Journal of Southeast University (English Edition) Vol. 24, No. 2, pp. 174 – 177 June 2008 ISSN 1003—7985

Effect of slice scope on data slice-based class cohesion metrics

Zhou Yuming Xu Baowen

(School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: This paper suggests that a single class rather than methods should be used as the slice scope to compute class cohesion. First, for a given attribute, the statements in all methods that last define the attribute are computed. Then, the forward and backward data slices for this attribute are generated by using the class as the slice scope and are combined to compute the corresponding class data slice. Finally, the class cohesion is computed based on all class data slices for the attributes. Compared to traditional cohesion metrics that use methods as the slice scope, the proposed metrics that use a single class as slice scope take into account the possible interactions between the methods. The experimental results show that class cohesion can be more accurately measured when using the class as the slice scope.

Key words: cohesion; class; data slice; metrics; object-oriented

Class cohesion denotes the degree to which elements in a class belong together. It is believed that classes with high cohesion are easier to comprehend, maintain, modify, and reuse. Indeed, high cohesion within a class has been an important design goal when developing classes. However, since the fuzzy concept of cohesion is elusive, it is difficult to evaluate the cohesion of a class in practice. Therefore, much effort has been made to develop cohesion metrics in the last decade.

To date, more than twenty class cohesion metrics have been proposed[1-11]. Chidamber and Kemerer proposed a well-known cohesion metric LCOM (lack of cohesion in methods)[2]. LCOM is an inverse metric; i. e., a class with a high LCOM value means a low cohesion. Bieman and Kang proposed two cohesion metrics, TCC (tight class cohesion) and LCC (loose class cohesion)[3]. Briand et al. proposed a unified framework for cohesion measurement in object-oriented systems[4]. These cohesion metrics can only be used in the detailed design and implementation phases, as they are based on method-attribute reference information. Bansiya et al. proposed CAMC (cohesion among methods in a class)[5]. CAMC is based on method signatures in a class and hence can be used in the analysis and overall design phases. Based on CAMC, Counsell et al. proposed NHD (normalised Hamming distance) and SNHD (Scaled NHD)[6-7]. Marcus et al. proposed C3 (the conceptual cohesion of classes) based on information retrieval techniques[8].

Unlike previous work, Ott and her colleagues used data slices to quantify class cohesion[11-12]. Program slicing is a program analysis technique introduced by Weiser in 1979 and has been widely applied in program understanding, debugging, testing, maintenance, and metrics[13]. In 1984, Weiser used program slicing to define coverage, overlap, and tightness metrics[14]. Longworth found that these metrics can be adapted to measure module cohesion[15]. Bieman et al. used data slices to measure functional cohesion[16]. Ott et al. used data slices to measure class cohesion[12]. To measure the cohesion of a class, Ott et al. first computed class data slices for each attribute in the class, and then used data tokens common to class data slices as the basis to compute three metrics: strong data cohesion, weak data cohesion, and data adhesiveness. For a given attribute, the corresponding class data slice is generated by a two-step approach: 1) Compute a method data slice for this attribute in each method in this class; 2) Concatenate all method data slices to obtain the class data slice. This two-step approach assumes that methods in a class are independent. However, such is not the case in a real class. Consequently, class cohesion cannot be accurately measured. To address this problem, the class rather than individual methods should be used as the slice scope when computing a class data slice. Our results show that class cohesion can be more accurately measured when using the class as the slice scope.

1 Data Slice-Based Class Cohesion Metrics

For a statement s in a program and a variable v defined or used in s, the backward slice consists of all statements and predicates that might affect the value of v in s, and the forward slice consists of all statements and predicates that might be affected by the value of v in s. To date, many algorithms have been developed to efficiently compute both intra-procedural and inter-procedural slices[17].

In 1994, Bieman and his colleagues introduced the concept of data slices to measure functional cohesion[16]. For a function f and an output variable v of f, Bieman et al. first found the statement s that satisfies the following conditions: 1) v is defined in s; and 2) Any statement in the path from s to the exit statement does not define v. In the following, s is called a FinalDef statement. Once such a FinalDef statement s is found, a backward slice of s with respect to v, BS(f, s, v), is computed. Then, for each statement t in BS(f, s, v), if v is defined in t, then a forward slice FS(f, t, v) is computed. After that, BS (f, s, v) and all such forward slices FS (f, t, v) are combined to obtain a metric slice MS(f, v)[18]. Finally, the data slice of f with respect to v, MDS (f, v), is defined as the sequence of all data tokens (i. e. variable and constant definitions and references) in the statements and predicates in MS(f, v). Note that only intra-procedural slicing is used to compute MDS(f, v). For SumProduct shown in Tab. 1, the data slices corresponding to Sum and Prod are[11]:

Received 2007-10-17.

Biography: Zhou Yuming (1974—), male, doctor, professor, cs. zhou. yuming@ seu. edu. cn.

Foundation items: The National Natural Science Foundation of China (No. 60425206, 60633010), the High Technology Research and Development Program of Jiangsu Province (No. BG2005032).

Citation: Zhou Yuming, Xu Baowen. Effect of slice scope on data slice-based class cohesion metrics[J]. Journal of Southeast University (English Edition), 2008, 24(2): 174 – 177.

$$\begin{aligned}\mathrm{MDS}(\mathrm{SumProduct}, \mathrm{Sum}) &= n_1 \cdot \mathrm{Sum}_1 \cdot i_1 \cdot \mathrm{Sum}_2 \cdot 0_1 \cdot i_2 \cdot 1_2 \cdot \\ &\quad i_3 \cdot n_2 \cdot i_4 \cdot \mathrm{Sum}_3 \cdot \mathrm{Sum}_4 \cdot i_5 \\ \mathrm{MDS}(\mathrm{SumProduct}, \mathrm{Prod}) &= n_1 \cdot \mathrm{Prod}_1 \cdot i_1 \cdot \mathrm{Prod}_2 \cdot 1_1 \cdot i_2 \cdot 1_2 \cdot \\ &\quad i_3 \cdot n_2 \cdot i_4 \cdot \mathrm{Prod}_3 \cdot \mathrm{Prod}_4 \cdot i_6\end{aligned}$$

Tab. 1 Data slice profile for SumProduct

Sum	Prod		SumProduct
2	2	1	void SumProduct(int *n*, * Sum, * Prod);
		2	{
1	1	3	int *i*;
2		4	* Sum = 0;
	2	5	* Prod = 1;
5	5	6	for(*i* = 1; *i* < *n*; *i* + +) {
3		7	* Sum = * Sum + *i*;
		8	* Prod = * Prod * *i*;
		9	}
	3	10	}

Ott et al. computed class cohesion based on the concept of class data slices[16]. For each attribute a in a class c, the corresponding class data slice CDS(c, a) is computed using a two-step approach: 1) For each method m in c, compute MDS (m, a); 2) Generate CDS (c, a) by concatenating the data tokens obtained from all method data slices with respect to a, i. e. CDS(c, a) = MDS(m_1, a) MDS(m_2, a) … MDS(m_n, a), where m_i is the i-th method in c and n is the number of methods in c. For example, consider the Java class Stack shown in Tab. 2, the class data slice with respect to size can be computed as follows:

$$\begin{aligned}&\mathrm{MDS}(\mathrm{Stack}, \mathrm{size}) = \mathrm{size}_1 \cdot s_1 \cdot \mathrm{array}_1 \cdot \mathrm{size}_2 \\ &\mathrm{MDS}(\mathrm{IsFull}, \mathrm{size}) = \mathrm{top}_3 \cdot \mathrm{size}_3 \\ &\mathrm{MDS}(\mathrm{Size}, \mathrm{size}) = \mathrm{size}_4 \\ &\mathrm{MDS}(\mathrm{IsEmpty}, \mathrm{size}) = \mathrm{MDS}(\mathrm{Vtop}, \mathrm{size}) = \\ &\qquad \mathrm{MDS}(\mathrm{Push}, \mathrm{size}) = \mathrm{MDS}(\mathrm{Pop}, \mathrm{size}) = \varnothing\end{aligned}$$

Therefore,

$$\mathrm{CDS}(\mathrm{Stack}, \mathrm{size}) = \mathrm{size}_1 \cdot s_1 \cdot \mathrm{array}_1 \cdot \mathrm{size}_2 \cdot \mathrm{top}_3 \cdot \mathrm{size}_3 \cdot \mathrm{size}_4$$

Tab. 2 shows the data slice profile for Stack when individual methods are used as the slice scope. The class slice abstraction of a class c, CSA(c), is defined as the set of all class data slices; i. e. CSA(c) = {CSD(c, a) | a is an attribute in c}. Furthermore, if a data token lies on more than one class data slice in CSA(c), it is called a "glue" token. If a data token lies on all class data slices in CSA(c), it is called a "super-glue" token. For a class c, G(CSA(c)) and SG(CSA(c)) are respectively the set of all "glue" tokens and the set of all "super-glue" tokens in c.

Based on the concepts of glue and super-glue tokens, Ott et al. defined three class cohesion metrics; SDC(strong data cohesion), WDC(weak data cohesion), and DA(data adhesiveness):

$$\mathrm{SDC}(c) = \frac{|\,\mathrm{SG}(\mathrm{CSA}(c))\,|}{|\,\mathrm{tokens}(c)\,|}$$

$$\mathrm{WDC}(c) = \frac{|\,\mathrm{G}(\mathrm{CSA}(c))\,|}{|\,\mathrm{tokens}(c)\,|}$$

$$\mathrm{DA}(c) = \frac{\sum_{d \in G(\mathrm{CSA}(c))} |\,\{s \in \mathrm{CSA}(c) \mid s \text{ containing } d\}\,|}{|\,\mathrm{tokens}(c)\,| \times |\,\mathrm{CSA}(c)\,|}$$

where tokens(c) is the set of all data tokens of class c. SDC is the relative number of super-glue data tokens in a class c, WDC is the relative number of glue data tokens in a class c; and DA is the distribution of glue data tokens in the class data slices. All the three metrics have a value between 0 and

Tab. 2 Data slice profile for Stack: individual methods used as slice scope

Array	Top	Size		Class Stack
			1	public class Stack {
			2	private int[] array;
			3	private int top;
			4	private int size;
			5	
			6	Stack(int *s*) {
2		2	7	size = *s*;
2		2	8	array = new int[size];
	2		9	top = 0;
			10	}
			11	
			12	public boolean IsEmpty() {
	2		13	return top = = 0;
			14	}
			15	
			16	public boolean IsFull() {
	2	2	17	return(top = = size);
			18	}
			19	
			20	public int Size() {
		1	21	return size;
			22	}
			23	
			24	public int Vtop() {
3	3		25	return array[top - 1];
			26	}
			27	
			28	public void Push(int item) {
1	1		29	if(IsFull())
			30	System. out. println("Full Stack.");
			31	else
3	3		32	array[top + +] = item;
			33	}
			34	
			35	public int Pop() {
	1		36	if(IsEmpty())
			37	System. out. println("Empty Stack.");
			38	else
	1		39	- - top;
3	3		40	return array[top + 1];
			41	}
			42	}

1 and a larger value represents a higher cohesion. Among these metrics, DA is the most accurate one. SDC, WDC, and DA of the class Stack are computed as follows:

$$\mathrm{SDC}(\mathrm{Stack}) = 0$$

$$\mathrm{WDC}(\mathrm{Stack}) = \frac{16}{23} = 0.696$$

$$\mathrm{DA}(\mathrm{Stack}) = \frac{16 \times 2}{23 \times 3} = 0.464$$

2 Effect of Slice Scope on Cohesion Metrics

In section 2, the class data slice with respect to an attribute is obtained by concatenating all individual method data slices with respect to this attribute. This approach assumes that methods in a class are independent. However, it is not true in a real class when considering the following two cases:

- Constructors are not independent from other methods. Constructors are special methods in a class and are often used to initialize attributes that will be referenced by other methods. When an object of the class is created, one constructor will be implicitly or explicitly invoked. In other words, it is the first invoked method in the class. For example, Stack(int *s*) is a constructor in Stack and is used to initialize size, array, and top. All the other methods in Stack cannot be invoked prior to Stack(int *s*).
- Invoking methods and invoked methods are not independent. It is common that one method calls other methods to implement its function in a class. For example, in Stack, Push(int item) calls IsFull() and Pop() calls IsEmpty().

The assumption that methods in a class are independent may affect the accuracy of class data slices. Considering array in Stack, according to Ott et al.'s approach, CDS (Stack, array) does not include the data tokens in statement 17. However, statement 17 may affect the return value of IsFull, which may further affect whether statement 32 will be executed (in Push). Note that array is defined in statement 32. Therefore, in theory, the data tokens in statement 17 should be included in CDS(Stack, array). Similarly, according to Ott et al.'s approach, CDS(Stack, top) does not include the data tokens in statement 7. However, since the constructor Stack(int *s*) is the first invoked method, it may affect the return value of IsFull(), which may further affect whether statement 32 will be executed(in Push). Therefore, CDS(Stack, top) should contain the data tokens in statement 7.

It is clear that the inaccuracy of class data slices may result in an incorrect measurement of class cohesion. To address this problem, we suggest that the class rather than an individual method be used as the slice scope when computing class data slices. More specifically, for a given attribute *a*, we first find all FinalDef statements with respect to *a* in all methods, and then use the class as the slice scope to obtain the corresponding class data slice. Tab. 3 shows the data slice profile for Stack when the class is used as the slice scope. Compared to the data slice profile shown in Tab. 2, we can see that:

Tab. 3 Data slice profile for Stack: the class used as slice scope

Array	Top	Size		Class Stack
			1	public class Stack {
			2	private int[ ] array;
			3	private int top;
			4	private int size;
			5	
			6	Stack(int *s*) {
2	2	2	7	size = *s*;
2		2	8	array = new int[size];
2	2		9	top = 0;
			10	}
			11	
			12	public boolean IsEmpty() {
	2		13	return top = = 0;
			14	}
			15	
			16	public boolean IsFull() {
2	2	2	17	return(top = = size);
			18	}
			19	
			20	public int Size() {
		1	21	return size;
			22	}
			23	
			24	public int Vtop() {
3	3		25	return array[top - 1];
			26	}
			27	
			28	public void Push(int item) {
1	1	1	29	if(IsFull())
			30	System. out. println("Full Stack.");
			31	else
3	3	3	32	array[top + +] = item;
			33	}
			34	
			35	public int Pop() {
	1		36	if(IsEmpty())
			37	System. out. println("Empty Stack.");
			38	else
	1		39	- - top;
3	3		40	return array[top + 1];
			41	}
			42	}

- CDS (Stack, array) includes four more data tokens, i. e., two data tokens in statement 9 and two data tokens in statement 17;
- CDS(Stack, top) includes two more data tokens, i. e., two data tokens in statement 7;
- CDS(Stack, size) includes four more data tokens, i. e., one data token in statement 29 and three data tokens in statement 32.

As a result, the number of super-glue tokens increases from 0 to 8 and the number of glue tokens increases from 16 to 18. As can be seen, the metric values of SDC, WDC, and DA also have significant changes.

$$\mathrm{SDC(Stack)} = \frac{8}{23} = 0.348$$

$$\mathrm{WDC(Stack)} = \frac{18}{23} = 0.783$$

$$\mathrm{DA(Stack)} = \frac{8 \times 3 + 10 \times 2}{23 \times 3} = 0.638$$

3 Conclusion

Methods in a class are assumed to be independent in data slice-based class cohesion metrics. However, this assumption is not true in a real class. As a result, class cohesion cannot be accurately measured. To address this problem, this paper uses the class rather than individual methods such as the slice scope to compute class data slices for data slice-based cohesion metrics. The results show that class cohesion can be more accurately measured when using the class as the slice scope. We have implemented an Eclipse plug-in that uses static slice information to compute data slice-based cohesion metrics for Java classes. In future work, we will conduct experiments on open-source software to study whether the slice scope affects the fault-proneness predictability of data slice-based cohesion metrics.

References

[1] Li W, Henry S. Object-oriented metrics that predict maintainability[J]. *Journal of Systems and Software*, 1993, **23**(2): 111 – 122.

[2] Chidamber S R, Kemerer C F. A metrics suite for object-oriented design [J]. *IEEE Transactions on Software Engineering*, 1994, **20**(6): 476 – 593.

[3] Bieman J M, Kang B K. Cohesion and reuse in an object-oriented system [C]//*Proceedings of the* 1995 *Symposium on Software Reusability*. Seattle, USA, 1995: 259 – 262.

[4] Briand L C, Daly J W, Wüst J K. A unified framework for cohesion measurement in object-oriented systems [J]. *Empirical Software Engineering*, 1998, **3**(1): 65 – 117.

[5] Bansiya J, Etzkorn L, Davis C, et al. A class cohesion metric for object-oriented designs [J]. *Journal of Object-Oriented Program*, 1999, **11**(8): 47 – 52.

[6] Counsell S, Mendes E, Swift S, et al. Evaluation of an object-oriented cohesion metric through Hamming distances, BBKCS-02-10 [R]. London: Birkbeck College of University of London, 2002.

[7] Counsell S, Swift S, Cramton J. The interpretation and utility of three cohesion metrics for object-oriented design [J]. *ACM Transactions on Software Engineering and Methodology*, 2006, **15**(2): 123 – 149.

[8] Marcus A, Poshyvanyk D. The conceptual cohesion of classes [C]//*Proceedings of the* 21*st IEEE International Conference on Software Maintenance*. Budapest, Hungary, 2005: 133 – 142.

[9] Chae H S, Kwon Y R, Bae D H. A cohesion measure for object-oriented classes [J]. *Software*: *Practice & Experience*, 2000, **30**(12): 1405 – 1431.

[10] Chen Zhenqiang, Zhou Yuming, Xu Baowen, et al. A novel approach to measuring class cohesion based on dependence analysis [C]//*Proceedings of the* 18*th International Conference on Software Maintenance*. Montreal, Canada, 2002: 377 – 384.

[11] Ott L M, Bieman J M. Program slices as an abstraction for cohesion measurement [J]. *Information and Software Technology*, 1998, **40**(11/12): 691 – 699.

[12] Ott L M, Bieman J, Kang B K, et al. Developing measures of class cohesion for object-oriented software [C]//*Proceedings of Annual Oregon Workshop on Software Metrics*. Oregon, Portland, 1995.

[13] Weiser M D. Program slices: formal, psychological, and practical investigations of an automatic program abstraction method[D]. Michigan: Department of Computer and Communication Sciences of University of Michigan, 1979.

[14] Weiser M D. Program slicing [J]. *IEEE Transactions on Software Engineering*, 1984, **10**(4): 352 – 357.

[15] Longworth H D. Slice based program metrics[D]. Michigan: Department of Computer Science of Michigan Technological University, 1985.

[16] Bieman J M, Ott L M. Measuring functional cohesion [J]. *IEEE Transactions on Software Engineering*, 1994, **20**(8): 644 – 657.

[17] Ottenstein K J, Ottenstein L M. The program dependence graph in a software development environment [J]. *ACM SIGPLAN Notices*, 1984, **19**(5): 177 – 184.

[18] Ott L M, Thuss J J. Slice based metrics for estimating cohesion [C]//*Proceedings of the* 9*th Conference on Software Maintenance*. Baltimore, Maryland, USA, 1993: 71 – 81.

切片作用域对基于数据切片类内聚性度量的影响

周毓明　　徐宝文

（东南大学计算机科学与工程学院，南京 210096）

摘要：提出应该使用类而不是方法作为切片作用域来计算类的内聚性. 首先，对一个给定的属性，找出每个方法中对该属性最后一次定义的语句. 然后，以类为切片作用域计算这些语句相对于该属性的前向和后向数据切片，通过合并得到该属性的类数据切片. 最后，在所有属性的类数据切片的基础上计算类的内聚性. 与传统的以方法为切片作用域的内聚性度量相比，以类为切片作用域的内聚性度量考虑了类中方法之间可能存在的交互. 实验结果表明，当使用类为切片作用域时，所得的度量值能更准确地量化类的内聚性.

关键词：内聚性；类；数据切片；度量；面向对象

中图分类号：TP311.5