Analyzing Slicing of Program through Cohesion Metric

Kumar Rajnish
Department of CSE
Birla Institute of Technology, Mesra
Ranchi, India
krajnish@bitmesra.ac.in

Mayank Manohar
Department of CSE
Birla Institute of Technology, Mesra
Ranchi, India
mayankmanohar8@gmail.com

Abstract

This paper presents a new cohesion metric for a program which is used to analyze the slicing criterion. Based on the value of propose cohesion metric it is decided to perform slicing in a program or not. The propose cohesion metrics is also evaluated analytically against Weyuker's Property and perform comparison with the existing cohesion metrics of Meyers et al. The results in this paper shows that the propose cohesion metric is a good method for analyzing the slicing of a program or not. Data for 16 'C' programs has been collected from open source software system. For analyzing the results IBM SPSS software were used.

Keywords- Cohesion; Metric; Program Slicing; Weyuker's Property; Program; Module;

I. INTRODUCTION

Program slicing is a method of program analysis which is used to extract a set of statements in a program which is relevant for a particular computation. The idea of program slicing was given by Mark Weiser [1]. Program slicing can be used in various software engineering activities such as program understanding, program maintenance, debugging, testing, complexity measurement etc. Program slicing is a method of obtaining subparts of a program with a collective meaning. A program slice consists of the parts or components of a program that affect the values computed at some point of interest, referred to as a slicing criterion. A program slice consists of a pair <s, V> where s represents the statement number of the program and V represents the subset of variables. With the help of program slicing one can find a smaller program which still maintenance the original aspect of the program. A program slicing can be used to measure the cohesion level of software units. However, limited work has been done in quantitative cohesion metrics. The most popular work on qualitative cohesion metrics has been done by Bieman and Ott. et al [2,14]. Their cohesion metrics are based on program slicing. A module consists of collection of processing elements working together to build outputs of the modules given in [14]. A module can be categorized into low cohesion and high cohesion in which a modules with low cohesion consists of two or more independent processing elements whereas a modules with high cohesion consists of collection of highly related processing elements.

Riazur Raheman et al [3] address different types of program slicing techniques by considering a very simple example. Program slice is computed by analyzing dependence relations between program statements. To compute program slices they constructed intermediate structures of a program such as program dependence graph. Also, they address the comparison between different types of slices. Meyers et al [4] suggests that slice-based cohesion metrics quantify overall code quality. Effects of software evolution on slice based metrics is measured. Gives base-line values for the slice-based cohesion metrics: Tightness, Min Coverage, Coverage, Max Coverage, and Overlap. Base-line values are useful in the identification of degraded modules. Finally, they compares the different metrics "head-to-head." Thus, providing a better understanding of their relationships and indicating which metrics provide a similar view of a program and which provide complimentary views of the program.

Fumiaki OHATA et al. [5] implement a slicing method for Object Oriented programs. Their slicing method is an intermediate method between static slicing and dynamic slicing. Their proposed method dynamically analyze all the data dependence and control dependence relations about method invocations and their analyze precision is better as compared to static slicing. Also their analysis costs is less than that of dynamic slicing. Timothy M Meyers et al. [6] suggests large scale empirical investigation of slice-based cohesion metrics. They provides a head to head metrics comparison. Their metrics have the capacity to be used at the program level to guide the effects of reverse engineering's attempt to "improve" code . Finally their metrics provide good estimates of expected metric values. Their values can be used at the module level to focus the attention of

reverse engineers on particularly object modules. Mehmet Kaya et al. [7] improve the structure of an existing class without changing its external behavior. They presents a new cohesion metric based on program slicing and graph theory for units using object oriented paradigm. Their aim to find out if a class is cohesive, handling one specific operation. When a class has more than one abstraction, this technique suggests a restructuring for generating more cohesive units based on this new cohesion metric. Sonam Jain et al. [8] implement a mixed slicing approach of static and dynamic slicing i.e. S-D slicing approach in generating a program slice. Specifically, in this study they develop a code that have an Object Oriented approach by using both static and dynamic slicing. Durga Prasad Mohapatra et al. [9] survey the existing slicing techniques for object-oriented programs. Many commercial object oriented programs are concurrent in nature.

Concurrency is typically implemented in the form of multithreading or message passing using socket or both. They review the available techniques in slicing of concurrent object-oriented programs. Another trend that is clearly visible in object-oriented programming is client-server programming in a distributed environment. Andrea De Lucia et al. [10] implements a conditioned slicing approach as a general framework for program comprehension that addresses all slicing paradigms. A program comprehension can be used to implement complex functionalities. They provide a conditioned slicing as a general framework for program comprehension. Heung Seok Chae et al. [11] implements a cohesion measurement tool (HYSS) for C++ programs, to automate the computation of the various cohesion measures including cohesion based on member connectivity (CBMC). Using HYSS they performed a case study with the Interviews system in order to demonstrate the effectiveness of CBMC. Their result showed that CBMC captured a new aspect of properties of classes that was not captured in the existing cohesion measures. Norihiro Yoshida et al. [12] provides support in the comprehension of functions, and proposed a technique to extract sets of code fragments which realize the same features within a function by making use of cohesion metrics. David Bowes et al. [13] focuses on the difficulties associated with what they anticipated would be a small and simple replication study. Their focus particularly on the problems related to specifying precisely and implementing consistently the definition of metrics being used to collect data.

The aim of the work presented here is to propose new cohesion metric for a program which used to analyze the slicing criterion. Based on the value of propose cohesion metric it is decided to perform slicing in a program or not. To increase the usefulness of the propose cohesion metric it also compare with the existing cohesion metrics which discussed in the Section III and analytical evaluation against Weyuker's property discussed in Section IV.

The rest of the paper is organized as follows: Section II deals with the Weyuker's properties. Section III deals with the existing cohesion metric which is used in our study. Section IV deals with the propose cohesion metric along with the examples illustrations and its analytical evaluation against Weyuker's properties. Section V deals the results and discussion. Section VI deals with the conclusion and future scope respectively.

II. WEYUKER'S PROPERTY

The Weyuker properties [15] are listed in Table 1. The notations used are as follows: P, Q and R denote programs, P+Q denotes combination of program P and Q; M denotes the chosen metrics, M (P) denotes the value of the metric for program P, and P≡Q (P is equivalent to Q) means that two program designs, P and Q, provide the same functionality. The definition of combination of two programs is taken here to be the same as suggested by [16], i.e., the combination of two programs results in another program whose properties (methods and instance variables) are the union of the properties of the component programs. Also, "combination" stands for Weyuker's notion of "concatenation".

TABLE I. WEYUKER'S PROPERTY

Property No	Property Name	Description
1	Non-coarseness	Given a program P and a metric M, another program Q can always be found such that, M (P) ≠ M (Q).
2	Granularity	There is a finite number of programs having the same metric value. This property will be met by any metric measured at the program level.
3	Non-uniqueness (notion of equivalence)	There can exist distinct program P and Q such that, M (P) = M (Q).
4	Design details are important	For two program designs, P and Q, which provide the same functionality, it does not imply that the metric values for P and Q will be the same.
5	Monotonicity	For all programs P and Q the following must hold: M (P) ≤ M (P + Q) and M (Q) ≤ M (P + Q) where P + Q implies combination of P and Q.
6	Non-equivalence of interaction	∃ P, ∃ Q, ∃ R such that M (P) = M (Q) does not imply that M (P+R) = M (Q+R).
7	Interaction among statements	Permutation of program statements can change the metric value.
8	No change on renaming	If P is renaming of Q then M (P) = M (Q)
9	Interaction increases complexity	∃ P and ∃ Q such that: M (P) + M (Q) < M (P + Q).

III. EXISTING COHESION METRIC

Meyers et al [4] provides five different sliced-based cohesion metrics which is used in this study. These metrics are listed below:

Tightness (M) measures the number of statements included in every slice. Higher value of Tightness indicates a higher degree of functional cohesion within the module.

$$\text{Tightness (M)} = \frac{|SL_{\text{int}}|}{length(M)}$$

Min Coverage (M) is the ratio of smallest slice in a module to the module's length. Higher value of Min Coverage indicates that the smallest slice in the module requires most of the statements in the module whereas all the other slices include a greater number of statements.

$$\text{Min Coverage (M)} = \frac{\min |SL_i|}{length(M)}$$

Coverage (M) compares the length of slices to the length of the entire module. Lower the value of Coverage indicates several distinct processing elements and also causes low cohesion.

$$\text{Coverage (M)} = \frac{1}{|V_o|}\sum_{i=1}^{Vo}\frac{|SL_i|}{length(M)}$$

Max Coverage (M) is the ratio of largest slice in a module to the module's length. Higher the value of Max Coverage indicates that longest slice in a module requires most of the statements in the module.

$$\text{Max Coverage (M)} = \frac{\max |SL_i|}{length(M)}$$

Overlap (M) is the measure of average number of overlapping statements in all the slices. A high value of Overlap may indicate high code interdependence since most of the statements belong to the most of the slices.

$$\text{Overlap (M)} = \frac{1}{|V_o|}\sum_{i=1}^{Vo}\frac{|SL_{\text{int}}|}{|SL_i|}$$

IV. PROPOSE SLICING COHESION METRIC

Under this section we present the definition of propose cohesion metric, examples for illustration, analytical evaluation of propose cohesion metric against Weyuker's properties and interpretation based on the result obtained.

A. Definion

This section presents the propose cohesion metric which is used for predicting the slicing requirement of a program. The propose cohesion metric is defined as follows:

$$\text{Direct Cohesion Metric (DCM)} = \left\{\sum_{i=1}^{n} S_i \cap V_{in} + \sum_{i=1}^{n} S_i \cap V_{out}\right\} / N$$

Where,

S =Statement of the program.

V_{in}=Input variable of the program.

V_{out}=Output variable of the program.

N=Total number of statements of the program.

Lack of Cohesion Metric (LCM) = 1 – DCM [0,1].

Based on the value of DCM and LCM it will decide whether slicing is required in a program or it is difficult to perform slicing in a program.

Certain important criteria have been design for program slicing from the relationship between DCM and LCM which are mentioned below:

Criteria 1: if DCM > 0.5 then

Program is cohesive in nature as LCM is lies between 0 to 0.5. There is less chances to perform slicing of program as properties (input and output data) of program are related to each other within the program.

Criteria 2: if DCM < 0.5 then

Program is less cohesive in nature and LCM is lies between 0.5 to 1. There is more chances to perform slicing of a program as properties of program are less related to each other within the program,.

B. Examples for illustartion

/// Program 1: This program is used to calculate the sum and product of i^{th} number

```
S1    void main()
S2    {
S3      int i;
S4      int sum = 0;
S5      int product = 1;
S6      for(i = 0; i < N; ++i)
S7      {
S8          sum = sum + i;
S9          product = product *i;
S10       }
S11         cout<< sum;
S12         cout<< product;}
```

From the above program values for DCM and ICM is as follows:

DCM = 0 and LCM = 1 – DCM = 1 -0 = 1

From the Criteria discussed above program 1 needs to be sliced and it can be divided in two parts (Say program P2 and P3) counts the given statement numbers:
For P2: S1, S3, S4, S6, S8 and S11.
For P3: S1, S3, S5, S6, S9 and S12.

/// Program 2 : This program is used to calculate the average and percentage of five different subjects

```
S1    void main()
S2    {
S3     int m1,m2,m3,m4,m5,total;
S4     float average, percentage;
S5     printf("Enter marks for subject one - ");
S6     scanf("%d",&m1);
S7     printf("Enter marks for subject two - ");
S8     scanf("%d",&m2);
S9     printf("Enter marks for subject three - ");
S10    scanf("%d",&m3);
S11    printf("Enter marks for subject four - ");
S12    scanf("%d",&m4);
S13    printf("Enter marks for subject five - ");
S14    scanf("%d",&m5);
S15    total=m1+m2+m3+m4+m5;
S16    average=total/5;
S17    percentage=(average/100)*100%;
S18    printf("\nThe average of five subjects
is%f",average);
S19    printf("\nPercentage=%f",percentage);
S20    getch();}
```

From the above program values for DCM and LCM is illustrated below:

DCM=0.6 and LCM=1-DCM=1-0.6=0.4

From the Criteria discussed above program 2 is difficult to slice as the value of DCM is greater than 0.5 and LCM is 0.4 which indicates program 2 little more cohesive.

C. Analytical Evaluation of Propose Metric against Weyuker's Properties

Certain assumptions have been defined below for performing analytical evaluation of DCM against Weyuker's Properties:

1. Two input variables of the same name used in two different programs / modules is considered to be same after combining two programs in a single program.
2. Two output variables of the same name in two different programs / modules is considered to be distinct variable and after merging one output variable will be assigned to another variable to resolve the conflict.
3. When two programs / modules are combined number of statements of combined program will increases and statements of a program will be placed according to the rules of the programming language.

Let Z_p = DCM and LCM for Program P

Let Z_Q = DCM and LCM for Program Q

Which are the function of a number of inputs, number of outputs and number of statements of a program which are independent and identically distributed.

Property 1 and *Property 3* are satisfied it is because of the assumption of statistically distribution of inputs and outputs of variables. It means there is a situation where M (P) = M (Q) and M (P) ≠ M (Q).

Property 2 is satisfied because there is a finite number of programs having the same metric value and this property will be met by any metric measured at the program/module level.

Property 4 is satisfied because the choice of input variables, output variables and the number of statements of programs are design implementation dependent.

From assumptions mentioned above considering the program 1 of Section B (Program of Sum and Product of first n numbers say P+Q). Program 1 may be divided into two programs say P (sum of first n numbers) and Q (product of first n numbers) program P has contained 9 statements and program Q has also contained 9 statements. After merging program P and program Q whose number of statements is 12. It is clearly obvious that for all programs P and Q the conditions: M (P) ≤ M (P + Q) and M (Q) ≤ M (P + Q) holds, where P + Q

implies combination of P and Q. Hence, property 5 is satisfied.

Considering the three programs say P, Q and R. Program P and program Q may have the same number of statements with varying input and output variables. Suppose program R have common input or output variables with varying in number of statements. Typically at some situation where M(P)= M(Q) but it does not mean that M(P+R)=M(Q+R). Hence, *property 6* is satisfied.

Property 7 requires the permutation of program statements can alter the metric value. It is because changing the order of statements in IF-THEN-ELSE block can change the logic of the program. Hence property 7 is satisfied. Only this property can applicable in traditional program.

Property 8 is also satisfied it is because changing of name of program can never affect upon the programs metric value.

Property 9 (Interaction Increases Complexity) is not satisfied.

Consider any two programs A and B with N_A and N_B number of statements for program A and program B respectively, the following association holds:

DCM(A) = N_A and DCM (B) = N_B

DCM(A + B) = $N_A + N_B - \alpha$.

Where, α is the number of common statements. Therefore, DCM (A + B) $\leq$ DCM (A) + DCM (B)

Table II. Analytical Evaluation Results against Weyuker's Property [√: Metric Satisfies the properties ×: Metric which does not satisfy the Properties]

S.NO	PROPERTY NAME	DCM
1.	Non-coarseness	√
2.	Granularity	√
3.	Non-uniqueness (notion of equivalence)	√
4.	Design details are important	√
5.	Monotonicity	√
6.	Non-equivalence of interaction	√
7.	Interaction among statements	√
8.	No change on renaming	√
9.	Interaction increases complexity	×

V. RESULTS AND DISCUSSION

In this section, we analyze the results of the used program with several calculated parameters like tightness, min coverage, coverage, max coverage, overlap, DCM, LCM. These programs are collected from the open source software system. The result is provided in Tables III and figure 1 shows the corresponding graph of Table III. The Correlation Coefficient of existing and propose cohesion metric are shown in Table V.

In this paper, an attempt has been made to design an cohesion metric with the help of some program collected from open source software system. We calculated many existing metric and these metrics are compared with the help of proposed metric. To validate the proposed metric the correlation has been calculated with the existing metrics used in this work, and it is found that the correlation is in acceptable level. The correlation between tightness and min coverage is 0.865, tightness and coverage is 0.924, dcm and tightness is 0.854 and the rest of the results are given in table V. In addition to it, Weyuker's property are also used to validate the proposed metric, eight out of nine properties are satisfied by the proposed measure. According to the above evaluated results it can be concluded that the proposed measure qualify as a worthy measure as seen in the correlation outcome between two measures (provided in Table V) and Weyuker's property described in section II.

Table III. Metric Values

Program	Tightness	Mincoverage	Coverage	Maxcoverage	Overlap	DCM	LCM	LOC
P1	0.20	0.50	0.50	0.50	0.40	0	1	9
P2	0.56	0.62	0.73	0.84	0.78	0.56	0.44	18
P3	0.30	0.63	0.63	0.63	0.47	0.15	0.85	33
P4	0.41	0.67	0.67	0.67	0.62	0.18	0.82	12
P5	0.39	0.62	0.67	0.77	0.58	0.16	0.84	11
P6	0.58	0.64	0.76	0.85	0.79	0.40	0.60	19
P7	0.67	0.74	0.77	0.80	0.87	0.45	0.55	14
P8	0.50	0.64	0.65	0.69	0.77	0.26	0.74	24
P9	0.58	0.72	0.75	0.79	0.77	0.30	0.70	14
P10	0.50	0.63	0.75	0.86	0.66	0.20	0.80	25
P11	0.72	0.83	0.83	0.83	0.86	0.60	0.40	13
P12	0.25	0.45	0.48	0.55	0.52	0	1	21
P13	0.40	0.60	0.60	0.60	0.67	0	1	9
P14	0.67	0.75	0.75	0.75	0.89	0.30	0.70	11
P15	0.28	0.42	0.48	0.50	0.61	0	1	11
P16	0.42	0.65	0.65	0.65	0.63	0	1	14

Table IV. Summary Statistics for DATA SET of Table III

	Minimum	Maximum	Mean	Std. Deviation
Tightness	.20	.72	.4644	.15912
Min Coverage	.42	.83	.6319	.10685
Coverage	.48	.83	.6669	.10793
Max Coverage	.50	.86	.7050	.12399
Overlap	.40	.89	.6806	.14617
DCM	.00	.60	.2225	.20201
LCM	.40	1.00	.7775	.20201
LOC	9.00	33.00	16.1875	5.78756

Table V. Correlation Coefficient of Metrics

Metric	Tightness	Min Coverage	Coverage	Max Coverage	Overlap	DCM	LCM
Tightness	-	0.865	0.924	0.841	0.953	0.854	-0.854
Min Coverage	0.865	-	0.912	0.734	0.721	0.717	-0.717
Coverage	0.924	0.912	-	0.938	0.784	0.839	-0.839
Max Coverage	0.841	0.734	0.938	-	0.714	0.825	-0.825
Overlap	0.953	0.721	0.784	0.714	-	0.773	-0.773
DCM	0.854	0.717	0.839	0.825	0.773	-	-1
LCM	-0.825	-0.717	-0.839	-0.825	-0.773	-1	-

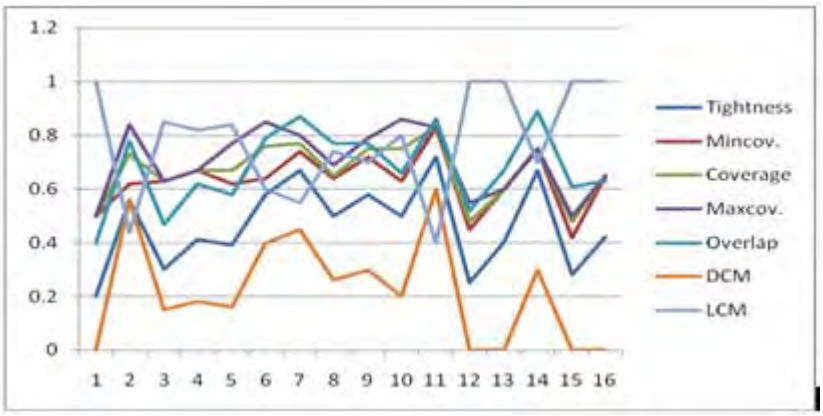

Figure 1: Metric values for 16 Programs

VI. CONCLUSION AND FUTURE WORK

In this paper, an attempt has been made to propose a cohesion metric and based on the value of that cohesion metric it can be find out that whether slicing is required in a program or not. If the value is below 0.5 then the program can be easily sliced else it is difficult to slice a program. This greater indicates the coupling between the class so we cannot slice the program very comfortably so we need to restructure our program for better slicing. The result of the proposed metric has been also compared with the previous used cohesion metrics and the correlation is calculated between proposed and existing metric. And the result of correlation that is given in Table V indicates that the better relationship exists between these cohesion measures. To further validate the proposed cohesion metric Weyuker's property is used, eight out of nine properties are satisfied. The experimental result reveals that the proposed measures qualify the lot of paper work formalities to be a good cohesion measure.

The future scope is focuses on some fundamental issues: this paper uses C programs without function call. So this work can be extended to validate this work using the programs including function calls.

Acknowledgement

I would like to thank my guide Dr. Kumar Rajnish, department of Computer Science and Engineering, Dr. Sandip Dutta, HOD, department of Computer Science and Engineering, BIT Mesra and all my friends without whom this work cannot be possible.

References

[1] M. Weiser. "Program slicing". IEEE Transactions on SE 10(4), 1984.
[2] J.Bieman and L. Ott. Measuring functional cohesion.IEEE Transaction on software Engineering, 20(8);644-657, August 1994.
[3] Riazur Raheman[1], Amiya Kumar Rath[2] and M Hima Bindu[3] "International Journal of Advanced Research in Computer Science and Software Engineering" Volume 3,Issue 11, November 2013,pp. 435-442.
[4] Timothy M. Meyers and David Binkley "Slice-Based Cohesion Metrics and Software Intervention" Proceedings of the 11th Working Conference on Reverse Engineering (WCRE'04) 2004 IEEE.
[5] Fumiaki OHATA, Kouya HIROSE, Katsuro INOUE and Masato FUJII "A Slicing Method For Object-oriented Programs Using Lightweight Dynamic Information" 2001 IEEE.
[6] Timothy M. Meyers and David Binkley "An Empirical study of Slice-Based Cohesion and Coupling Metrics" ACM Transactions on Software Maintenance, Vol. V, No. N, November 2007, Pages 1–25.
[7] Mehmet Kaya and James W. Fawcett "A New Cohesion Metric and Restructuring Technique for Object Oriented Paradigm" 2012 IEEE 36th International Conference on Computer Software and Applications Workshops, pp. 296-301.
[8] Sonam Jain[1], Sandeep Poonia[2] "A New approach of program slicing:Mixed S-D(static & dynamic) slicing" International Journal of Advanced Research in Computer and communication Engineering, Vol. 2,Issue 5, May 2013.
[9] Durga Prasad Mohapatra, Rajib Mall and Rajeev Kumar "An Overview of Slicing Techniques for Object-Oriented Programs" Informatica 30 (2006) 253–277.
[10] Andrea De Lucia,Anna Rita Fasolino and Malcolm Munro "Understanding Function Behaviors through Program Slicing"1996,IEEE, pp. 9-18.
[11] Heung Seok Chae, Yong Rae Kwon and Doo Hwan Bae "A Cohesion measure for Object-Oriented Classes, Softw. Pract. Exper. 2000; 30:1405–1431.
[12] Norihiro Yoshida, Masataka Kinoshita, Hajimu Iida "A cohesion metric approach to dividing source code into functional segments to improve maintainability" 2012 16th European Conference on Software Maintenance and Reengineering.
[13] David Bowes, Tracy Hall and Andrew Kerr "Program Slicing Based-Cohesion Measurement: The Challenges of Replicating Studies Using Metrics" Proceeding WETSoM '11 Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics Pages 75-80 ACM New York, NY, USA ©2011.
[14] L. Ott and J. Thuss. Slice based metrics for estimating cohesion. In Proceedings of the First Intetnationl Software Metrics Symposium, pages 71 –81, 1993 .
[15] Weyuker. J. E. "Evaluating Software Complexity Measures", IEEE Trans. on Software Engineering, 14, 1998, 1357-1365.
[16] Chidamber and Kemerer "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, VOL. 20, NO. 6, JUNE 1994.