Program Slice Metrics and Their Potential Role in DSL Design

Steve Counsell[1], Tracy Hall[1], David Bowes[2], Sue Black[3],

[1] Dept. Information Systems and Computing,
Brunel University, Uxbridge,

[2]Dept. of Computing,
University of Hertfordshire, Hatfield, UK,

[3]Dept. of Computer Science,
University of Westminster, Harrow, UK,
{Steve.Counsell, Tracy. Hall}@brunel.ac.uk,
d.h.bowes@herts.ac.uk, sblack@gmail.com.

Abstract. The advantages a DSL and the benefits its use potentially brings imply that informed decisions on the design of a domain specific language are of paramount importance for its use. We believe that the foundations of such decisions should be informed by analysis of data empirically collected from systems to highlight salient features that should then form the basis of a DSL. To support this theory, we describe an empirical study of a large OSS called Barcode, written in C, and from which we collected two well-known 'slice' based metrics. We analyzed multiple versions of the system and sliced its functions in three separate ways (i.e., input, output and global variables). The purpose of the study was to try and identify sensitivities and traits in those metrics that might inform features of a potential slice-based DSL. Results indicated that cohesion was adversely affected through the use of global variables and that appreciation of the role of function inputs and outputs can be revealed through slicing. The study presented is motivated primarily by the problems with current tools and interfaces experienced directly by the authors in extracting slicing data and the need to promote the benefits that analysis of slice data and slicing in general can offer.

Keywords: DSL, Slicing, Cohesion, Metrics.

Submitted to KISS 2009, Brisbane, 2009.

1 Introduction

A wide range of DSLs exist and, as a community, the number and application of DSLs is likely to grow in popularity in the coming years [2, 14, 15, 18, 20, 21]. One question that arises however during development of any DSL is which syntactical features it should comprise for maximum expressive power and effectiveness. We would want the design of any DSL to be informed by empirical data specific to that domain using tried and trusted software metrics. Program slicing is an area that has received a great deal of attention in the past few years and has been applied successfully in a number of program analysis contexts. While tools for extracting slice-based data exist, these do not give the user much freedom in interpreting the output, especially when a function can be sliced in multiple ways. Moreover, there are a wide range of problems associated with extracting and most importantly interpreting data extracted from slice-based tools. The data extracted in the study presented is a case in point. In this paper, we therefore provide justification for a dedicated slice-based DSL through data extracted from a large C system. Two well-known slice based metrics were extracted from multiple versions of the same system. We sliced on input, output and global variables. Results show promise in being able to highlight the features of program functions that are relevant to any analysis involving program slicing, in particular the different emphases of the three types of variable.

The paper is organized as follows. In the next section we present motivation and related work. In Section 3, we present preliminary descriptions including definition of the two metrics used in the paper. In Section 4, we present analysis of each of the three categories through the two metrics. We then present conclusions and point to future work in Section 5.

2 Motivation and Related Work

The motivation for the work described in this paper stems from several sources. First, vast amounts of slicing analysis has been undertaken in the past, but the languages used for extracting and interpreting slicing data have been notoriously difficult to use and interpret. The authors themselves suffered from a number of logistical problems of extracting slicing data from the presented Barcode system. Second, in the light of the value of program slicing as a software engineering concept, a reflective DSL for analyzing code is long overdue. This paper motivates the need for such a DSL by showing the traits of functions that slice-based metrics can reveal and the opportunities for re-engineering and refactoring that emerge as a result. Finally, the process of arriving at the data presented in this paper represented a long chain in both developer time and effort from first, using and tailoring CodeSurfer [11], second, definition and extraction of the metrics and, finally, extraction into Excel and subsequent sorting, interpretation and analysis. We feel significant benefits can be accrued through a single, one-stop, slice-based DSL. The purpose of this paper is to demonstrate the viability and visibility of such a DSL.

In terms of slicing literature, the paper from which the slicing metrics we present and which is considered the seminal slicing text is that of Weiser [27]. The techniques of program slicing have been adopted and used by many disciplines and in a multitude of contexts [4, 5, 6, 7, 19, 23, 24, 27]. Ott and Thuss explored some of Weiser's original metrics [25] and also introduced several of their own. These metrics were then analyzed from an empirical viewpoint. Bieman and Ott [3] used program slicing in the context of 'glue' that held those tokens together. Meyers and Binkley [22] undertook a large-scale empirical study of five slice-based metrics (largely those of Ott and Thuss) and provide baseline values for those metrics on a longitudinal basis; lowly-rated modules according to those baselines would be candidates for re-engineering. The research also showed that the same set of metrics could be used to analyze the decay of systems. In this paper, we try to demonstrate the value of program slicing for illuminating function features that we can abstract to a DSL.

As a concept, cohesion was introduced as early as 1979 when Yourdon and Constantine introduced their seven point ordinal scale for component cohesion [29]. Stevens et al. looked at inter-module metrics earlier [26]. In terms of the OO paradigm, the best known and most researched cohesion metric is the Lack of Cohesion of Methods (LCOM) proposed by C&K [10]. LCOM measures the relationship of methods and variables of a class by counting the number of method pairs accessing different variables minus the number of method pairs accessing the same variables. A high LCOM for a class is undesirable and indicates high complexity in that class. The research in this paper builds upon work comparing cohesion metrics and properties of metrics in general [8, 12, 13, 16] where a comparative study of OO cohesion metrics highlighted the strengths and weaknesses of each; various other studies of cohesion have also been undertaken [1, 9, 12].

3. Preliminaries

3.1 Metrics Definitions

The two metrics which we explore in this paper were originally proposed by Weiser [27], namely 'Tightness' and 'Overlap' and we use exactly the same definitions of those metrics. Before formally defining the two metrics, we first describe the formal underpinnings of a slice's components proposed by Ott and Thuss [24] and which we also adopt in this paper.

We denote a set of variables used by a function F as V_F and $V_{\{IP, OP, GL\}}$ as the subset of V_F representing either input (IP), output/return (OP) or global variables (GL); the choice of which of the three variables to slice on is drawn from the set of three elements of V. F represents a program 'function', defined as a unit of code under consideration. We further note that in the OO paradigm, this would equate to a class, the level at which OO cohesion metrics have tended to be applied in past studies [1, 10, 12]. We denote a slice SL_i as that obtained for $v_i \in V_{\{IP, OP, GL\}}$ and SL_{int} as the intersection of SL_i over all $v_i \in V_{\{IP, OP, GL\}}$. We use the same example function from

[22] for consistency. This function is shown in Appendix A, the purpose of which is to determine the smallest and largest of an array of integers. The slice of each output variable and intersection are shown, in this case for two output (smallest and largest) as output variables in 'printf' statements:

Tightness(F) = $\frac{|SL_{\text{int}}|}{length(F)}$ (Tightness measures the number of statements that occur in every slice.)

Overlap(F) = $\frac{1}{|\text{Vo}|} \sum_{i=1}^{|\text{Vo}|} \frac{|SL_{\text{int}}|}{|\text{SLi}|}$ (Overlap measures 'how many statements

in a slice are found only in that slice [22]'.)

From the definitions of Tightness and Overlap, we obtain the following values for the function in Appendix A:

$$\text{Tightness} = \frac{11}{19} = 0.58 \quad \text{and} \quad \text{Overlap} = \frac{1}{2}\left(\frac{11}{14} + \frac{11}{16}\right) = 0.74$$

The relatively high value of Overlap is due to high value of SL_{int} relative to the size of the two slices for 'smallest' and 'largest'. The value of Tightness reflects the fact that SL_{int} accounts for just over half the function length.

4 Data Analysis

4.1 Data Collection

Table 1 gives a description of the three categories used to slice as part of our analysis. We sliced in three different ways and produced the values of the two metrics for each function as a result.

Table 1. Categories of variable

Categories	**Description**
Formal ins (IP)	Input parameters for the function
Formal outs (OP)	The set of return variables
Global variables (GL)	The set of variables which are used by the *module*

We investigate two key research questions as part of our study. First, we hypothesize that any slices using GL will show significantly lower values of the two slice-based metrics than those for IP or OP; the values of the two slice-based metrics will reflect that difference. This theory is based on the belief that global variables are generally accepted as 'bad' programming practice whose use should be avoided and that program 'cohesion' is impaired by their use. Second, and most importantly can information gleaned from the values of the two metrics and their interpretation inform an understanding of the requirements for a slicing-based (DS) language?

4.2 Slicing Analysis

4.2.1 Input (IP) variables

Figure 1 gives the profile for the two metrics when considering slices based on IP variables only. The mean value of the Tightness metric from the data in this figure is 0.73 and the median Tightness value is 0.92; the corresponding values for Overlap are 0.84, with median 1. Clearly, from a cohesion perspective, IP variables contribute significantly and positively to function cohesiveness as defined by the two metrics. A high value of Tightness implies that there is a relatively high intersection set SL_{int} (and tightly bound inter-dependence between input variables). Also noticeable from Figure 1 are a number of low and zero values for both Tightness and Overlap. These are primarily from main functions (which do not have input values) and from functions whose functionality is not usually thought of input-oriented; these tend to be functions such as 'print' which generally require no input values and have a low reliance on manipulation of data. Inspection of the data revealed that these metric values were found for functions such as 'Barcode_ps_print' and 'main'.

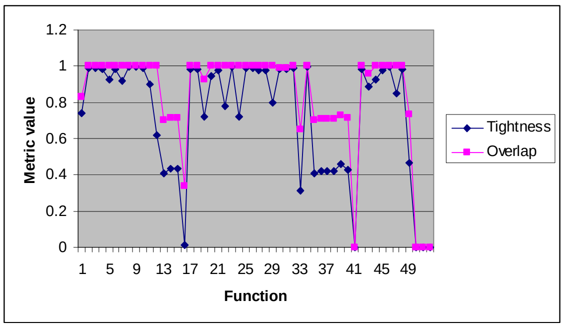

Fig. 1 Tightness and Overlap values (IP)

4.2.2 Output (OP) variables

Figure 2 shows the values of the Tightness and Overlap metrics when just slicing on OP variables is considered. The mean value of the Tightness metric for this set of values is 0.36, with median 0.30. For the Overlap metrics, the corresponding mean is 1, with median 1. Two features of the data for OP variables are evident from Figure 2 and the summary statistics. First, the values of Tightness are far lower than for IP variables (mean 0.36 for OP, compared with 0.73 for IP). The explanation for such a characteristic of the data is that OP variables tend to be used in functions to produce a single output/return value which is then returned by the function. In other words, OP variables do not tend to be as inter-dependent and intrinsic to the algorithmic operations of the function as IP variables are and this consequently affects the values of the two metrics; the SL_{int} is lower on average as a result.

The second feature of the data is the large set of values of 1 for the Overlap metric (given by the horizontal line in Figure 2). Every value for Overlap in this set of slices is 1. The reason for this high number can be explained as follows. With a relatively small number of OP variables, small slice sizes, and low intersection of those slices, the value of the Overlap will always approach (or be equal to) 1 by definition; the Tightness metric values under the same set of conditions are significantly lower because of the relationship between a low numerator (SL_{int}) and relatively high denominator (i.e., length).

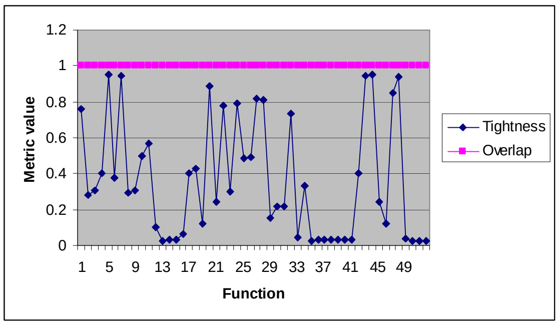

Fig. 2 Tightness and Overlap values (OP)

4.2.3 Global (GL) variables

Figure 3 shows the values of the Tightness and Overlap metrics when all Barcode functions are sliced on GL variables only. The mean value of Tightness for the set of values in Figure 1 is 0.33 and the median is 0.29. The corresponding values of mean and median for Overlap are 0.77 and 0.75, respectively. These values are significantly smaller than the corresponding values for IP and OP variables, suggesting that cohesion (measured by Tightness and Overlap) is adversely affected by the use of GL. The erratic range of values for *both* Tightness and Overlap is also a feature of Figure 3 (unlike Figures 1 and 2). In Figure 1, there appears to be a strong correlation between the Tightness and Overlap values. Because of the constant ‘1’ values in the data for OP variables, no correlation could be computed. The relatively low number of ‘1’ values for the Overlap metric in Figure 3 and the lack of Tightness values in the range 0.6-0.8 indicates the negative influence that GL has on the values of the two metrics. The explanation for the erratic values in Figure 3 is that it reflects the haphazard and inconsistent use of global variables in Barcode functions within the respective modules. In other words, the danger of using such variables in any language is in the lack of scope restriction and the problems that this presents for maintenance when the declaration of a global variable is modified.

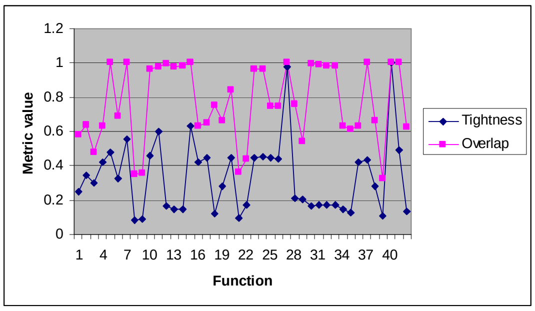

Fig. 3 Tightness and Overlap values (GL)

Table 2 shows Pearson's (parametric assuming a normal distribution), Kendall's and Spearman's (non-parametric) coefficients for the three sets of data i.e., Tightness vs. Overlap. All values are significant at the 1% level. The extent of the difference between IP variables and GL variables is evident from the values in Table 2. The correlations in Table 2 indicate that there are larger differences between the two metrics when GL are considered (c.f. input variables).

Table 2. Correlation values

Variable/Coefficient	**Pearson's**	**Kendall's**	**Spearman's**
Input variables	0.93	0.72	0.82
Output variables	n/c	n/c	n/c
Global variables	0.48	0.44	0.57

To reflect on the first of the two research question posed in Section 4.1, the over-riding conclusion from this analysis is that, in line with software engineering practice, global variables have a direct effect on the cohesion of a system when measured using Tightness and Overlap and that slicing metrics can show features of functions and modules that would not necessarily be revealed by application of static, *ad hoc* metrics. We see the study as the first in a line of studies using slicing to abstract features that could be incorporated into a slice-based DSL. Thus, in terms of the second research question, we see significant promise in slicing as a means of teasing out germane features of functions that could be incorporated into a slice-based DSL. One aspect of such a language is that it could be used to identify functions worthy of re-engineering and refactoring [17]; DSLs have been the target of particular interest from the XP/Agile and refactoring communities [17, 18] and we see this as a main thrust of future work.

5. Conclusions

In this paper, we have described an empirical analysis based on slice data. We collected two slice-based metrics, Tightness and Overlap on multiple versions of the Barcode system. The study was motivated by the difficulty we encountered as researchers in extracting and then analysing slice-based data using a gamut of tools. Our motivation was therefore two-fold; first to demonstrate that slice-based metrics combined with tool use can be a very useful software engineering mechanism. Secondly, to envisage a one-step DSL that would be of use to developers and project managers for evaluating and interpreting traits in their software. In both cases, we feel the research question was answered positively. There are a number of avenues for future work. First, we want to investigate fault data from each of the functions and the relationship between those faults and the slicing metrics. Thereafter, to integrate fault analyses with program analyses such as that presented in this paper. Second, we would like to design and implement the DSL based on program slicing to understand how and where program slicing can be targeted. All the data used in this study can be made available upon request from the authors.

Acknowledgements. The research in this paper is kindly supported by a grant from the UK Engineering and Physical Sciences Research Council (EPSRC) (EP/E055141/1).

References

1. Bansiya, J., Etzkorn, L., Davis, C., and Li, W. A class cohesion metric for object-oriented designs. Journal of Object-Oriented Programming 11(8), pp. 47-52, 1999.
2. Barstow, D., Domain-specific automatic programming. IEEE Transactions on Software Engineering, SE-11(11):1321-36, November 1985.
3. Bieman, J., and Ott, L. Measuring functional cohesion. IEEE Trans. on Software Eng. 20, 8 (1994), pp. 644-657.
4. Binkley, D. Gold, N. and Harman, M. An empirical study of static program slice size. ACM Trans. Software Engineering Methodology (TOSEM) 16(2):1-32, 2007.
5. Binkley, D., Harman, M., and Krinke, J., Empirical study of optimization techniques for massive slicing. ACM Trans. Program. Lang. Syst. 30(1): (2007)
6. Binkley D and Harman M., Locating dependence clusters and dependence pollution, IEEE International Conference on Software Maintenance, Budapest, Sept. 2005 pages 177-186.
7. Binkley, D., Harman, M., Raszewski, I., and Smith, C. An empirical study of amorphous slicing as a program comprehension tool. Proc. of the Intl. Workshop on Program Comprehension, Limerick, Ireland, pp. 161-170, 2000.
8. Bowes, D., Counsell, S and Hall, T., Calibrating program slicing metrics for practical use, Proceedings of TAIC PART, Windsor, UK, 2008, Computer Society Press.

9. Briand, L., Daly, J., and Wust, J. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering Journal 3(1), 65-117, 1998.
10. Chidamber, S., and Kemerer, C. A metrics suite for object oriented design. IEEE Trans. on Software Engineering 20(6) (1994), 467-493.
11. www.grammatech.com/products/codesurfer/
12. Counsell, S., Swift. S., and Crampton J. The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design. ACM Transactions on Software Engineering and Methodology, 15(2):123 – 149, 2006.
13. Counsell, S., Bowes D., and Hall T., Evolutionary Cohesion Metrics: The Empirical Contradiction. Proceedings of The Psychology of Programming Interest Group (PPIG), Open University, January 2009.
14. Krueger, C., Software reuse. ACM Computing Surveys, 24(2):131-183, June 1992.
15. van Deursen, A., and Klint, P., Little languages: Little maintenance? Journal of Software Maintenance, 10:75-92, 1998.
16. Fenton, N., Pfleeger, S. Software Metrics, A Rigorous and Practical Approach Thomson Intl. Comp. Press, (1996).
17. Fowler, M. Refactoring (Improving the Design of Existing Code). Addison Wesley, 1999.
18. http://martinfowler.com/articles/languageWorkbench.html
19. Gold, N., Harman, M., Binkley, D. and Hierons, R., Unifying program slicing and concept assignment for higher-level executable source code extraction. Softw., Pract. Exper. 35(10): 977-1006 (2005).
20. Herndon, R., Berzins, V., The realizable benefits of a language prototyping language. IEEE Transactions on Software Engineering, 14:803-809, 1988.
21. Mernik, M., Heering, J., Sloane, A., When and how to develop domain-specific languages. ACM Computing Surveys, 37(4):316–344, 2005.
22. Meyers, T and Binkley, D. Slice-based Cohesion Metrics and Software Intervention, Proceedings Working Conference on Reverse Engineering, Delft, Netherlands, pages 256-265.
23. Meyers, T. and Binkley, D. An empirical study of slice-based cohesion and coupling metrics. ACM Trans. on Software Engineering and Methodology, 17(1), 2007.
24. Ott L, Thuss J., (1993) Slice based metrics for estimating cohesion; Proceedings of the International Software Metrics Symposium, 71–81, Baltimore, US.
25. Ott L. and Thuss, J., The relationship between slices and module cohesion. Proceedings of International Conference on Software Engineering, Pittsburgh, US, 1989, pages 198-204.
26. Stevens, W., Myers, G., and Constantine, L. Structured design. IBM Systems Journal 13, 2 (1974), 115-139.
27. Weiser, M. Program slicing. Proceedings Int. Conf on Soft Eng., San Diego, 1981. IEEE Press, pp. 439-449.
28. Weiser M (1982) Programmers use slices when debugging, Comm. of the ACM, 25(7):446-452, July 1982
29. Yourdon, E., and Constantine, L. Structured Design. Prentice Hall, Englewood Cliffs, New Jersey, 1979.

Appendix A: Function slices taken from [22]

Function	$SL_{smallest}$	$SL_{largest}$	SL_{int}
main()			
{			
int i;	|	|	|
int smallest;	|	|	
int largest;		|	
int A[10];	|	|	|
for (i=0; i <10; i++)	|	|	|
{			
int num;	|	|	|
scanf("%d",	|	|	|
&num);	|	|	|
A[i] = num;			
}			
	|	|	|
smallest = A[0];		|	
largest=smallest;			
	|	|	|
i=1;	|	|	|
while (i <10)			
{	|		
if (smallest >	|		
A[i])		|	
smallest = A[i];		|	
if (largest < A[i])			
largest = A[i];	|	|	|
i = i +1;			
}	|		
		|	
printf("%d \n",			
smallest);			
printf("%d \n",			
largest);			
}			
Length =19	14	16	11