STATIC ANALYSIS OF OBJECT ORIENTED SYSTEMS USING EXTENDED CONTROL FLOW GRAPH

Ananya Kanjilal[1], Swapan Bhattacharya[2]
[1]Dept. of Information Technology, B.P. P.I.M.T, Kolkata - 52, India, [1]ag_k@rediffmail.com
[2]Dept. of Computer Science&Engg, Jadavpur University,Kolkata-32, bswapan0000@yahoo.com

Abstract - The basic features of object oriented software makes it difficult to apply traditional testing methods like McCabe's Cyclomatic complexity in object oriented systems.The control flow graph used in procedural systems has been extended to be applicable for object oriented systems as ECFG. ECFG is a layered CFG where nodes refer to methods rather than statements. Our work is based on analyzing static characteristics of a system using ECFG. Extended cyclomatic complexity (E-CC) refers to the number of independent execution paths within the software. In this paper we have described different ways in which CFGs of individual methods are connected in an ECFG and E-CC for these different cases are given along with small code examples for substantiating our findings. Finally we have taken an ECFG of a fairly large system and applied these cases to arrive at the E-CC of the system considered as our example.

1. INTRODUCTION

Traditional testing methods have been successfully implemented in procedure oriented systems. However with the increased use of object orientation, the focus has shifted from a flat model towards a hierarchical layered model. Within object-oriented approaches, testing has almost been ignored in favor of analysis, design and coding [2]. The basic properties of object oriented software - encapsulation, inheritance and polymorphism make it impossible to apply traditional testing techniques to OO systems as it is.

One of the commonly used examples of white-box testing technique is “basis path testing” which ensures that every path of a program is executed at least once. McCabe’s cyclomatic complexity metric determines the number of linearly independent paths through a piece of software using the control flow graph (CFG) to determine a set of test cases which will cause executable statements to be executed at least once [1].

In this paper we study the complexity of object-oriented programs using Extended control Flow graphs (ECFG). The control flow graph used in procedural systems has been extended to be applicable for object oriented systems as ECFG [21]. ECFG is a collection of CFGs in a layered manner. It is also a directed graph but the nodes refer to methods rather than statements. The collection of CFGs may be connected in various ways depending upon the nature of method calls in the OO software.The proposed model would be helpful in identifying testing paths through an Object Oriented software thereby aiding in test selection.

2. RELATED WORK

A lot of research work in going on in the field of analysis of object oriented systems. Some focus on analysis during design and others during testing.

2.1. Analysis during design

0-7803-8560-8/04/$20.00@2004IEEE

Formal methods provide high level of assurance of correctness of a system and its conformance to specifications. However formal methods have failed to assure correctness of a large system with design complexity. Partial correctness checks for hierarchically designed systems has been one area of work. [3]. Test case selection from formal specifications has been another research direction. Stephane Barbey et al has proposed a test case selection methodology based on formal specifications aiming at verifying the correctness of method interaction of objects. [4]. Marie-Claude proposed a systematic way of using formal specifications for deriving test cases independent of the formal method used [5]. Algebraic specification is one of the popular approaches among formal methods and selection of test cases for determining observational equivalence has been the focus of research. [6] Due to the fundamental difference between OO systems and traditional systems, new metrics are needed to measure the unique aspects of OO systems. Some of the metrics deal with the design complexity, which definitely has an impact on testing and maintenance. Validation of three such design metrics and their ability to predict maintenance time has been the topic of research [7]

OMT (Object Modeling Technique) provides a reasonable approach to the development of object oriented systems but fails to integrate the object, dynamic and functional models which give three complementary views of the same system. A new model formalizing the dynamic model and integrating it with the object model is defined using a well known specification language LOTOS [8]. Design Complexity metrics like CK metrics is gradually gaining importance in industry. Research work has been done in analyzing CK metrics and proving its ability in determining software defects[9]

2.2. Analysis during Testing

Software is tested usually to achieve two goals- achieve quality by detecting and removing defects (*debug testing*) and assessing existing quality for measuring reliability (*operational testing*). The relationship between the two testing goals using a probabilistic analysis has been the focus of research work [10].Identification of issues and problems in testing object-oriented software has been the focus of research. [2],[11]

In the area of object oriented testing, Anita Goel et al has designed a probe based testing technique that observes the internal details of execution at different levels of abstraction and helps in selection of test cases for regression testing [12]. TATOO is a testing and analysis tool for object oriented software and provides a systematic approach to testing tailored towards object behavior and particularly for class integration testing [13]. A combination of use cases and cause effect graphing has lead to the development of a rigorous approach for

acceptance testing which ensures function coverage as well [14]. Evolutionary testing is an approach for automation of structural test case design. It searches test data that fulfill given structural test criteria by evolutionary computation[15].Bixin Li describes new techniques for analyzing information flow in OO systems. This technique based on object-oriented program slicing techniques computes the amount and width of information flow, correlation coefficient and coupling among basic components [16]. Data flow analysis is also an important aspect of testing as data is a major component of the programs.Dynamic data flow analysis in Java programs has been done to detect data flow anomalies [17]

Testing distributed object oriented systems is another domain of research. Jessica Chen in her paper has described a prototype of an automated test control toolkit for a system consisting of processes communicating through CORBA and have identified the problems of new deadlocks that might be introduced depending upon the thread model used and test constraint specified.[18].Concurrency introduces non-determinism and increases the complexity of testing. K. Saleh et al. have discussed the issue of synchronization anomalies in a concurrent Java program and proposed to detect them using probes [19]. A survey of some of the existing works related to object oriented testing reveals that not much work has been done in the enhancement of graphical system design techniques. TTCN-3 is the only standardized language for the specification and implementation of test cases. Two approaches for the graphical presentation and development of test cases has been discussed – GFT (graphical presentation format for TTCN-3) and the UML 2.0 Testing profile.

3. SCOPE OF WORK

Our work shares the same viewpoint with [20] in the sense that a well-defined graphical model, which would form the basis of testing of object-oriented systems is imperative. ECFG is a new model proposed as an extension to McCabe's Control Flow Graph [21]. In this paper we briefly discuss the features of ECFG and then describe the different cases of connectivity of methods in an ECFG. Deriving E-CC for each of these cases have been explained with small code examples. Finally we have taken an ECFG of a fairly large software and applied our rules for deriving E-CC for the same. Though our model can be equally well be applied for procedural systems with function calls, its applicability is more in OO systems where the ECFG would be revealing the connectivity of methods as dynamic binding at runtime.

4. ECFG : FEATURES

i) The graph is similar to CFG consisting of nodes and edges between nodes except at the top level where some nodes may be disconnected. It is a series of graphs arranged in layers.

ii) Nodes in CFG refer to a statement(s) whereas in ECFG nodes refer to methods.

iii)Every method has associated graphs(CFG) and cyclomatic complexity(CC) values. Methods, not found in the required class may be a part of its parent class.

iv)Object declaration is similar to variable declaration of procedural languages but is not a sequence statement (as in CFG) since it refers to constructor method.

v) Edges between nodes are formed whenever any methods calls another method. As discussed in next section, there may be different ways in which nodes are connected.

5. EXTENDED CYCLOMATIC COMPLEXITY

In an ECFG, the methods (or CFGs) may be connected in one of the six possible ways. E-CC, the composite complexity of the two or more graphs taken together is distinct in each case. The physical significance of CC i.e. the number of independent paths is maintained while calculating E-CC as well.

Case 1 : When two or more graphs are connected in series, i.e. the methods execute in sequence one after the other.

If V(G1),V(G2).....V(Gn) are complexities of individual graphs then the E-CC is the largest among them all.

i.e. E-CC = V(Gx) if V(Gx) > V(G1),V(G2)….. V(Gn) and 1< x < n

Proof : When the graphs are in series, the paths required to cover the highest complexity graph would automatically include the paths for graphs having lesser number of paths.

Example: Let M_1 & M_2 be two methods in series. Fig1, Fig2 & Fig3 refer to CFGs and ECFG.

```
m1(int a, int b)
{
1  if (a>b)                    →     V(M1) = 2
2     printf("A is greater");
3  }
```

```
                          m2(int a)
                          {
                          4  while(a != 10)
V(M2) = 2  ←              5  {printf("a=%d",a);
                             a++;}
                          6  }
```

Referring to Fig 3, E-CC = max(2,2) = 2

Basis set = { 1-2-3-4-5-4-6,1-3-4-6}

Case 2 : When two or more graphs are embedded within a graph i.e. a method calls another method which in turn calls another and so on.

If V(G1), V(G2) …… V(Gn) are complexities of each individual graphs and they are embedded within each other i.e. G1 embeds G2, G2 embeds G3 and so on, then the E-CC is given as follows :

E-CC = V(G1) + V(G2) + V(G3) + ……V(Gn) – (n-1) where n-1 graphs (2 to n) are embedded within G1.The same holds true even if there is no nested embedding i.e. G1 embeds G2,G3... Gn.

Proof : When the graphs are embedded, one of the paths is replaced by a graph. So resulting complexity i.e. independent paths is the sum total of complexities of all minus n (no. of paths that are replaced).

Example: Consider the same two methods M_1 & M_2 connected as M_2 embedding M_1.

The method M2 is slightly different with one statement having a call to M1 as given below.The individual CFGs are same as Fig1 & Fig 2. The ECFG is shown in Fig 4.

```
m2(int a)
{
4  while (a!= 10)
5  { m1(2,5);
     printf(" a = %d", a);     →  V(M1) = 2
     a++;}                     →  V(M2) = 2
6  }
```

M2 embeds M1 in such a way that the path 4-5 in M2 is

replaced by the paths in M1 as shown in Fig 4.

E-CC = 2 + 2 –1 =3.

Case 3 : If a graph embeds another graph more than once, e.g G1 embeds G2 thrice, then G2 is taken to be embedded only once since once tested it need not be tested again.

Composite complexity E-CC will be same as in Case 2 where all the complexity values refer to unique graphs - G1, G2 …….. Gn.

Case 4 : When many graphs embed the same graph i.e. more than one method call the same method. Here also the repeated graph is considered only once but the point of entry should be tested in every context.

E-CC = (V(G1)+V(G2)-1)+V(G3)+…...+V(Gn)+(n-2)

where V(G1) : complexity of G(1) that is embedded multiple times

and V(G2) …V(Gn) are complexity values for graphs which embed G1 and

n-2 : no. of graphs embedding G1 except one.

Proof : The embedded graph G1 needs to be tested only once and hence it is considered only once and composite complexity is calculated. However for all other graphs embedding G1 one path depicting entry point should be tested.

Example : Let us consider a method M3 connected as M3 called from M1 and M2.

```
   m1(int a, int b)                    m2(int a)
   {                                   {
1     if (a>b)                      4   while (a!= 10)
2      printf("A is greater");      5   { m3(2);
       m3(10);                          printf(" a = %d", a);
3  }                                    a++;}
                                        6    }

              m3(int a)
              {
           7    if (a = 10)
           8      printf("ok");
           9   }
```

The CFGs of M_1 and M_2 are shown in Fig 1 and Fig 2 earlier. The CFG of M3 and the ECFG is shown as Fig 5 and Fig 6 respectively.

$V(M_1) = 2$ $V(M_2) = 2$ $V(M_3) = 2$

As per ECFG, E-CC = $(CC(M_1)+CC(M_3)–1)+CC(M_2) + 1$ = (2 + 2 –1) + 2 + 1 = 6

M_1 and M_2 embeds M_3 in such a way that the path 1-3 in M_1 and 4-5 in M_2 is replaced by the paths in M_3 as in Fig 7.

Case 5 : When one graph is recursively repeated i.e. a method is recursively called.

The E-CC = V(G1) + 2, V(G1) is the complexity of the method in recursion.

Proof : The graph needs to be tested only once and the recursion represents a decision statement and hence represents two more paths - either method is recursively called again or it exits from recursive loop.

Example: Consider this code and its ECFG in Fig 8.

```
   recur (int i)
1  {
2      if ( i/2 == 0)
3       {     printf("ok");}
4         else
          {   printf("Not ok");
5         }
6       if (i !=0)
1         recur (--i);
          }
7  }              E-CC = V(recur) +2 = 2 + 2 = 4
```

Case 6: When recursion involves more than one method. This is a combination of case 1,2,3.

Composite complexity E-CC = V(G') + 2 where G' is more than one method connected as per case 1 or 2.

Proof : Composite complexity would be the complexity of the methods taken together (following Case 1 or Case 2) plus two for recursion as in case 3.

6. OBSERVATIONS- ECFG FOR CASE STUDY

We have considered a fairly large example in Java for our case study. The example consists of four classes and thirty nine methods.The ECFG is shown in Fig 9.We have applied the different cases of extended cyclomatic complexity as discussed in the previous section 5 to our example and arrived at the Extended Cyclomatic Complexity or E-CC of the system used in our case study.

Refer to Fig 9, certain regions in the graph have been identified which is simplified further and composite complexity of those branches has been calculated. Methods 1-28,31 &32 has V(G) values 1, methods 30 & 33 to 38 has value 2, method 29 and 39 has 10 & 15 respectively.

The regions are A,B,C,D,E,F,G,H,I and J. The methods within each region can be seen in Fig 9.

It is obvious from above that certain methods are embedded multiple times by many methods (called by many) e.g. 3,19,27,28. So, apart from the above we consider few more regions whose composite complexity would be calculated as per Case 4.

I : Method 27 is embedded by 33,34,35,37,38,39

J : Method (28 embeds A) by 33,34,35,37,38,39

K : Method 3 by 29,37,38,39

L : Method 19 by 36,39

As per different cases, E-CC is calculated and following values are obtained :

V(A) = 10, V(B) = 1, V(C) = 16, V(D) = 2, V(E) = 2, V(F) =2, V(G) =2, V(H) = 7, V(I) = 9, V(J) = 16, V(K) = 10, V(L) = 2

The ECFG consists of the subgraphs C, E, F, G, H, I, J, K and L after simplification. Each represent certain set of independent paths which should be considered while testing. The overall Extended Cyclomatic Complexity (E-CC) is therefore

E-CC = V(C) + V(E) + V(F) + V(G) + V(H) + V(I) + V(J) + V(K) + V(L)

= 16 + 2 + 2 + 2 +7 + 9 + 16 + 5 + 2 = 61

This suggests that there are 61 possible independent paths in the system, which should be considered while testing.

7. CONCLUSION

In this paper we propose a graph based methodology for analysis of OO systems focussing on the structure of program code and arrives at an analogous model to CFG for testing of OO systems. Our future course of work will be focused on consolidating this extended graph, defining its characteristics and the methodology for its construction. Cyclomatic complexity metric identifies the minimum number of paths required for testing. We propose to implement the Extended Cyclomatic complexity metric and derive test paths for OO systems similar to McCabe's basis set which would be essential for test vector generation. E-CC as a measure of testability and relating it to testing

effort and defect rates are some other directions which we intend to explore in our future research works.

REFERENCES

[1] C. Ghezzi, M. Jazayeri, D. Mandrioli "Fundamentals of software engineering", Prentice Hall, India, 1998.

[2] Stephane Barbey, Alfred Strohmeier, "The Problematics of Testing Object-Oriented Software", SQM'94 Second Conference on Software Quality Management, Edinburgh, Scotland, UK, July 26-28 1994.

[3] Murali Rangarajan, Perry Alexander and Nael Abu-Ghazaleh, "Using Automatable Proof Obligations for Component Based design Checking", IEEE Conference and Workshop on Engineering of Computer-Based systems, March 1999.

[4] Stephane Barbey, Didier Buchs and Cecile Peraire, "A Theory of Specification Based Testing for Object-Oriented Software", Proceedings of EDCC2 (European Dependable Computing Conference), Taormina, Italy, October 1996.

[5] Marie-Claude Gaudel, "Testing from Formal Specifications, a Generic Approach", online publication, Springer-Verlag, May 2001.

[6] Huo Yan Chen, T.H. Tse and Yue Tang Deng, "ROCS : an object-oriented class-level testing system based on the Relevant Observable Contexts technique", Information and Software Technology, Vol 42, Issue 10, pp 677-686, July 2000.

[7] Rajendra K. Bandi, Vijay K. Vaishnavi and Daniel E. Turk, "Predicting Maintenance Performance Using Object-Oriented Design Complexity Metrics", IEEE transactions on Software Engineering, Vol. 29 No. 1, January 2003.

[8] Betty H. C. Cheng and Enoch Y. Wang, "Formalizing and Integrating the Dynamic Model for Object-Oriented Modeling", IEEE transactions on Software Engineering, Vol. 28 No. 8, August 2002.

[9] Ramanath Subramanyam and M.S. Krishnan, "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity : Implications for Software Defects, IEEE transactions on Software Engineering, Vol. 29 No. 4, April 2003.

[10] Phyllis G. Frankl, Richard G. Hamlet, Bev Littlewood and Lorenzo Strigini, "Evaluating Testing Methods by Delivered Reliability", IEEE transactions on Software Engineering, Vol. 24, No. 8, August 1998.

[11] Stephane Barbey, Manuel Ammann , Alfred Strohmeier, "Open Issues in Testing Object-Oriented Software", ECSQ'94 (European Conference on Software Quality), Basel, Switzerland, October 17-20 1994.

[12] Anita Goel, S.C. Gupta and S.K. Wasan, "Probe Mechanism for Object-Oriented Software Testing", online publication, Springer-Verlag, February 2003.

[13] Amie L. Souter, Tiffany M. Wong, Stacey A. Shindo and Lori L. Pollock, "TATOO: Testing and Analysis Tool for Object-Oriented Software", online publication, Springer-Verlag, March 2001.

[14] Jose L. Fernandez, "Acceptance Testing of Object-Oriented Systems", online publication, Springer-Verlag, June 2002.

[15] Joachim Wegener, Andre Baresel and Harmen Sthamer, "Evolutionary test environment for automatic structural testing", Information and Software Technology, Vol 43, Issue 14, pp 841-854, December 2001.

[16] Bixin Li, "A technique to analyze information-flow in object oriented programs", Information and Software Technology, Vol 45, Issue 6, pp 841-854, April 2001.

[17] A.S. Boujarwah, K. Saeh and J. Al-Dalla, "Dynamic data flow analysis for Java programs", Information and Software Technology, Vol 42, Issue 11, pp 765-775, August 2003.

[18] Jessica Chen, "On using Static Analysis in Distributed System Testing", LNCS, Springer-Verlag, ISSN - 0302-9743,Vol 1999/2001, pp 145, January 2001.

[19] K. Saleh, Abdel Aziz Boujarwah and Jehad Al-Dallal, "Anomaly detection in concurrent Java programs using dynamic data flow analysis", Information and Software Technology, Vol 43, Issue 15,pp 973-981, December 2001.

[20] I. Schieferdecker and Jens Grabowski, "The Graphical Format of TTCN-3 in the context of MSC and UML", LNCS, Springer-Verlag, Vol 2599/2003, pp 233 - 252, January 2003.

[21] Ananya Kanjilal,Goutam Kanjilal,Swapan Bhattacharya, "Extended Control Flow Graph : An Engineering Aproach", CIT'03, Sixth International Conference on Information Technology, Bhubaneswar, India,December 22-25, 2003.

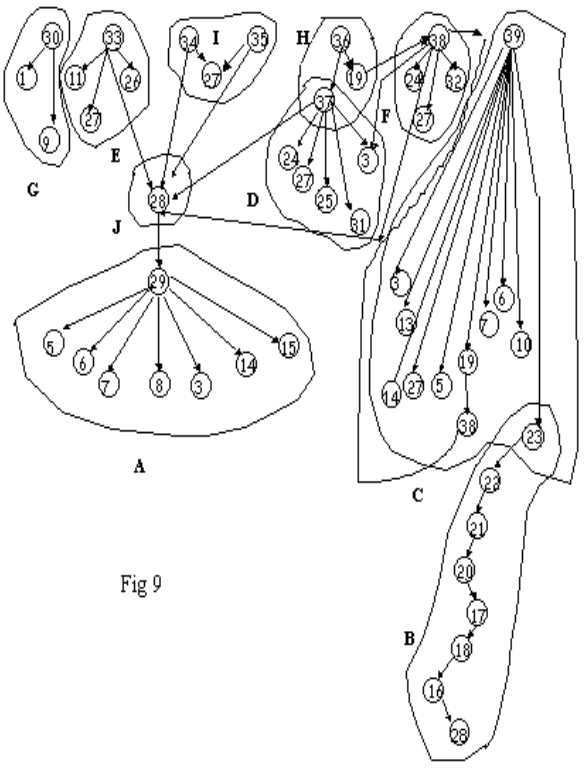

Fig 9

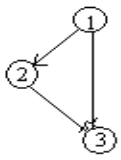

Fig 1

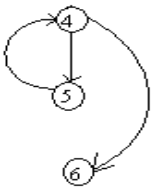

Fig 2

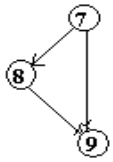

Fig 5

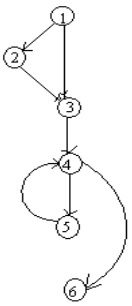

Fig 3

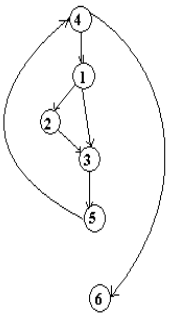

Fig 4

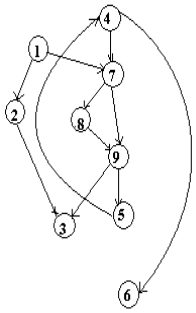

Fig 7

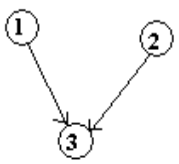

Fig 6

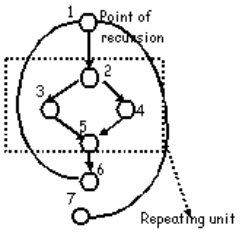

Fig 8