned about the inter and intra module's data dependency complexity. Numerous studies show that the data dependency of a program has a significant effect on the programer's tracing, debugging, and understanding capability [18][22].

A. Dunsmore's Live Variables Metrics

Dunsmore proposed average number of live variables[18] per statement based upon the following live variable definitions : A variable is live from its first to its last reference within a procedure. Experimental results indicate that Dunsmore's metric has good correlation with software complexity[18].

B. Chung's Live Definitions and Live Variables

Chung proposed a family of dynamic data flow metrics based on the idea of live definitions and live variables [8][12][13]. A definition is occured in a statement where a value is assigned. The definition of a variable x is said to reach at the top (bottom) of a block (node) i iff there exists a definition clear path with respect to x from the definition to the top (bottom) of i. A definition clear path with respect to x is a path with no redefinition of x. A definition of x is live at the top of block (node) i, if the definition can reach i and there is a reference of the definition afterwards. A variable x is said to live at the top (bottom) of block (node) i, if there exists at least one live definition of x. The number of live variables or live definitions in a block or program are used in measuring the complexity of that block or program.

3. Path Complexity

In structural testing, the number of paths in a program could be exercised might be infinite. Since it is impractical to exercise all paths in a program, techniques to guide the selection of testing paths become important. Path complexity is utilized to establish a testing order for path selection. The idea is a path with higher path complexity is more error prone. That is, by testing higher path complexity path first, we might be able to detect program errors earlier. In addition, based on the tested paths and the tested results, a program tester is in a better situation of choosing the next testing path than that based on the first testing path being selected randomly. Furthermore, the complexity of the most complicated path is a good metric in measuring the complexity of the program which contain the path. According to McCabe's experimental results, programs with higher complexity have higher error rates [1][5]. The definition of path complexity and the most complicated path is described below.

A program graph (control flow graph) is derived from a program; it is defined as a directed graph PG = (N,E,s,t), where N is a set of nodes reachable from the start node, E is a set of directed edges representing the control flows, s N is the start node and t N is the terminal node with no edge going out. The statements within a node must be executed from the first statement and exit form the last statement of the node. A complete path is a path which has start node as its first node and exit node as its last node. Path complexity is defined as the sum of the node complexity of each node of a path. Since node complexity is to measure the complexity of a node, the techniques of software metrics are applied in measuring node complexity. Several possible choices of node complexity measurements are listed below. ICom(i) represents the compleixty of node i, and ACom(i) is the maximum accmulated complexity form the node i to the exit node.

A. Size based metrics

1. LOC : ICom(i) = the number of lines of code in node i.
2. Volume: ICom(i) = the volume in node i.
3. Effort: ICom(i) = the effort measure of node i.

B. Control flow based metrics

1. Edges: ICom(i) = the number of in-out edges.
2. Decision: ICom(i) = 1 if i is a decision node i, otherwies ICom(i) = 0.
3. Nested level: ICom(i) = the nested level of node i.

C. Data flow based metrics

1. Reaching definition:
   ICom(i) = the number of reaching definition at the top of node i.
2. Live variables:
   ICom(i) = the number of live variables at the top of node i.
3. Live definitions:
   ICom(i) = the number of live definitions at the top of node i.

Since the number of paths in a program with loops is infinite, the paths in a program can be classified into different groups: loop(0), loop(1), ... loop(n), where 0..n is the number of duplicated back-edges [Chung 89c]. A most complicated path with respect to loop(n) is defined as a complete path which has higest path complexity in the groups of loop(n).

4. PCT Methodologies

The idea of path complexity can be applied in the area of metrics and testing. In metrics application, similar to Dunsmore's average live variables and Chung's average live definitions [18][13], the average path complexity measure is proposed. Averge path compleixty within one level is to sum up all path complexity in each path and then divided by the number of paths within this level. Also, The most complicated path could be another candidate metrics.

In testing applicaton, since the total number of paths can be tested in a program is computational undecidable, most testing methodologies base upon basic testing units for constructing testing methods. Some of these basics are: a statement, a branch, or a LCSAJ. The basic unit for the proposed method is a path.

There might be enormous number of paths in a program graph with loops. The paths can then be classfied into loop(0), loop(1),...loop(n). It is obviously that the higher value of n, the more complicated method is needed to determine path complexity of the paths in it. Also, when n is fixed, the total number of paths is fixed. Based on this, a PCT hierarchy is established. When applying PCT, PCT(0) means considering only loop-free paths, and PCT(1) means considering up to loop-once paths, and PCT(n) considering up to loop n times paths. A PCT testing hierarchy is proposed as below:

PCT(0) : every loop-free path needs to be exercised at least once.

PCT(1) : every loop-once path needs to be exercised at least once.

.
.
.

PCT(n) : every loop-n-times path needs to be exercised at least once.

Two testing criteria for conducting PCT testing are: intra level first criterion and inter level first criterion. These two criteria are assuming that the paths need to be exercised are in groups loop(0) to loop(n).

1. Intra level first criterion

Paths in loop(i) are exercised according to the descending path complexity order, and paths in loop(i+1) cannot be exercised until all paths in loop(i) are tested.

2. Inter level first criterion

Paths in loop(i) are exercised according to descending path complexity order, and for all groups from loop(0) to loop(n) each time exercise one path.

Intra level first algorithm

```
Do  i := 0 to n
  begin

  repeat

   in loop(i)
   select the path P with highest path complexity;
   conduct test;
   remove P from loop(i);
  until  loop(i) is empty;

   end;
```

Inter level first algorithm

```
Repeat

do  i := 0 to n
begin

  in loop(i)
  select the path P with highest path complexity;
  conduct test;
  remove P from loop(i);

end;

until  loop(0)..loop(n) is empty;
```

This testing hierarchy provides users a good framework for selecting proper testing correctness level. This research presented two algorithms for deriving PCT from the first two levels: PCT(0) and PCT(1). They are based on loop-free and loop-once paths, respectively. A loop-free path is a path with no duplicated node. A loop-once path is a path with no duplicated back-edge. Algorithms for searching the most complicated loop-free path and loop-once path are presented in the next two sections respectively.

4.1 Loop-free MCP

The idea of finding the most complicated loop-free path is to measure the maximum accumulated complexity (ACom) form each node to the exit node by depth first visiting. The path with the maximum path complexity is the most complicated loop-free path.

The algorithm for finding the most complicated loop-free path is described as follows:

1. Visit a node x in depth first order.
2. If x does not have any son (immediate successor), then x is the exit node.
3. Otherwise, select one of the sons of x,say y.

   a) If y has been visited before and its ACom is not obtained, then a loop is encountered. Go back to step 3 to traverse the next back.
   b) If y has been visited before and its ACom is obtained, then a cross-edge is encountered. Go back to step 3 to find and traverse an unvisited son.

   c) If all sons of x have been checked, select a son which has the greatest ACom, say the node s. Set the ACom of x as the sum of the ACom of s and the internal complexity of s. Mark the branch from x to s as IN_PATH indicating that s is in the most complicated path. If all sons of x do not have their ACom value, then x must be in a loop and its ACom cannot be assigned.

After traversing x, according to the depth first traversing, we need to go back to a father of x and check a sibling of x. If x is the entry node, then the process of setting ACom is complete. The most complicate path is the path composed of nodes connected by IN_PATH branches. The complexity of the path is the ACom of x.

4.2 Loop-once MCP

The algorithm for finding the most complicated loop-once path is constructed based upon the loop-free algorithm. The process of setting the ACom of each node in the depth first visiting order is still applied. The only difference is in the action taken when a loop is found. In the loop-free algorithm, the ACom value of the nodes in a loop is unassigned. In loop-once algorithm, the method called cycle duplication is used to enforce that each back-edge encountered is traversed only once.

The cycle duplication method is illustrated as follows. Recall that in the loop-free algorithm's step 3) part a), a cycle is encountered.

1) Identify all nodes in the cycle, called C cycle.
2) Create a set of new nodes, called C'. If all nodes in C do not have a bypass-edge, then for each node x in C create a new node x' for C'.

If some nodes in C have bypass-edgesm then only create new nodes corresponding to the nodes between the first node of C and the first node with bypass-edge. A new node is created associated with the node reached by the bypass-edge.

3) Create a bypass-edge from the first node of C to the second node of C' (to bypass the cycle just met).
4) Create branches connecting all conjunctive nodes in C'.
5) Move all nonessential out-branches (branches not constructing the most complicated path) of nodes in C to their associated nodes in C'. Move all in-branches to nodes in C to their associated nodes in C'.

If there is a nonessential branch connecting nodes in C, say (x,y), then remove the branch and create a new branch connecting their associated nodes in C'.

6) Traverse the first node in C'.

With the application of the cycle duplication method, a program graph will be transformed during the process of setting the accumulated complexity (ACom). After the ACom of the entry node is obtained, the most complicated loop-once path is found.

More detail algorithm for determining the most complicated loop-once paths and examples for these two algorithms can be found in [13].

5. Conclusion

Path complexity techniques and two test criteria are proposed. Also, the idea of the most complicated path is introduced. Some algorithms associated with the most complicated path are presented. Path complexity techniques can be applied in two areas: testing and metrics. This could lead to reduce software development cost and improve testing effencicy and software quality. Furthermore, the proposed algorithms could be implemented to become an useful tool in software environment.

Reference

1. [McCabe 82] McCabe, T. J., "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.

2.[Woodward 79], Woodward,M.R., Hennel,M.A,and Hedley,D.," A Measure of Control Flow Complexity in Program Text," IEEE Trans. on Software Eng., vol. SE-5, no.1,pp45-50,1979.

3.[Ntafos 84] Ntafos, S.C., "On Required Element Testing," IEEE Trans. on Software Eng., vol, SE-10, no. 6, pp. 795-803, Nov. 1984.

4.[Halstead 77] Halstead, M.H., Elements of Software Science, Elsevier, New York, 1977.

5.[McCabe 76] McCabe, T.J., "A Complexity Measure," IEEE Trans. on Software Eng., SE-2, no. 4, pp. 308-320. Dec.1976.

6.

7.[Rapps 82] Rapps, S., and Weyuker, E.J., "Data Flow Analysis Techniques for Test Data Selection," Pro. 6th Int'l Conf. Eng., Tokyo, Japan, pp. 272-282, Sept. 1982.

8. [Chung 88a] Chung, C.M., A Software Metrics Based Testing Environment, Ph.D. Dissertation, The advanced Center for Computer Studies, Univ. of Southwestern Louisiana, Lafayette, Louisiana, 1988.

9. [Chung 88b] Chung, C.M. and Yang, M.G., "Software Maintability," The 1988 Science, Engineering and Technology Semiars, May 28-29 Houston, Texas.

10. [Chung 88c] Chung, C.M., Edwards, W.R., and Yang, M.G., "A Software Metrics Based Software Environment," Proceeding of International Computer Symposium, Taipei, Taiwan, 1988.

11. [Chung 89a] Chung, C.M., Yang, M.G., and Hesih, G.T., "Testing- Specific Metrics," The 1989 Science, Engineering, and Technology Semiars, May 29-30 Houston, Texas.

12. [Chung 89b] Chung, C.M., Edwards, WR., and Yang, M.G., "Static and Dynamic Data Flow Metrics," Policy and Information journal, June 1989.

13. [Chung 89c] Chung, C.M., et. al., "Algorithms for Finding the Most Complicated Loop-free and Loop-once Paths,"Proceeding of National Computer Symposium, pp. 522-530
Dec. 1989, R.O.C.

14. [Miller 77] Miller, E. F. Jr., "Program Testing: Art Meets Theory," IEEE Computer, pp. 42-51, July 1977.

15. [Miller 84] Miller, E. F. Jr., "Software Testing Technology: An Overview," Handbook of Software Engineering, Van Nostrand Reinhold Company Inc., pp. 359-379. 1984.

16. [Laski 83] Laski, J.W. and Bogdan Korel, "A Data Flow Oriented Program Testing Strategy," IEEE Trans. Software Eng., vol. SE-9, no. 3, May 1983.

17. [Laski 85] Laski, J.M., "On Data Flow Guided Program Testing," Proceeding of the 8th International Conference on Software Eng., pp. 62-71, 1985.

18. [Dunsmore 79] Dunsmore, H. E., and Gannon, J.D., "Data Referencing: an Empirical Investigation," IEEE Computer, pp.50-59, Dec. 1983.

19. [Ntafos 79] Ntafos, S.C. and Hakimim S.L., "On Path Cover Problems in Digraphs and Applications to Program Testing," IEEE Trans. on Software Eng., vol. SE-5, pp. 520-529, Sept. 1979.

20. [Chen 78] Chen, Edward T., "Program Complexity and Programmer Productivity," IEEE Trans. on Software Eng., vol. SE-4, pp.187-194, 1978.

21. [Harrison 81] Harrison, W. A., and Magel, K. I., "A Complexity Measure Based on Nesting Level," ACM SIGPLAN Notices, vol. 16, no. 3, pp. 63-74, Mar. 1981.

22. [Weiser 85] Weiser, M.D., Gannon,J.D. and McMullin,P.R.,"Comparison of Structural Test Coverage Metrics," IEEE Software pp.80-84,1985.

23.[Weyuker 86] Weyuker, E.J., "Axiomatizing Software Test Data Adequacy," IEEE Trans. on Software Eng., vol. SE-12, no. 12, pp. 1128-1138. Dec. 1986.