Volume 3, Issue 7, July 2013 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Improvised DU Pairs Algorithm for Unit Testing of Object Oriented Software

S.Suguna Mallika
Department of Computer Science and Engineering
CVR College of Engineering
Vastunagar, Mangalpally, R.R.District, India

Abstract- Testing is one of the most time consuming activities in the entire software development life cycle. In this paper, an effort has been made to propose an important modification to a testing strategy that has been proposed for unit testing of Object Oriented software. Unit testing of Object Oriented testing primarily boils down to testing of individual classes of the software. Testing of a class is analogous to the testing of methods defined in the class. In the current work the data flow based testing strategy employed to analyze the method which is of highest importance has been further revised and modified and applied to an appropriate case study to demonstrate the results. The current strategy which has been modified is more robust in its demonstration as to identifying the priority method. The current work demonstrates that the testing effort required by the tester is reduced.

Keywords- data flow testing, class testing, object oriented testing, unit testing

I. Introduction

The amount of time consumed by testing in the entire software development life cycle is around 60%. This not only shows the importance of testing in a software product life cycle, but also intrigues the test engineers at large to think of innovative ways to accomplish testing in a more robust manner. At different points of software development if the tester would like to check the health of the component quickly and thoroughly then it poses a major challenge to him. Every time a change is made to a component the tester would have to check the health of the component by performing unit testing. But unit test cases also are sometimes quite numerous and might consume quite enough time before they completed the execution of those test cases. With respect to Object oriented software, unit testing refers to an individual class or a group of classes rather. Testing an individual class boils down to testing a set of methods defined as part of the class. Instead of running all the test cases pertaining to all the methods, a strategy has been proposed to come up with the most important method out of the existing ones so that the tester could execute all the methods pertaining to the priority method and execute only random test cases pertaining to the other methods. This would help the tester in arriving at robust conclusions about the health of the component. It would save the tester with some time while still achieving the desired quality.

II. Proposed Strategy

The proposal is that all possible du-chaining of the data members defined in the class is done. Then a method which is the most repeated in the majority of the du chains is assigned a high priority factor and is subjected to logic coverage testing. The remaining sequences of methods are tested under normal conditions using state based testing techniques applying on the constraints of the object's state. For arriving at assigning priority of the methods, i.e to understand the most important methods of all, an algorithm called the DU pairs algorithm has been proposed. The algorithm***** in [3] is adapted to suit the needs of independent application classes and comprises of the following steps. The Step 3 has been modified specifically after observing the results over different case studies.

Step 1: def-use pairs of all the data members defined in the class are made. [3].

Note: A def-use pair corresponds to the listing of a pair of line numbers where the first number indicates the line of occurrence of the definition of the variable and the second number indicates the line number where usage of the variable is seen without the variable getting killed in between.

Step 2: The pairs of line numbers are mapped to method names in which the line numbers are occurring thus arriving with the set of method names in which the definition of a data member is occurring and the method name in which it is being used subsequently.[3].

Step 3: Now an individual counter is maintained against each method (excluding the constructor method) and du pairs of methods is observed.

Note: *If the constructor method were to be considered, then it has been observed that most of the case studies resulted with the constructor function as the most priority method on which the algorithm has been run. The constructor methods are being excluded because this method is explicitly meant for defining the data members and not use them especially which could be tested by using a couple of test cases. Having considered the fact that constructor functions are for defining the data members and hence for every data member's definition this method would certainly get listed thereby increasing the count of the constructor method.*

Step 4: With the occurrence of each method in the listing, the counter against that corresponding method is incremented. [3]

Step 5: Step 4 is repeated until all the methods listed in the du pairs are exhausted. [3].

Step 6: Now for each of the methods a table is prepared with the final counter value against it. [3].

Step 7: The table is sorted based on the counter value for each method. [3].

Step 8: Starting with the maximum counter value priority is assigned in the increasing order. I.e method with the maximum counter value is assigned a priority of 1 and so on. [3].

Step 9: If more than one method arrive at the same counter value same priority is assigned to both the methods. [3].

Step 10: After assigning the priority the method with the highest priority is subjected to full logic coverage testing while the other methods are subjected to equivalence partitioning and boundary value analysis testing. [3].

III. Case Study

The above algorithm has been manually traced on a sample case study i.e a matrix class under consideration. Code for the matrix Class which has methods like add, multiply, transpose, equals, toString written is as follows:

```
1.  import java.util.Scanner;
2.  public class Matrix {
3.  private int[][] matrixElements;
4.  private int rows;
5.  private int cols;
6.  public Matrix() {
7.  rows = 2;
8.  cols = 2;
9.  matrixElements = new int[2][2];
10. }
11. public Matrix(int rows, int cols) {
12. this.rows = rows;
13. this.cols = cols;
14. this.matrixElements = new int[rows][cols];
15. }
16. public boolean equals(Matrix matrix){
17. boolean equalFlag = true;
18. if(this.rows != matrix.rows && this.cols != matrix.cols){
19. return false;
20.
21. }
22. for(int rowIndex = 0;rowIndex<matrix.rows; ++rowIndex){
23. for(int colIndex =0;colIndex < matrix.cols;++colIndex){
24. if(this.matrixElements[rowIndex][colIndex] != matrix.matrixElements[rowIndex][colIndex]){
25. equalFlag = false;
26. return false;
27. }
28. }
29. }
30. return equalFlag;
31. }
32. public Matrix(int rows, int cols, int[][] matrix) {
33. this.rows = rows;
34. this.cols = cols;
35. this.matrixElements          = new int[rows][cols];
36. for(int rowIndex=0; rowIndex <rows; ++rowIndex)
37. for(int colIndex=0; colIndex <cols; ++colIndex)
38. this.matrixElements[rowIndex][colIndex]     = matrix[rowIndex][colIndex];
```



```
39.
40. }
41. public Matrix add(Matrix matrix1, Matrix matrix2) {
42. Matrix matrix3;
43. matrix3 = new Matrix(matrix1.rows, matrix2.cols);
44. for(int row=0; row < matrix1.rows; ++row)
45. for(int col=0; col < matrix1.cols; ++col)
46. matrix3.matrixElements[row][col] += matrix1.matrixElements[row][col] +
47. matrix2.matrixElements[row][col];
48. System.out.println(matrix3);
49. return matrix3;
50. }
51. public Matrix multiply(Matrix matrix1, Matrix matrix2) {
52. if(matrix1.cols != matrix2.rows) {
53. System.out.println("\nMatrix Multiplication Not possible\n");
54. System.exit(1);
55.
56. }
57. Matrix matrix3;
58. matrix3  = new Matrix(matrix1.rows, matrix2.cols);
59. for(int rowIndex=0; rowIndex < matrix1.rows; ++rowIndex)
60. for(int colIndex = 0; colIndex < matrix2.cols; ++colIndex)
61. for(int tempIndex=0; tempIndex < matrix1.cols; ++tempIndex)
62. matrix3.matrixElements[rowIndex][colIndex] += matrix1.matrixElements[rowIndex][tempIndex]
63. matrix2.matrixElements[tempIndex][colIndex];
64. return matrix3;
65. }
66. public Matrix transpose(Matrix matrix) {
67. Matrix transposedMatrix = new Matrix(matrix.cols, matrix.rows);
68. for(int rowIndex=0; rowIndex < matrix.rows; ++rowIndex)
69. for(int colIndex = 0; colIndex < matrix.cols; ++colIndex)
70. transposedMatrix.matrixElements[colIndex][rowIndex] = matrix.matrixElements[rowIndex][colIndex];
71. return transposedMatrix;
72. }
73. public String toString() {
74. String matrixResultString = "\n";
75. for(int row = 0; row < rows; ++row) {
76. for(int col = 0; col < cols; ++col)
77. matrixResultString += matrixElements[row][col] + "  ";
78. matrixResultString += "\n\n";
79. }
80. return matrixResultString;
81. }
82. }
```

Computation of the DU Pairs for each data member defined in the class:

The first number specifies the line of occurrence of the definition of the data member and the second number signifies the usage of the data member. The listing of methods is given subsequently with the method name in which the definition of the data member occurred followed by the method name where the data member has been used without the definition getting killed.

For data member "matrixElements" :

(9, 24) – (Matrix, equals), (9, 46) – (Matrix, add)
(9, 62) – (Matrix, multiply), (9, 70) – (Matrix, transpose)
(9, 77) – (Matrix, toString), (14, 24) – (Matrix, equals)
(14, 46) – (Matrix, add), (14, 62) – (Matrix, multiply)
(14, 70) – (Matrix, transpose), (14, 77) – (Matrix, toString)
(38, 24) – (Matrix, equals), (38, 46) – (Matrix, add)
(38, 62) – (Matrix, multiply), (38, 70) – (Matrix, transpose)

(38, 77) – (Matrix, toString), (46, 46) – (add, add)
(62, 62) – (multiply, multiply), (70, 70) – (transpose, transpose)

For data member "rows":

(7, 18) – (Matrix, equals), (7, 22) – (Matrix, equals)
(7, 43) – (Matrix, add), (7, 44) – (Matrix, add)
(7, 52) – (Matrix, multiply), (7, 58) – (Matrix, multiply)
(7, 59) – (Matrix, multiply), (7, 67) – (Matrix, transpose)
(7, 68) – (Matrix, transpose), (7, 75) – (Matrix, toString)
(12, 14) – (Matrix, Matrix), (12, 18) – (Matrix, equals)
(12, 22) – (Matrix, equals), (12, 43) – (Matrix, add)
(12, 44) – (Matrix, add), (12, 52) – (Matrix, multiply)
(12, 58) – (Matrix, multiply), (12, 59) – (Matrix, multiply)
(12, 67) – (Matrix, transpose), (12, 68) – (Matrix, transpose)
(12, 75) – (Matrix, toString), (33, 18) – (Matrix, equals)
(33, 22) – (Matrix, equals), (33, 43) – (Matrix, add)
(33, 44) – (Matrix, add), (33, 52) – (Matrix, multiply)
(33, 58) – (Matrix, multiply), (33, 59) – (Matrix, multiply)
(33, 67) – (Matrix, transpose), (33, 68) – (Matrix, transpose)
(33, 75) – (Matrix, toString)

For data member cols:
(8, 18) – (Matrix, equals), (8, 24) – (Matrix, equals)
(8, 43) – (Matrix, add), (8, 45) – (Matrix, add)
(8, 52) – (Matrix, multiply), (8, 58) – (Matrix, multiply)
(8, 60) – (Matrix, multiply), (8, 67) – (Matrix, transpose)
(8, 69) – (Matrix, transpose), (8, 76) – (Matrix, toString)
(13, 18) – (Matrix, equals), (13, 24) – (Matrix, equals)
(13, 43) – (Matrix, add), (13, 45) – (Matrix, add)
(13, 52) – (Matrix, multiply), (13, 58) – (Matrix, multiply)
(13, 60) – (Matrix, multiply), (13, 67) – (Matrix, transpose)
(13, 69) – (Matrix, transpose), (13, 76) – (Matrix, toString)
(35, 18) – (Matrix, equals), (35, 24) – (Matrix, equals)
(35, 43) – (Matrix, add), (35, 45) – (Matrix, add)
(35, 52) – (Matrix, multiply), (35, 58) – (Matrix, multiply)
(35, 60) – (Matrix, multiply), (35, 67) – (Matrix, transpose)
(35, 69) – (Matrix, transpose), (35, 76) – (Matrix, toString)

IV. Result Analysis

Table I
Priority Computation Table

S.No.	Method Name	Count	Priority
1	Equals	15	3
2	add	18	2
3	Multiply	24	1
4	Transpose	18	2
5	toString	9	4

After the priority computation for each of the methods, as per the strategy the method multiply () has to undergo full logic coverage testing. And the remaining methods would be subjected to Equivalence Partitioning or Boundary value Analysis testing techniques.

V. Conclusions

By following this strategy, the following drawbacks are overcome which are present with the existing strategies:

- This strategy helps in saving testing time while delivering quality software.
- Computational complexity of techniques like symbolic execution and automated deduction.

VI. Limitations and Future Work

- The strategy needs to be further strengthened to suit the needs of collaborating classes, especially if the class has methods which take no parameters at all.
- An automated tool is to be developed to implement the algorithm which when fed with the class file would display the priority methods for the tester.

References

[1] Roger S. Pressman, Software Engineering a Practitioner's Approach, Seventh Edition, McGraw-Hill Int'l Edition. Boris Beizer, Software Testing Techniques, Second Edition, International Thomson Computer Press, 1990.
[2] S. Suguna Mallika, J. Vamsi Vijay Krishna, P.Amulyasri, "A Data Flow Based Novel Testing Strategy for Unit Testing of Object Oriented Software", International Journal of Science and Engineering Applications(IJSEA) Digital Library, pages 75-78, Volume 1 Issue 1, November 2012 Edition
[3] Harry M. Sneed, "Testing Object Oriented Software Systems", Proceeding ETOOS '10 Proceedings of the 1st Workshop on Testing Object-Oriented Systems ACM New York, NY, USA
[4] Ugo Buy, Alessandro Orso and Mauro Pezze, "Automated Testing of Classes" ISSTA '00, ACM, Portland, Oregon.
[5] Ilinca Ciupa, Andreas Leitner, Manuel Oriol, and Bertrand Meyer, "ARTOO: Adaptive Random Testing for ObjectOriented Software", ICSE'08, May 10–18, 2008, Leipzig, Germany, ACM 9781605580791/08/05.
[6] Lucas and Serpa Silva, "Evolutionary Testing of Object-Oriented Software" ACM SAC'10, March 22-26, 2010, Sierre, Switzerland, 978-1-60558-638-0/10/03.
[7] HUO YAN CHEN, T. H. TSE and T. Y. CHEN,"TACCLE: A Methodology for Object-Oriented Software Testing at the Class and Cluster Levels" ACM Transactions on Software Engineering and Methodology, Vol. 10, No. 4, January 2001, Pages 56–109.
[8] S.R.Chidamber and C.F.Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, Vol. 20, No. 6, 1994, pp. 476-493.
[9] Mauro Pezze and Michal Young, "Testing Object Oriented Software", IEEE Proceedings of the 26th international conference on Software Engineering (ICSE' 04).
[10] Vincenzo Martena, Alessandro Orso, Mauro Pezz`, "Interclass Testing of Object Oriented Software" Proceedings of the Eighth IEEE international Conference on Engineering of Complex Computer Systems (ICECCS'02)
[11] Xiajiong Shen and Qian Wang, PeipeiWang and Bo Zhou, "A Novel Technique Proposed for Testing of Object Oriented Software Systems"
[12] Amie L. Souter, Lori L. Pollock, "OMEN: A Strategy for Testing Object-Oriented Software", ISSTA '00, Portland, Oregon, ACM 1-58113-266-2/00/0008
[13] Pei Hsia, Xiaolin Li, David C. Kung, "Class Testing and Code-Based Criteria"
[14] HUO YAN CHEN, T. H. TSE and F. T. CHAN, T. Y. CHEN, "In Black and White: An Integrated Approach to Class-Level Testing of Object-Oriented Programs", ACM Transactions on Software Engineering and Methodology, Vol. 7, No. 3, July 1998, Pages 250–295.

