Information and Software Technology 75 (2016) 90–104

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

An empirical investigation into the effect of slice types on slice-based cohesion metrics

Yibiao Yang, Yangyang Zhao, Changsong Liu, Hongmin Lu, Yuming Zhou*, Baowen Xu

State Key Laboratory for Novel Software Technology, Nanjing University, China

ARTICLE INFO

Article history:
Received 2 June 2015
Revised 28 February 2016
Accepted 1 April 2016
Available online 12 April 2016

Keywords:
Cohesion
End slice
Metric slice
Metrics

ABSTRACT

Context: There is a debate about whether end slice or metric slice is preferable for computing slice-based cohesion metrics. However, up till now, there is no consensus about this issue.
Objective: We aim to investigate the relationship between end-slice-based and metric-slice-based cohesion metrics and then determine which type of slice is preferable for computing slice-based cohesion metrics.
Method: We used forty widely used open-source software systems to conduct the study. First, we compute the baseline values for end-slice-based and metric-slice-based cohesion metrics. Then, we investigate their relationships with module size. Finally, we employ correlation analysis and principal component analysis to analyze the relationships between end-slice-based and metric-slice-based cohesion metrics.
Results: End-slice-based and metric-slice-based cohesion metrics have similar baseline metric values. Furthermore, they exhibit a similar negative correlation with module size. In particular, the results from correlation analysis and principal component analysis reveal that they essentially measure the same cohesion dimensions.
Conclusion: From the viewpoint of metric values, there is little difference between end-slice-based and metric-slice-based cohesion metrics. We hence recommend choosing end slice for computing slice-based cohesion metrics in practice, as extra cost involved in data collection could be avoided.

1. Introduction

Cohesion is an important software quality attribute which refers to the tightness of elements within a module [1,2]. Highly cohesive modules are desirable in a system as they are easier to develop, maintain, and reuse, and hence are less fault-prone [1,2]. In the last three decades, researchers developed many slice-based cohesion metrics to quantify the cohesion of a module at the function level of granularity [3–10]. In [11], Meyers and Binkley found that slice-based cohesion metrics could be used to quantify the deterioration that accompanies software evolution. More recently, our study shows that slice-based cohesion metrics are not redundant with respect to the most commonly used code/process metrics and are of practically important value in the context of fault-proneness prediction [12]. Therefore, slice-based cohesion metrics can be used as an important quality indicator for practitioners. In other words, practitioners could use these metrics to identify potentially faulty modules for quality enhancement.

For a given function, the computation of a slice-based cohesion metric consists of the following three steps. The first step is to identify the output variables of the function, including function return values, modified global variables, printed variables, and modified reference parameters [13]. The second step is to employ program slicing techniques to obtain one slice with respect to each output variable of the function [9,14]. The third step is to use the resulting slices to compute the slice-based cohesion metric. Of these three steps, the second step is the most crucial, as it is the basis for the computation of slice-based cohesion metrics. In the literature, there is a debate about whether end slice or metric slice should be used for computing slice-based cohesion metrics [11,12,15]. An end slice with respect to variable v is the *backward slice* with respect to v at the end point of the module [9,16]. A metric slice with respect to variable v is the union of the *backward slice* with respect to v at the end point of the module and the *forward slice* computed from the top of the backward slice [7]. Consequently, a metric slice is more time-consuming to compute compared with an end slice. In most previous studies, end slice was used to compute slice-based cohesion metrics [5–9,11,15–18]. However, Ott and her colleagues argued that metric slice can result in more accurate cohesion metrics [10]. Up till now, there is

* Corresponding author. Tel.: +86 25 89682450.
E-mail address: cs.zhou.yuming@gmail.com (Y. Zhou).

http://dx.doi.org/10.1016/j.infsof.2016.04.001
0950-5849/

no consensus about which type of slice is preferable for slice-based cohesion metrics.

In this study, we aim to attack this issue. To this end, we conduct a comparative study to investigate the relationship between end-slice-based and metric-slice-based cohesion metrics. The subject systems in our study consist of forty open-source software systems. We use a source code analysis tool called Frama-C to collect slice-based cohesion metrics. Based on the data collected from these forty systems, we first compute the baseline metric values for end-slice-based and metric-slice-based cohesion metrics. As stated by Meyers and Binkley [11], baseline values for slice-based metrics "are useful for identifying degraded modules" and can "aid in the transfer of technology from academia to industry". Similar to [11], we use the average metric values and the 95% confidence interval to compute the baseline metric values. Then, we investigate their relationships with module size. This will help us determine "whether slice-based cohesion metrics are nothing more than a proxy for module size" [11]. If they are, it will mean that they do not provide new information. Consequently, there would be no need to use them in practice, especially considering the relatively high computation cost incurred. Finally, we employ correlation analysis and principal component analysis (PCA) to analyze the relationships between end-slice-based cohesion metrics and metric-slice-based cohesion metrics. The purpose of this aims to determine whether end-slice-based and metric-slice-based cohesion metrics indeed measure the same cohesion dimensions, although different types of slices are used. The experimental results show that end-slice-based and metric-slice-based cohesion metrics have similar baseline metric values. Furthermore, they exhibit a similar negative correlation with module size. In other words, modules with larger size tend to be less cohesive. In particular, the results from correlation analysis and PCA reveal that end-slice-based and metric-slice-based cohesion metrics essentially measure the same software cohesion dimensions. Therefore, from the viewpoint of metric values, there is little difference between end-slice-based and metric-slice-based cohesion metrics. We hence recommend choosing end slice rather than metric slice, as extra cost evolved in data collection could be avoided.

The rest of this paper is organized as follows. Section 2 introduces slice-based cohesion metrics that we will investigate. Section 3 introduces the experimental methodology used for this study, including the data sets and the data analysis method. Section 4 reports in detail the experimental results. Section 5 discusses the findings. Section 6 presents the related work. Section 7 examines the threats to validity of our study. Section 8 concludes the paper and outlines directions for future work.

2. Slice-based cohesion metrics

In this section, we first introduce the concept of end slice and metric slice. Then, we describe these slice-based cohesion metrics that will be investigated in this study. Finally, we use an example function to illustrate the computations of end-slice-based and metric-slice-based cohesion metrics.

2.1. End slice and metric slice

As aforementioned, for a given module, the most crucial step for the computations of slice-based cohesion metrics is to obtain either the end slice or the metric slice for each output variable of a module. An end slice with respect to variable v is the *backward slice* with respect to v at the end point of the module [9,16]. A metric slice with respect to variable v is the union of the *backward slice* with respect to v at the end point of the module and the *forward slice* computed from the top of the backward slice [7]. More specifically, a *backward slice* of a module at statement n with respect to variable v is the sequence of all statements and predicates that might affect the value of v at n. A *forward slice* of a module at statement n with respect to variable v is the sequence of all statements and predicates that might be affected by the value of v at n. It is easy to know that end slice only takes into account the *uses* data relationship. However, metric slice also takes into account the *used by* data relationship [3].

We next use an example function *fun* shown in Table 1, which aims to determine the smallest, the largest, and the range of an array, to illustrate the concepts of end slice and metric slice. In Table 1, the first column lists the statement number (excluding non-executable statements such as blank statements, "{", and "}"). The second column lists the code of the example function. As can be observed, the output variables of function *fun* are *smallest*, *largest*, and *range*. The former two variables are the modified reference parameters and the latter one is the function return value. The third to fifth columns list the end slice for each output variable. The sixth to eighth columns list the forward slice computed from the top of the end slice with respect to each output variable. The ninth to eleventh columns list the metric slice for each output variable. Here, a vertical bar "|" in the last nine columns denotes that the indicated statement is part of the corresponding slice for the named output variable. As can be seen, statement is the basic unit for these slices. In other words, end slice and metric slice shown in Table 1 are indeed *statement-level end slice* and *statement-level metric slice*. We next use the output variable *largest* as an example to explain the computation of end slice and metric slice. In Table 1, for the output variable *largest* of the function *fun*, we can find that: 1) the 7th, 9th, 10th, 11th, 14th, 15th, and 16th statements belong to its end slice since these statements will have an direct or indirect impact on the final value of *largest* at the end of the function *fun*; 2) the 14th, 15th, 17th, and 18th statements belong to its forward slice of the variable largest since these four statements are directly or indirectly impacted by the first definition statement of variable *largest* (i.e. the 10th statement "∗largest = ∗smallest;").

In order to obtain the dependencies at a finer granularity, researchers further propose the concepts of data-token-level end slice and metric slice, in which data token is the basic unit [3]. As shown in the second column in Table 1, one statement might consist of a number of data tokens (i.e. the definitions of and references to variables and constants). For example, the ninth statement "∗smallest = A[0];" consists of the following three data tokens: "smallest", "A", and "0″. In this sense, data token is a finer granularity than statement. Table 2 shows the data-token-level end slice and metric slice for each output variable of the function fun. In this table, T_i in the second column indicates the i-th data token for T in the function, "1″ and "0″ in the last six columns respectively denotes that the indicated data-token is or is not part of the corresponding slice for the named output variable.

2.2. Slice-based cohesion metrics

After obtaining the end slices or metric slices for individual output variables of a given function, we can use them to calculate slice-based cohesion metrics. In this study, a function is regarded as a module and the output variables of a function consist of the function return value, modified global variables, modified reference parameters, and standard outputs by the function. In the last three decades, researchers proposed the following ten slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Tightness, Overlap, SFC, WFC, A, NHD*, and *SBFC* [3–10].

Of these ten metrics, *Coverage, Tightness*, and *Overlap* can be originally traced back to Weiser's work [16]. For a given module, Weiser first sliced on every output statement occurred in the mod-

Table 1
Statement-level end slice and metric slice for function fun.

Line	Code	End slice			Forward slice			Metric slice		
		largest	smallest	range	largest	smallest	range	largest	smallest	range
	int fun(									
1	int A[]	|	|	|				|	|	|
2	int size,	|	|	|				|	|	|
3	int *largest,	|		|				|		|
4	int *smallest)	|	|	|				|	|	|
	{										
5	int i;	|	|	|				|	|	|
6	int range;			|							|
7	i = 1;	|	|	|				|	|	|
8	range = 0;										
9	*smallest = A[0];	|	|	|		|		|	|	|
10	*largest = *smallest;	|		|	|	|		|	|	|
11	while(i < size) {	|	|	|				|	|	|
12	if(*smallest > A[i])		|	|		|			|	|
13	*smallest = A[i]		|	|		|			|	|
14	if(*largest < A[i])	|		|	|	|		|	|	|
15	*largest = A[i]	|		|	|	|		|	|	|
16	i++;	|	|	|				|	|	|
	}										
17	range = *largest - *smallest;			|	|	|	|	|	|	|	|
18	return range;			|	|	|	|	|	|	|	|
	}										

Data tokens included in the end slice for smallest are underscored

Table 2
Data-token level end slice and metric slice.

Line	Token	End slice			Metric slice		
		largest	smallest	range	largest	smallest	range
1	A_1	1	1	1	1	1	1
2	$size_1$	1	1	1	1	1	1
3	$largest_1$	1	0	1	1	0	1
4	$smallest_1$	1	1	1	1	1	1
5	i_1	1	1	1	1	1	1
6	$range_1$	0	0	1	0	0	1
7	i_2	1	1	1	1	1	1
7	1_1	1	1	1	1	1	1
8	$range_2$	0	0	0	0	0	0
8	0_1	0	0	0	0	0	0
9	$smallest_2$	1	1	1	1	1	1
9	A_2	1	1	1	1	1	1
9	0_2	1	1	1	1	1	1
10	$largest_2$	1	0	1	1	1	1
10	$smallest_2$	1	0	1	1	1	1
11	i_3	1	1	1	1	1	1
11	$size_2$	1	1	1	1	1	1
12	$smallest_4$	0	1	1	0	1	1
12	A_3	0	1	1	0	1	1
12	i_4	0	1	1	0	1	1
13	$smallest_5$	0	1	1	0	1	1
13	A_4	0	1	1	0	1	1
13	i_5	0	1	1	0	1	1
14	$largest_3$	1	0	1	1	1	1
14	A_5	1	0	1	1	1	1
14	i_6	1	0	1	1	1	1
15	$largest_4$	1	0	1	1	1	1
15	A_6	1	0	1	1	1	1
15	i_7	1	0	1	1	1	1
16	i_8	1	1	1	1	1	1
17	$range_3$	0	0	1	1	1	1
17	$largest_5$	0	0	1	1	1	1
17	$smallest_6$	0	0	1	1	1	1
18	$range_4$	0	0	1	1	1	1

ule to obtain end slices and then used those end slices to compute *Coverage, Overlap*, and *Tightness*. In his work, *Coverage* was defined as the ratio of average slice size to module size, *Overlap* was defined as the average ratio of non-unique to unique statements in each slice, and *Tightness* was defined as the percentage of statements common in all slices. After that, Longworth investigated Weiser's slice-based metrics and found that *Coverage*, a modified definition of *Overlap*, and *Tightness* could be used as cohesion metrics of a module. According to Longworth, the modified definition of *Overlap* was the average ratio of non-unique statements to slice size [7]. Later, Ott and Thuss formalized all these three slice-based cohesion metrics [3]. They redefined *Overlap* as the average ratio of the slice interaction (called "cohesive section" by Harman *et al.* [6]) size to slice size and proposed two additional slice-based cohesion metrics: *MinCoverage* and *MaxCoverage*. They defined *MinCoverage* as the ratio of the size of the smallest slice to the module size and *MaxCoverage* as the ratio of the size of the largest slice to the module size. Furthermore, they introduced the concept of *metric slices* and used *metric slices* to compute slice-based cohesion metrics. Ott and Thuss chose to slice on output variables rather than output statements, including function return values, modified global variables, printed variables, and modified reference parameters. As such, the slice-based cohesion metrics suite proposed by Ott and Thuss consists of five metrics: *Coverage, Overlap, Tightness, MinCoverage*, and *MaxCoverage*. More specifically, for a given module *M, Coverage* measures the extent to which the slices cover the module by the ratio of the mean slice size to the module size:

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

where V_o denotes the set of output variables of module *M*, SL_i is the slice obtained for $v_i \in V_o$, and $length(M)$ denotes the module size. For a given module *M, MinCoverage* measures the extent to which the smallest slice covers the module by the ratio of the size of the smallest slice to the module size:

$$MinCoverage = \frac{1}{length(M)}\min_i |SL_i| \quad (2)$$

In contrast, *MaxCoverage* measures the extent to which the largest slice covers the module by the ratio of the size of the largest slice to the module size:

$$MaxCoverage = \frac{1}{length(M)}\max_i |SL_i| \quad (3)$$

Overlap measures the extent to which slices are interdependent by the average ratio of the size of the "cohesive section" to the size

of each slice:

$$Overlap = \frac{1}{|V_o|}\sum_{i=1}^{|V_o|}\frac{|SL_{\text{int}}|}{|SL_i|} \quad (4)$$

where SL_{int} (called "cohesive section" by Harman *et al.* [6]) is the intersections of SL_i over all $v_i \in V_o$. *Tightness* measures the extent to which all the slices in the module belong together by the ratio of the size of the "cohesive section" to the module size:

$$Tightness = \frac{|SL_{\text{int}}|}{length(M)} \quad (5)$$

SFC, *WFC*, and *A* were proposed by Bieman and Ott [4]. They used metric slices to compute these three metrics. More specifically, they defined the slice abstraction of a module as the set of data-token-level metric slices with respect to its output variables. In particular, a data token is called a "glue token" if it lies on more than one metric slices, and is called a "super-glue token" if it lies on all metric slices in the slice abstraction. On the one hand, *SFC* measures the extent to which all the slices in the module belong together by the ratio of the number of super-glue tokens to the total number of data tokens in the module:

$$SFC = \frac{|SG(SA(M))|}{|tokens(M)|} \quad (6)$$

where $SA(M)$ is the slice abstraction of module M, $SG(SA(M))$ is the set of super-glue tokens in module M, and $tokens(M)$ is the set of data tokens in module M. On the other hand, *WFC* measures the extent to which the slices in the module belong together by the ratio of the number of glue tokens to the total number of data tokens in the module:

$$WFC = \frac{|G(SA(M))|}{|tokens(M)|} \quad (7)$$

where $G(SA(M))$ is the set of glue tokens in module M. In addition, *A* measures the extent to which the glue tokens in the module are adhesive by the average ratio of the number of glue tokens to the total number of data tokens in the module:

$$A = \frac{\sum_{t \in G(SA(M))} \text{slices containing t}}{|tokens(M)| \times |SA(M)|} \quad (8)$$

Noted that *SFC* is exactly equivalent to the data-token-level *Tightness* metric and *A* is equivalent to the data-token-level *Coverage* metric proposed by Ott and Bieman [10].

NHD (normalized Hamming distance) and SBFC were respectively proposed by Counsell *et al.* [5] and Dallal [8]. Both NHD and SBFC are based on the concept of slice occurrence matrix. NHD is originally defined based on the statement-level end slice, while SBFC is originally defined based on the data-token-level end slice. For a given module, the slice occurrence matrix has columns indexed by its output variables and rows indexed by its statements or data tokens. The (i, j)-th entry of the matrix has a value of 1 if the i-th statement or i-th data token is in the end slice with respect to the j-th output variable and otherwise 0. In this matrix, each row is called a slice occurrence vector. Table 2 shows the data-token level slice occurrence matrix with respect to end slice profile and metric slice profile for the example function *fun*. According to Counsell et al [5], *NHD* measures the extent to which the statements in the slices are the same by the ratio of the total actual slice agreement between rows to the total possible agreement between rows in the statement-level slice occurrence matrix:

$$NHD = 1 - \frac{2}{lk(k-1)}\sum_{j=1}^{l} c_j(k - c_j) \quad (9)$$

The slice agreement between two rows is the number of places in which the slice occurrence vectors of the two rows are equal. Here, k is the number of statements, l is the number of output variables, and c_i is the number of 1 s in the j-th column of the statement-level slice occurrence matrix. According to Dallal [8], *SBFC* measures the extent to which the slices are similar by the average degree of the normalized similarity between columns in the data-token-level slice occurrence matrix:

$$SBFC = \begin{cases} 1 & if |V_o| = 1 \\ \frac{\sum_{t=1}^{|tokens(M)|} x_i(x_i - 1)}{|tokens(M)| \times |V_o| \times (|V_o| - 1)} & \text{otherwise} \end{cases} \quad (10)$$

The normalized similarity between a pair of columns is the ratio of the number of entries where both columns have a value of 1 to the total number of rows in the matrix. Here, x_i is the number of 1 s in the i-th row of the data-token-level slice occurrence matrix.

The above-mentioned ten slice-based cohesion metrics are originally defined with different types of slices: end slices or metrics slices. Table 3 provides an overview of the main literature concerning slice-based cohesion metrics. For each study, Table 3 lists the year published, the studied cohesion metrics, whether the study is an empirical study relating slice-based cohesion metrics to external quality attributes, the tool used for collecting slice-based cohesion metrics, and the kind of slice they used for slice-based cohesion metrics. As can be seen, except the last one, all empirical studies used end slice to compute slice-based cohesion metrics. According to the definitions of end slice and metric slice, we know that metric slice size is larger than or equal to end slice size for a given output variable of a module. Meyers and Binkley chose to use end slices to compute slice-based cohesion metrics for the reason that metric slices are "likely to be larger than a traditional slice with a corresponding drop in usefulness" [11]. At the same time, they pointed out that metric slice might contain more information and hence more useful, or, conversely, metric slice might include redundancy information and hence less useful. Therefore, they highlighted that "empirical investigation is required to determine the relation between the utility of these two definitions". Unlike these studies, our previous study used metric slices to compute these slice-based cohesion metrics since we observed that metric-slice-based cohesion metrics appear to be more accurate than end-slice-based cohesion metrics from an example function [12]. However, we still do not know which one should be used for computing slice-based cohesion metrics. Overall, the core observation from Table 3 is that, up till now, there is no consensus about whether end slice or metric slice should be used for computing slice-based cohesion metrics. In this study, we aim to determine which type of slice is preferable for computing slice-based cohesion metrics. This study is important for both researchers and practitioners, as it can guide the development of practical fault-proneness prediction models.

Note that all the ten slice-based cohesion metrics can be computed at the statement or data-token level, although some of them are originally defined either at the statement level or at the data-token level. The data-token level is at a finer granularity than the statement level since a statement might contain a number of data-tokens. Previous studies suggested that software metrics at a finer granularity would have a higher discriminative power and hence may be more useful for fault prediction [20,21]. In this study, we focus on a comprehensive comparison of metric-slice-based cohesion metrics with end-slice-based cohesion metrics. Therefore, we take into account slice-based cohesion metrics not only at the statement level but also at the data-token level. As aforementioned, at both data-token level and statement level, *SFC* is equivalent to *Tightness* and *A* is equivalent to *Coverage*. Therefore, in the subsequent analysis in our study, we will investigate the following thirty-two slice-based cohesion metrics:

Table 3
Slice-based cohesion metrics with respect to end slice and metric slice.

Study	Year	The Studied Cohesion Metrics	ES?	Tool	End or metric slice
Weiser [9,16]	1981	Coverage, Overlap, Tightness	No	N/A	End slice
Longworth [7]	1985	Coverage, Overlap, Tightness	No	N/A	End slice
Ott and Bieman [19]	1992	Coverage, MaxCoverage, MinCoverage, Overlap, Tightness	No	N/A	Metric slice
Ott and Thuss [3]	1993	Coverage, MaxCoverage, MinCoverage, Overlap, Tightness	No	N/A	Metric slice
Bieman and Ott [4]	1994	SFC, WFC, A	No	N/A	Metric slice
Harman *et al.* [6]	1995	Coverage, MaxCoverage, MinCoverage, Tightness, Overlap	No	N/A	End slice
Ott and Bieman [10]	1998	Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, SFC, WFC, A	No	N/A	Metric slice and End slice
Meyers and Binkley [15]	2004	Coverage, MaxCoverage, MinCoverage, Tightness, Overlap	Yes	Codesurfur	End slice
Meyers and Binkley [11]	2007	Coverage, MaxCoverage, MinCoverage, Tightness, Overlap	Yes	Codesurfur	End slice
Bowes *et al.* [17]	2008	Coverage, MaxCoverage, MinCoverage, Tightness, Overlap	Yes	Codesurfer	End slice
Black *et al.* [18]	2009	Tightness, Overlap	Yes	Codesurfur	End slice
Dallal [8]	2009	SBFC, Tightness, Overlap	Yes	Codesurfur	End slice
Counsell *et al.* [5]	2010	NHD, Tightness, Overlap	Yes	Codesurfur	End slice
Yang *et al.* [12]	2015	Coverage, MaxCoverage, MinCoverage, Tightness, Overlap, SFC, WFC, A, SBFC, NHD	Yes	Frama-C	Metric slice

ES: empirical study. N/A: not available

Table 4
Example metrics computations.

Type	Metric	Data-token level		Statement level	
		Computation	Value	Computation	Value
End slice	Coverage	= 1/3×(21/34+18/34+32/34)	= 0.696	= 1/3×(12/18+10/18+17/18)	= 0.722
	MaxCoverage	= 32/34	= 0.941	= 17/18	= 0.944
	MinCoverage	= 18/34	= 0.529	= 10/18	= 0.556
	Overlap	= 1/3×(12/21+12/18+12/32)	= 0.538	= 1/3×(8/12+8/10+8/17)	= 0.646
	Tightness	= 12/34	= 0.353	= 8/18	= 0.444
	WFC	= 27/34	= 0.794	= 14/18	= 0.778
	SBFC	= (12×3×2+15×2×1)/(34×3×2)	= 0.500	= (8×3×2+6×2×1)/(18×3×2)	= 0.556
	NHD	= 1-2/(3×34×33)×(21×13+18×16+32×2)	= 0.629	= 1-2/(3×18×17)×(12×6+10×8+17×1)	= 0.632
Metric slice	Coverage	= 1/3×(25/34+30/34+32/34)	= 0.853	= 1/3×(14/18+15/18+17/18)	= 0.852
	MaxCoverage	= 32/34	= 0.941	= 17/18	= 0.944
	MinCoverage	= 25/34	= 0.735	= 14/18	= 0.778
	Overlap	= 1/3×(24/25+24/30+24/32)	= 0.837	= 1/3×(13/14+13/15+13/17)	= 0.853
	Tightness	= 24/34	= 0.706	= 13/18	= 0.722
	WFC	= 31/34	= 0.912	= 16/18	= 0.889
	SBFC	= (24×3×2+7×2×1)/(34×3×2)	= 0.775	= (13×3×2+3×2×1)/(18×3×2)	= 0.778
	NHD	= 1-2/(3×34×33)×(25×9+30×4+32×2)	= 0.757	= 1-2/(3×18×17)×(14×4+15×3+17×1)	= 0.743

- Data-token-level end-slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, WFC, NHD*, and *SBFC*.
- Data-token-level metric-slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, WFC, NHD, and SBFC*.
- Statement-level end-slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, WFC, NHD*, and *SBFC*.
- Statement-level metric-slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Overlap, Tightness, WFC, NHD, and SBFC*.

2.3. An illustrating example

In this section, we illustrate the computations of the above-mentioned end-slice-based and metric-slice-based cohesion metrics for the example function fun in Section 2.1. Table 4 shows the computations of slice-based cohesion metrics. In this table, the second to ninth rows show the computations for end-slice-based cohesion metrics and the tenth to seventeenth rows show the computations for metric-slice-based cohesion metrics.

As can be seen, end-slice-based metrics indicate with typical values around 0.5 or 0.6, while metric-slice-based metrics indicate with typical values around 0.7 or 0.8. In particular, for each cohesion metric (except end-slice-based *MaxCoverage* is equal to metric-slice-based *MaxCoverage*), the metric-slice-based version has a considerably larger value than the corresponding end-slice-based version.

3. Research methodology

In this section, we first introduce the data sets used in this study. Then, we describe the data analysis methods.

3.1. Data sets

Table 5 summarizes the 40 subject systems used in this study. As can be seen, for each subject system, Table 5 reports the lines of C code (blank lines and comments are not included), the number of functions that can be identified by Frama-C, and the number of functions after preprocessing (e.g. the functions without output variables are excluded). This study makes use of these 40 systems for the following three reasons. First, they are widely used open source projects. Second, they can be analyzed by Frama-C [22], thus allowing us to collect slice-based cohesion metrics. Third, they are non-trivial software belonging to different problem domains.

The data sets used in this study are collected from these 40 systems. Each data point of a data set corresponds to one C function and consists of: 1) one source code size metric and 2) thirty-two slice-based cohesion metrics (8 end-slice-based cohesion metrics at statement level + 8 end-slice-based cohesion metrics at data-token level + 8 metric-slice-based cohesion metrics at statement level + 8 metric-slice-based cohesion metrics at data-token level). We used the following steps to obtain the data. As in [12], we

Table 5
The studied systems in our study.

System	Lines of C code	# functions	# function after preprocessing	System	Lines of C code	# functions	# function after preprocessing
acm-5.1	32,684	696	655	libgdata-0.8.0	35,371	2,070	1867
apr-1.0.0	45,579	549	523	libgsasl-1.8.0	30,434	315	314
bash-2.05	69,663	1,551	1,426	libiconv-1.10	86,432	314	309
cairo-1.0.0	46,217	1,142	1,108	libosip2-2.2.0	23,106	707	652
cim-5.1	26,073	575	501	librep-0.16	31,171	1,018	912
dico-2.0	56,824	1,370	1,265	libsecret-0.8	29,427	1,105	1,033
direvent-5.0	22,831	494	455	libunistring-0.9.2	205,777	581	549
freeipmi-1.2.8	282,472	3,567	3,451	lightning-2.0.0	63,261	582	573
gawk-4.0.0	46,454	800	706	m4-1.4.12	46,946	578	518
glib-2.18.4	217,224	6,739	6,112	make-4.0	26,371	359	310
global-6.2	46,266	915	845	mdk-1.0	21,015	483	456
glpk-4.40	60,054	1,405	1,214	mifluz-0.25.0	53,268	935	839
gmp-5.0.0	90,249	610	534	mpfr-2.4.0	59,713	400	361
gnats-4.2.0	28,855	571	519	mtools-4.0.10	17,589	536	474
gnuit-4.9.5	29,614	501	471	readline-4.3	19,062	481	459
gnu-pw-mgr-1.0	28,672	364	336	rush-1.5	29,189	484	450
gsl-1.8	170,254	4,574	4,509	uucp-1.07	53,613	767	707
gstreamer-1.0.1	121,860	3,870	3,376	vim-6.0	7,179	2,733	2,202
idutils-4.2	30,216	434	403	wget-1.12	40,022	631	584
libcm-0.1.1	9,530	537	485	zile-2.3.10	36,381	963	874

Table 6
Baseline values for slice-based cohesion metrics.

Type	Metric	Our results				Meyers and Binkley's results
		Data-Token		Statement		Statement
		Median	Mean	Median	Mean	Mean
End slice	Coverage	0.724 [0.718, 0.728]	0.695 [0.692, 0.698]	0.706 [0.701, 0.714]	0.685 [0.682, 0.687]	0.537 [0.534, 0.539]
	MaxCoverage	0.881 [0.877, 0.885]	0.779 [0.776, 0.781]	0.846 [0.842, 0.850]	0.766 [0.764, 0.769]	0.643 [0.640, 0.645]
	MinCoverage	0.644 [0.639, 0.652]	0.601 [0.597, 0.605]	0.630 [0.625, 0.639]	0.598 [0.595, 0.602]	0.334 [0.331, 0.338]
	Overlap	0.998 [0.991, 1.000]	0.772 [0.769, 0.775]	0.991 [0.982, 1.000]	0.767 [0.764, 0.770]	0.537 [0.532, 0.541]
	Tightness	0.622 [0.610, 0.632]	0.578 [0.575, 0.581]	0.588 [0.576, 0.600]	0.571 [0.567, 0.574]	0.296 [0.292, 0.299]
	WFC	0.748 [0.741, 0.750]	0.673 [0.670, 0.676]	0.712 [0.706, 0.714]	0.656 [0.653, 0.659]	–
	SBFC	0.881 [0.863, 0.895]	0.678 [0.675, 0.682]	0.818 [0.801, 0.842]	0.665 [0.661, 0.668]	–
	NHD	0.743 [0.739, 0.747]	0.772 [0.770, 0.7740]	0.708 [0.705, 0.712]	0.750 [0.748, 0.752]	–
Metric slice	Coverage	0.750 [0.745, 0.757]	0.708 [0.705, 0.711]	0.733 [0.727, 0.738]	0.697 [0.694, 0.700]	–
	MaxCoverage	0.902 [0.900, 0.905]	0.794 [0.791, 0.796]	0.867 [0.862, 0.867]	0.780 [0.778, 0.783]	–
	MinCoverage	0.683 [0.671, 0.690]	0.615 [0.612, 0.619]	0.667 [0.647, 0.667]	0.610 [0.607, 0.613]	–
	Overlap	0.998 [0.991, 1.000]	0.779 [0.776, 0.782]	0.990 [0.980, 1.000]	0.773 [0.770, 0.776]	–
	Tightness	0.656 [0.647, 0.667]	0.592 [0.588, 0.596]	0.619 [0.605, 0.625]	0.582 [0.578, 0.586]	–
	WFC	0.790 [0.783, 0.800]	0.692 [0.689, 0.696]	0.750 [0.750, 0.750]	0.676 [0.673, 0.679]	–
	SBFC	0.901 [0.894, 0.913]	0.691 [0.688, 0.695]	0.853 [0.833, 0.867]	0.676 [0.673, 0.679]	–
	NHD	0.757 [0.753, 0.762]	0.778 [0.776, 0.780]	0.714 [0.714, 0.718]	0.755 [0.753, 0.756]	–

Table 7
Correlation coefficient of slice-based cohesion metrics with SLOC (P: Pearson; S: spearman).

Type	Metric	(a) Our results				(b) Meyers and Binkley's results	
		Data token		Statement		Statement	
		P	*S*	*P*	*S*	*P*	*S*
End slice	Coverage	–0.226	–0.611	–0.223	–0.598	0.242	0.157
	MaxCoverage	–0.203	–0.586	–0.204	–0.572	0.120	–0.042
	MinCoverage	–0.234	–0.632	–0.232	–0.619	–0.046	–0.224
	Overlap	–0.156	–0.487	–0.153	–0.479	0.069	–0.043
	Tightness	–0.218	–0.603	–0.215	–0.591	0.005	–0.107
	WFC	–0.163	–0.558	–0.163	–0.545	–	–
	SBFC	–0.190	–0.510	–0.184	–0.496	–	–
	NHD	–0.147	–0.498	–0.132	–0.458	–	–
Metric slice	Coverage	–0.220	–0.612	–0.217	–0.598	–	–
	MaxCoverage	–0.173	–0.571	–0.177	–0.559	–	–
	MinCoverage	–0.237	–0.638	–0.234	–0.623	–	–
	Overlap	–0.165	–0.502	–0.162	–0.492	–	–
	Tightness	–0.222	–0.614	–0.218	–0.599	–	–
	WFC	–0.153	–0.558	–0.152	–0.542	–	–
	SBFC	–0.193	–0.519	–0.186	–0.503	–	–
	NHD	–0.149	–0.510	–0.133	–0.469	–	–

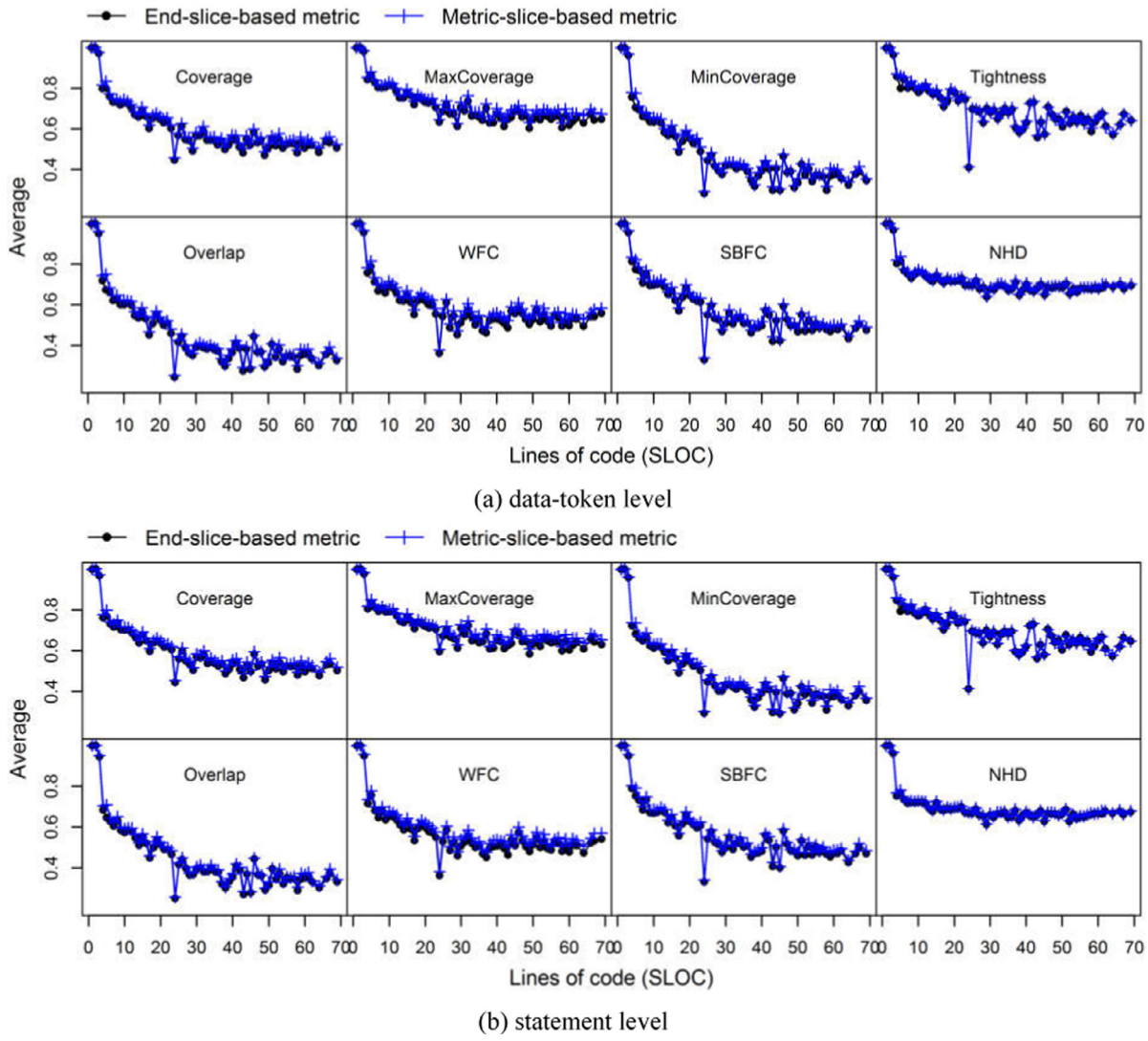

Fig. 1. Average metric values for modules with different SLOC.

use intra-procedural slicing to compute slice-based cohesion metrics for each function. In other words, metric slices are computed within a single function. In our study, calls to other functions were handled conservatively. More specifically, we developed two Frama-C plug-ins named INFERCON (INFER CONtract) and SLIBCOM (SLIce-Based COhesion Metrics) to compute slice-based cohesion metrics for each function. The INFERCON plugin is used to infer the contract for a called function conservatively. For a function, if it has a return value, INFERCON assumes that the returned value is data-dependent on all the arguments passed to the function. For any pointer argument p in the function, INFERCON assumes that the value pointed by p will be changed at the end of the function and hence p is data-dependent on all the arguments passed to the function. In addition, INFERCON assumes that any function is a terminating function. The SLIBCOM plugin is used to collect slice-based cohesion metrics for each function in each system. The SLIBCOM plug-in was based on the INFERCON plug-in and the other four plug-ins provided by Frama-C, namely "Value analysis", "Outputs", "Slicing", and "Impact analysis". For each function f in the system, the SLIBCOM plugin collected slice-based cohesion metrics by the following steps: 1) used INFERCON to infer contracts for the functions called by f; 2) employed the "Value analysis" plug-in to perform the value analysis in a context-insensitive way by using the inferred contracts for the called functions; 3) based on the results from the value analysis, used the "Outputs" plug-in to obtain the output variables of f; 4) leveraged the "Slicing" plug-in to obtain the end slices for each output variable; 5) based on those end slices, used the "Impact analysis" plug-in to obtain the corresponding "forward slices" and then combined them to obtain the metric slices for each output variable; and 6) used end slice profiles and metric slice profiles to respectively calculate end-slice-based and metric-slice-based cohesion metrics for the function f. Note that, the cohesion metric value of a function was set to undefined if either of the following two conditions was satisfied: (1) the execution time for the value analysis was very long; (2) the "Outputs" plug-in did not find any output variable. The reason for the former case is unknown. In our study, we terminated the value analysis when the execution time was longer than 30 minutes. The latter case occurred when the function under analysis did not return anything and had no side effect as well. In this case, the "Outputs" plug-in was unable to identify any output variable for the function under analysis. Besides, we compute the source code size metric SLOC for each function by using Frama-C.

In Table 5, the third and the seventh columns report the total number of functions can be identified by Frama-C. The forth and the eight columns report the total number of functions after

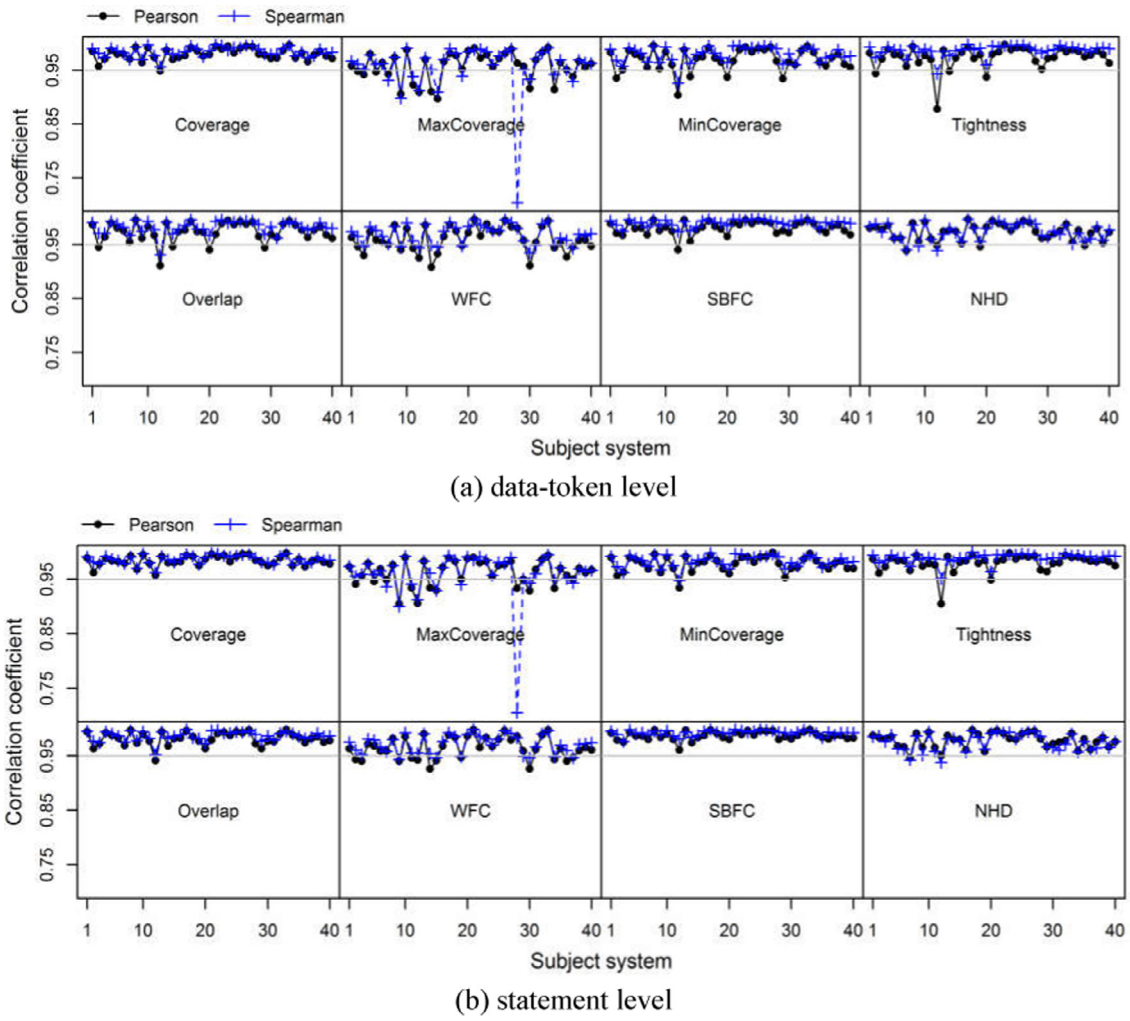

Fig. 2. The correlation coefficients between end-slice-based and metric-slice-based cohesion metrics.

removing the functions that have an "undefined" metric value. As can be seen, for each data set, only a very small number of functions were removed. We use only the data sets after pre-processing in all the subsequent analyses (in Section 4).

3.2. Data analysis method

In the following, we describe the data analysis method used in this study. We use both correlation analysis and principal component analysis (PCA) to analyze the relationship between end-slice-based cohesion metrics and metric-slice-based cohesion metrics with respect to statement level and data-token level. Following Meyers and Binkley's work [11], we apply both Pearson correlation and Spearman correlation to quantify the associations between end-slice-based and metric-slice-based cohesion metrics. This enables us to directly compare our results with their results. Pearson correlation can help examine whether there is a linear correlation between two variables. Although Pearson correlation assumes that the variables are normally distributed, as highlighted by Meyers and Binkley [11], the inferences do not depend crucially on the normality assumption [23]. In particular, when used with Spearman correlation together, we could examine whether there may be a non-linear correlation between the variables (this happens when there is a weak Pearson correlation but a strong Spearman correlation). According to Ott and Longnecker [23], for correlation coefficient r, the association is considered either weak ($r \leq 0.5$), moderate ($0.5-0.8$), or strong ($0.8 \leq r \leq 1.0$). In our context, if there is a strong correlation between end-slice-based and metric-slice-based cohesion metrics, they are, to some extent, redundant with respect to each other. Furthermore, we apply PCA, a statistical technique, to identify the underlying orthogonal dimensions (i.e. principal components, PCs) to explain the relations among the independent variables (end-slice-based and metric-slice-based cohesion metrics in this study) in a data set. These PCs are linear combinations of the standardized independent variables. In our study, for each data set, we use the following method to determine the corresponding number of PCs. First, we retain the PCs with eigenvalues larger than zero. Second, we apply the varimax rotation to PCs to map the independent variables to components. This helps to identify the variables that are strongly correlated and indeed measure the same property, though they may purport to capture different properties. Third, after obtaining the rotated component matrix, we map each independent variable to the component having the maximum loading. Fourth, we only retain the components to which at least one independent variable is mapped. In our context, end-slice-based and metric-slice-based cohesion metrics do not have any significant differences with each other if PCA shows that none of them define new PCs of their own.

4. Experimental results

In this section, we describe the experimental results for end-slice-based and metric-slice-based cohesion metrics in detail. In Section 4.1, we analyze their baseline metric values. In Section 4.2,

Table 8
PCA results for slice-based cohesion metrics at data-token level.

System	PC1		PC2		PC3		PC4		All
	%Var	Metric	%Var	Metric	%Var	Metric	%Var	Metric	
acm-5.1	0.391	I+I'+O+O'+T+T'+S+S'	0.230	C+C'+A+A'	0.156	N+N'	0.137	W+W'	√
apr-1.0.0	0.371	I+I'+O+O'+T+T'+S+S'	0.306	C+C'+A+A'+W+W'	0.125	N+N'	–	–	√
bash-2.05	0.369	I+I'+O+O'+T+T'+S+S'	0.227	C+C'+A+A'	0.172	N+N'	0.120	W+W'	√
cairo-1.0.0	0.367	I+I'+O+O'+T+T'+S+S'	0.233	C+C'+A+A'	0.151	N+N'	0.145	W+W'	√
cim-5.1	0.415	I+I'+O+O'+T+T'+S+S'	0.260	C+C'+A+A'	0.171	N+N'	0.098	W+W'	√
dico-2.0	0.407	C+C'+A+A'+I+I'+W+W'	0.321	O+O'+T+T'+S+S'	0.158	N+N'	–	–	√
direvent-5.0	0.407	C+C'+A+A'+I+I'+W+W'	0.336	O+O'+T+T'+S+S'	0.161	N+N'	–	–	√
freeipmi-1.2.8	0.424	I+I'+O+O'+T+T'+S+S'	0.218	C+C'+A+A'	0.158	N+N'	0.137	W+W'	√
gawk-4.0.0	0.379	I+I'+O+O'+T+T'+S+S'	0.358	C+C'+A+A'+W+W'	0.158	N+N'	–	–	√
glib-2.18.4	0.459	C+C'+A+A'+I+I'+T+T'+W+W'	0.263	O+O'	0.163	N+N'	0.082	S+S'	√
global-6.2	0.372	I+I'+O+O'+T+T'+S+S'	0.359	C+C'+A+A'+W+W'	0.162	N+N'	–	–	√
glpk-4.40	0.427	C+C'+A+A'+I+I'+T'+W+W'	0.305	O+O'+T+S+S'	0.145	N+N'	–	–	×
gmp-5.0.0	0.507	C+C'+A+A'+I+I'+T+T'+W+W'	0.297	O+O'+S+S'	0.147	N+N'	–	–	√
gnats-4.2.0	0.382	I+I'+O+O'+T+T'+S+S'	0.253	C+C'+A+A'	0.168	N+N'	0.115	W+W'	√
gnu-pw-mgr-1.0	0.435	C+C'+I+I'+O+O'+T+T'+S+S'	0.196	A+A'	0.157	N+N'	0.151	W+W'	√
gnuit-4.9.5	0.419	C+C'+A+A'+I+I'+T+T'+W+W'	0.327	O+O'+S+S'	0.158	N+N'	0.031		√
gsl-1.8	0.530	I+I'+O+O'+T+T'+W+W'+S+S'	0.357	C+C'+A+A'+N+N'	–	–	–	–	√
gstreamer-1.0.1	0.440	C+C'+A+A'+I+I'+T+T'+W+W'	0.262	O+O'	0.160	N+N'	0.082	S+S'	√
idutils-4.2	0.391	I+I'+O+O'+T+T'+S+S'	0.317	C+C'+A+A'	0.146	N+N'	0.085	W+W'	√
libcm-0.1.1	0.350	C+C'+A+A'+W+W'	0.339	I+I'+O+O'+T+T'+S+S'	0.174	N+N'	–	–	√
libgdata-0.8.0	0.351	I+I'+O+O'+T+T'+S+S'	0.303	C+C'+A+A'+W+W'	0.164	N+N'	–	–	√
libgsasl-1.8.0	0.381	I+I'+O+O'+T+T'+S+S'	0.263	C+C'+A+A'	0.176	N+N'	0.138	W+W'	√
libiconv-1.10	0.379	O+O'+T+T'+S+S'	0.365	C+C'+A+A'+I+I'+W+W'	0.176	N+N'	–	–	√
libosip2-2.2.0	0.417	C+C'+A+A'+I+I'+T+W+W'	0.333	O+O'+T'+S+S'	0.151	N+N'	–	–	×
librep-0.16	0.319	O+O'+T+T'+S+S'	0.253	C+C'+A+A'	0.169	N+N'	0.149	I+I'	√
libsecret-0.8	0.449	C+C'+A+A'+I+I'+T+T'+W+W'	0.296	O+O'+S+S'	0.163	N+N'	–	–	√
libunistring-0.9.2	0.397	O+O'+T+T'+S+S'	0.395	C+C'+A+A'+I+I'+W+W'	0.131	N+N'	–	–	√
lightning-2.0.0	0.419	I+I'+O+O'+T+T'+S+S'	0.260	C+C'+A+A'	0.170	N+N'	0.085	W+W'	√
m4-1.4.12	0.384	I+I'+O+O'+T+T'+S+S'	0.324	C+C'+A+A'+W+W'	0.172	N+N'	–	–	√
make-4.0	0.361	C+C'+A+A'+W+W'	0.355	I+I'+O+O'+T+T'+S+S'	0.160	N+N'	–	–	√
mdk-1.0	0.436	C+C'+A+A'+I+I'+T+T'+W+W'	0.322	O+O'+S+S'	0.150	N+N'	–	–	√
mifluz-0.25.0	0.563	C+C'+A+A'+I+I'+T+T'+W+W'	0.264	O+O'+S+S'	0.133	N+N'	–	–	√
mpfr-2.4.0	0.528	C+C'+A+A'+I+I'+T+T'+W+W'	0.284	O+O'+S+S'	0.151	N+N'	–	–	√
mtools-4.0.10	0.421	I+I'+O+O'+T+T'+S+S'	0.214	C+C'+A+A'	0.158	N+N'	0.153	W+W'	√
readline-4.3	0.329	I'+O+O'+T+T'+S+S'	0.222	C+C'+A+A'	0.174	N+N'	0.122	W+W'	×
rush-1.5	0.379	I+I'+O+O'+T+T'+S+S'	0.341	C+C'+A+A'+W+W'	0.164	N+N'	–	–	√
uucp-1.07	0.373	C+C'+A+A'+W+W'	0.348	I+I'+O+O'+T+T'+S+S'	0.156	N+N'	–	–	√
vim-6.0	0.387	I+I'+O+O'+T+T'+S+S'	0.238	C+C'+A+A'	0.162	N+N'	0.121	W+W'	√
wget-1.12	0.373	C+C'+A+A'+W+W'	0.356	I+I'+O+O'+T+T'+S+S'	0.162	N+N'	–	–	√
zile-2.3.10	0.364	C+C'+A+A'+W+W'	0.315	I+I'+O+O'+T+T'+S+S'	0.166	N+N'	–	–	√

√: indicate all end-slice-based and metric-slice-based metrics are in the same principal components
×: indicate all end-slice-based and metric-slice-based metrics are in different principal components

we investigate their relationships with module size as measured by SLOC. In Section 4.3, we report the results from correlation analysis and principal component analysis.

4.1. Baseline metric values

Table 6 summarizes the baseline values for slice-based cohesion metrics. The third and the fourth columns respectively present the median and mean values for slice-based cohesion metrics at data-token level. The fifth and the sixth columns respectively present the median and mean values for slice-based cohesion metrics at statement level. The seventh column presents the mean values for five slice-based cohesion metrics at statement level provided in [11]. In Table 6, after median and mean values, we also report the 95% confidence intervals. We use the most commonly used nonparametric bootstrap percentile confidence interval method [24], which makes no assumptions concerning the distribution of the data, to build the 95% confidence intervals for median and mean, respectively. Besides, our mean or median values are based on 43,360 modules of 40 systems while Meyers and Binkley's results were based on 22,651 modules. The mean metric values can provide the expected baseline values for slice-based cohesion metrics. As stated in [11], baseline values "make it possible to separate aberrant functional behavior from that which is "normal", thus enabling reverse engineers to identify degraded modules.

From Table 6, we can find that, for each slice-based cohesion metric, both the median and the mean value of the metric-slice-based version are only slightly larger than (less than 3%) the corresponding end-slice-based version. This indicates that, for each output variable, the metric slice (i.e. *backward slice* + *forward slice*) is almost the same as the corresponding end slice (i.e. *backward slice*) on average. Therefore, this indeed reveals that, in practice, there is no need to collect the *forward slice* for each output variable for computing slice-based cohesion metrics, as extra data collection cost could be avoided.

When compared with the mean values provided by Meyers and Binkley, we can find that the mean metric values (i.e. those values in the sixth column) in our study are considerably larger. The large difference between our study and Meyers and Binkley's study is due to the following possible reasons. First, we use different subject systems. The forty subject systems in our study include 43,360 modules while Meyers and Binkley's study had 22,651 modules. Second, the output variables are different. In Meyers and Binkley's study, output variables only consist of function return values and modified global variables. In our study, in addition to function re-

Table 9
PCA results for slice-based cohesion metrics at statement level.

System	PC1		PC2		PC3		PC4		All
	%Var	Metric	%Var	Metric	%Var	Metric	%Var	Metric	
acm-5.1	0.386	I+I'+O+O'+T+T'+S+S'	0.236	C+C'+A+A'	0.165	N+N'	0.127	W+W'	√
apr-1.0.0	0.373	I+I'+O+O'+T+T'+S+S'	0.252	C+C'+A+A'	0.145	N+N'	0.111	W+W'	√
bash-2.05	0.360	I+I'+O+O'+T+T'+S+S'	0.230	C+C'+A+A'	0.176	N+N'	0.125	W+W'	√
cairo-1.0.0	0.354	I+I'+O+O'+T+T'+S+S'	0.249	C+C'+A+A'	0.151	N+N'	0.135	W+W'	√
cim-5.1	0.409	I+I'+O+O'+T+T'+S+S'	0.260	C+C'+A+A'	0.186	N+N'	0.086	W+W'	√
dico-2.0	0.414	C+C'+A+A'+I+I'+T+T'+W+W'	0.311	O+O'+S+S'	0.165	N+N'	–	–	√
direvent-5.0	0.410	C+C'+A+A'+I+I'+W+W'	0.336	O+O'+T+T'+S+S'	0.169	N+N'	–	–	√
freeipmi-1.2.8	0.403	I+I'+O+O'+T+T'+S+S'	0.214	C+C'+A+A'	0.163	N+N'	0.129	W+W'	√
gawk-4.0.0	0.370	I+I'+O+O'+T+T'+S+S'	0.356	C+C'+A+A'+W+W'	0.163	N+N'	–	–	√
glib-2.18.4	0.465	C+C'+A+A'+I+I'+T+T'+W+W'	0.258	O+O'	0.168	N+N'	0.083	S+S'	√
global-6.2	0.384	I+I'+O+O'+T+T'+S+S'	0.346	C+C'+A+A'+W+W'	0.167	N+N'	–	–	√
glpk-4.40	0.409	C+C'+A+A'+I'+T'+W+W'	0.321	I+O+O'+T+S+S'	0.142	N+N'	–	–	×
gmp-5.0.0	0.489	C+C'+A+A'+I+I'+T+T'+W+W'	0.313	O+O'+S+S'	0.152	N+N'	–	–	√
gnats-4.2.0	0.380	I+I'+O+O'+T+T'+S+S'	0.248	C+C'+A+A'	0.176	N+N'	0.132	W+W'	√
gnu-pw-mgr-1.0	0.455	C+C'+I+I'+O+O'+T+T'+S+S'	0.214	A+A'	0.159	N+N'	0.119	W+W'	√
gnuit-4.9.5	0.423	C+C'+A+A'+I+I'+T+T'+W+W'	0.322	O+O'+S+S'	0.165	N+N'	–	–	√
gsl-1.8	0.532	C+I+I'+O+O'+T+T'+W+W'+S+S'	0.345	C'+A+A'+N+N'	–	–	–	–	×
gstreamer-1.0.1	0.449	C+C'+A+A'+I+I'+T+T'+W+W'	0.249	O+O'	0.163	N+N'	0.094	S+S'	√
idutils-4.2	0.381	I+I'+O+O'+T+T'+S+S'	0.310	C+C'+A+A'+W	0.159	N+N'	0.082	W'	×
libcm-0.1.1	0.334	C+C'+A+A'+W+W'	0.334	I+I'+O+O'+T+T'+S+S'	0.181	N+N'	–	–	√
libgdata-0.8.0	0.401	C+C'+A+A'+W+W'	0.319	I+I'+O+O'+T+T'+S+S'	0.171	N+N'	–	–	√
libgsasl-1.8.0	0.344	I+I'+O+O'+T+T'+S+S'	0.286	C+C'+A+A'	0.193	N+N'	0.131	W+W'	√
libiconv-1.10	0.444	C+C'+A+A'+I+I'+T+T'+W+W'	0.333	O+O'+S+S'	0.171	N+N'	–	–	√
libosip2-2.2.0	0.423	C+C'+A+A'+I+I'+T+T'+W+W'	0.335	O+O'+S+S'	0.156	N+N'	–	–	×
librep-0.16	0.308	C+C'+I+I'+T+T'+W+W'	0.272	O+O'+S+S'	0.203	A+A'	0.169	N+N'	√
libsecret-0.8	0.473	C+C'+A+A'+I+I'+T+T'+W+W'	0.278	O+O'+S+S'	0.164	N+N'	–	–	√
libunistring-0.9.2	0.448	C+C'+A+A'+I+I'+W+W'+N+N'	0.390	O+O'+T+T'+S+S'	–	–	–	–	√
lightning-2.0.0	0.403	I+I'+O+O'+T+T'+S+S'	0.269	C+C'+A+A'	0.189	N+N'	0.081	W+W'	√
m4-1.4.12	0.375	I+I'+O+O'+T+T'+S+S'	0.318	C+C'+A+A'+W+W'	0.184	N+N'	–	–	√
make-4.0	0.363	I+I'+O+O'+T+T'+S+S'	0.351	C+C'+A+A'+W+W'	0.160	N+N'	–	–	√
mdk-1.0	0.391	I+I'+O+O'+T+T'+S+S'	0.335	C+C'+A+A'+W+W'	0.164	N+N'	–	–	√
mifluz-0.25.0	0.562	C+C'+A+A'+I+I'+T+T'+W+W'	0.269	O+O'+S+S'	0.135	N+N'	–	–	√
mpfr-2.4.0	0.522	C+C'+A+A'+I+I'+T+T'+W+W'	0.295	O+O'+S+S'	0.151	N+N'	–	–	√
mtools-4.0.10	0.417	I+I'+O+O'+T+T'+S+S'	0.214	C+C'+A+A'	0.165	N+N'	0.152	W+W'	√
readline-4.3	0.322	I'+O+O'+T+T'+S+S'	0.239	C+C'+A+A'	0.176	N+N'	0.118	W+W'	×
rush-1.5	0.371	I+I'+O+O'+T+T'+S+S'	0.340	C+C'+A+A'+W+W'	0.173	N+N'	–	–	√
uucp-1.07	0.392	C+C'+A+A'+W+W'	0.324	I+I'+O+O'+T+T'+S+S'	0.152	N+N'	–	–	√
vim-6.0	0.373	I+I'+O+O'+T+T'+S+S'	0.247	C+C'+A+A'	0.166	N+N'	0.118	W+W'	√
wget-1.12	0.361	I+I'+O+O'+T+T'+S+S'	0.358	C+C'+A+A'+W+W'	0.168	N+N'	–	–	√
zile-2.3.10	0.373	C+C'+A+A'+I+I'+W+W'	0.287	O+O'+T+T'+S+S'	0.172	N+N'	–	–	√

√: indicate all end-slice-based and metric-slice-based metrics are in the same principal components
×: indicate all end-slice-based and metric-slice-based metrics are in different principal components

turn values and modified global variables, output variables also include printed variables and modified reference parameters.

4.2. Correlation with module size

In this section, we report the experimental results for the relationship between slice-based cohesion metrics and module size. Table 7 presents the correlation coefficients of slice-based cohesion metrics with SLOC. In this table, insignificant coefficients (i.e. p-value > 0.05) are marked in gray. Note that we combined the data from 40 systems to compute the correlation. This method is consistent with Meyer and Binkley's study [1], thus allowing us to directly compare our results with their results.

From Table 7, we have the following observations. First, under both data-token level and statement level, the coefficients between metric-slice-based cohesion metrics and SLOC are similar to those coefficients between end-slice-based cohesion metrics and SLOC. This indicates that, for each metric, end-slice-based version does not have a large difference with the corresponding metric-slice-based version. Second, all the correlation coefficients are negative and statistically significant in our study. This indicates that slice-based cohesion metrics are negatively correlated with module size. In other words, modules with a larger size tend to have lower cohesion metric values. However, the results from Meyers and Binkley were mixed for slice-based cohesion metrics at statement level. In their study, *Coverage* was found to be positively related to module size, which is hard to interpret. Furthermore, for *MaxCoverage* and *Overlap*, the Pearson correlation indicated that they were positively related to module size but the Spearman correlation indicated the opposite direction. This inconsistence between our study and their study may be due to the fact that we use different subject systems and more kinds of output variables. Compared with Meyers and Binkley's results, we believe that our results are more intuitively consistent.

Fig. 1 shows how slice-based cohesion metrics change with module size. For each metric, this figure reports the average metric value (i.e. the y axis) for those modules having the same module size (i.e. the x axis). As can be seen, for each metric, the metric-slice-based version has an average similar to the corresponding end-slice-based version. In particular, along with the growth of module size, the average metric values show a decreasing trend, regardless of whether the metric-slice-based version or the end-slice-based version is considered. In other words, large modules tend to be less cohesive than small modules. These results are consistent with those findings in Table 7.

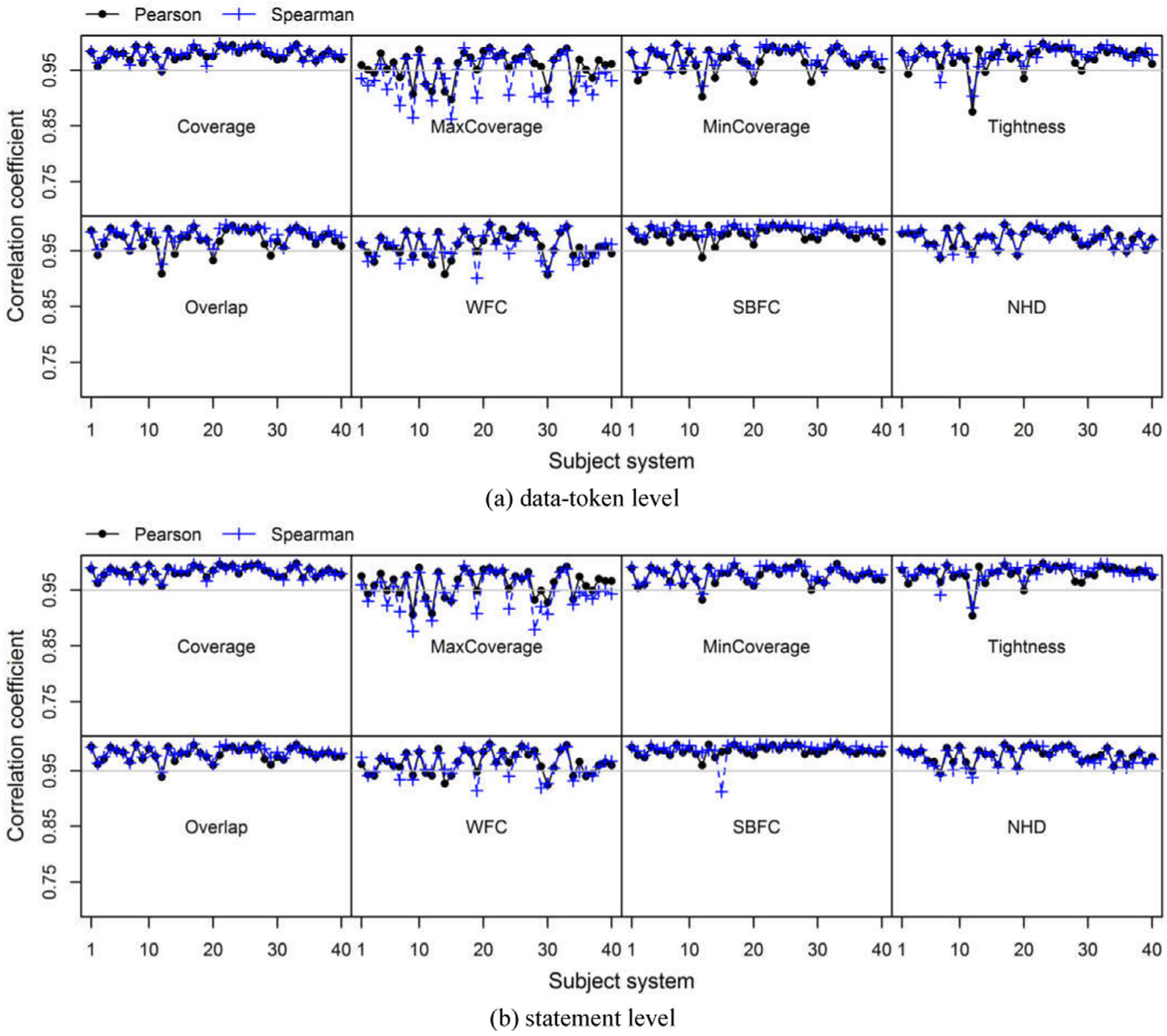

Fig. 3. The correlation coefficients between end-slice-based and metric-slice-based cohesion metrics after removing the confounding effect of module size.

4.3. The relationship between each pair of end-slice-based and metric-slice-based cohesion metrics

In this section, we use the results from both correlation analysis and PCA to investigate the relationship between each pair of end-slice-based and metric-slice-based cohesion metrics.

4.3.1. Correlation analysis

Fig. 2 presents, for each pair of end-slice-based and metric-slice-based cohesion metrics, the correlation coefficients from both Pearson correlation and Spearman correlation analysis on each subject system. Fig. 2 (a) presents the correlation coefficients for each pair of metrics at data-token level while Fig. 2 (b) is at statement level. The black lines represent the Pearson correlation coefficients and the blue lines represent Spearman correlation coefficients. Note that all these coefficients have a p-value less than 0.001.

From Fig. 2, we can find that, for each pair of end-slice-based and metric-slice-based metrics, almost all the correlation coefficients are larger than 0.9 and most of them are larger than 0.95 (the only exception is *MaxCoverage* on one system). This is true for both data-token and statement levels. The overall results reveal that each end-slice-based metric is strongly statistically correlated with the corresponding metric-slice-based version.

4.3.2. Principal component analysis

Tables 8 and 9 summarize the rotated components from PCA for end-slice-based and metric-slice-based cohesion metrics respectively at data-token level and statement level for each data set. In Tables 8 and 9, the first column is the system name. The second, forth, sixth, and eighth columns report the percentage of variance explained by the top 4 rotated components. The third, fifth, seventh, and ninth columns show how the metrics are clustered into the rotated components. The characters “C”, “A”, “I”, “T”, “O”, “W”, “S”, and “N” respectively represent the end-slice-based Coverage, MaxCoverage, MinCoverage, Tightness, Overlap, WFC, SBFC, and NHD metrics. While those characters followed by “′” represent the corresponding metric-slice-based versions. “√” in the last column indicates that all paired end-slice-based and metric-slice-

Table 10
PCA results for metrics at data-token level after removing the confounding effect of module size.

System	PC1		PC2		PC3		PC4		All
	%Var	Metric	%Var	Metric	%Var	Metric	%Var	Metric	
acm-5.1	0.375	I+I'+O+O'+T+T'+S+S'	0.226	C+C'+A+A'	0.156	N+N'	0.139	W+W'	√
apr-1.0.0	0.367	I+I'+O+O'+T+T'+S+S'	0.305	C+C'+A+A'+W+W'	0.127	N+N'	–	–	√
bash-2.05	0.367	I+I'+O+O'+T+T'+S+S'	0.232	C+C'+A+A'	0.170	N+N'	0.117	W+W'	√
cairo-1.0.0	0.371	I+I'+O+O'+T+T'+S+S'	0.231	C+C'+A+A'	0.151	N+N'	0.146	W+W'	√
cim-5.1	0.408	I+I'+O+O'+T+T'+S+S'	0.258	C+C'+A+A'	0.170	N+N'	0.103	W+W'	√
dico-2.0	0.408	C+C'+A+A'+I+I'+W+W'	0.320	O+O'+T+T'+S+S'	0.158	N+N'	–	–	√
direvent-5.0	0.403	C+C'+A+A'+I+I'+W+W'	0.330	O+O'+T+T'+S+S'	0.159	N+N'	–	–	√
freeipmi-1.2.8	0.421	I+I'+O+O'+T+T'+S+S'	0.218	C+C'+A+A'	0.158	N+N'	0.137	W+W'	√
gawk-4.0.0	0.374	I+I'+O+O'+T+T'+S+S'	0.361	C+C'+A+A'+W+W'	0.158	N+N'	0.032	–	√
glib-2.18.4	0.458	C+C'+A+A'+I+I'+T+T'+W+W'	0.258	O+O'	0.162	N+N'	0.087	S+S'	√
global-6.2	0.367	I+I'+O+O'+T+T'+S+S'	0.363	C+C'+A+A'+W+W'	0.163	N+N'	0.041	–	√
glpk-4.40	0.429	C+C'+A+A'+I+I'+T'+W+W'	0.298	O+O'+T+S+S'	0.144	N+N'	0.049	–	×
gmp-5.0.0	0.504	C+C'+A+A'+I+I'+T+T'+W+W'	0.299	O+O'+S+S'	0.145	N+N'	0.016	–	√
gnats-4.2.0	0.379	I+I'+O+O'+T+T'+S+S'	0.256	C+C'+A+A'	0.168	N+N'	0.113	W+W'	√
gnu-pw-mgr-1.0	0.435	C+C'+I+I'+O+O'+T+T'+S+S'	0.196	A+A'	0.157	N+N'	0.151	W+W'	√
gnuit-4.9.5	0.416	C+C'+A+A'+I+I'+T+T'+W+W'	0.324	O+O'+S+S'	0.155	N+N'	0.036	–	√
gsl-1.8	0.527	I+I'+O+O'+T+T'+W+W'+S+S'	0.358	C+C'+A+A'+N+N'	–	–	–	–	√
gstreamer-1.0.1	0.430	C+C'+A+A'+I+I'+T+T'+W+W'	0.257	O+O'	0.154	N+N'	0.093	S+S'	√
idutils-4.2	0.379	I+I'+O+O'+T+T'+S+S'	0.324	C+C'+A+A'+W'	0.147	N+N'	0.082	W	×
libcm-0.1.1	0.337	C+C'+A+A'+W+W'	0.334	I+I'+O+O'+T+T'+S+S'	0.169	N+N'	–	–	√
libgdata-0.8.0	0.366	I+I'+O+O'+T+T'+S+S'	0.322	C+C'+A+A'+W+W'	0.163	N+N'	–	–	√
libgsasl-1.8.0	0.378	I+I'+O+O'+T+T'+S+S'	0.266	C+C'+A+A'	0.175	N+N'	0.135	W+W'	√
libiconv-1.10	0.377	O+O'+T+T'+S+S'	0.372	C+C'+A+A'+I+I'+W+W'	0.174	N+N'	–	–	√
libosip2-2.2.0	0.399	C+C'+A+A'+I+I'+W+W'	0.334	O+O'+T+T'+S+S'	0.149	N+N'	–	–	√
librep-0.16	0.309	O+O'+T+T'+S+S'	0.244	C+C'+A+A'	0.178	I+I'	0.167	N+N'	√
libsecret-0.8	0.431	C+C'+A+A'+I+I'+T+T'+W+W'	0.279	O+O'+S+S'	0.153	N+N'	–	–	√
libunistring-0.9.2	0.399	O+O'+T+T'+S+S'	0.396	C+C'+A+A'+I+I'+W+W'	0.128	N+N'	–	–	√
lightning-2.0.0	0.415	I+I'+O+O'+T+T'+S+S'	0.266	C+C'+A+A'	0.172	N+N'	0.082	W+W'	√
m4-1.4.12	0.376	I+I'+O+O'+T+T'+S+S'	0.327	C+C'+A+A'+W+W'	0.172	N+N'	–	–	√
make-4.0	0.359	C+C'+A+A'+W+W'	0.348	I+I'+O+O'+T+T'+S+S'	0.157	N+N'	–	–	√
mdk-1.0	0.460	C+C'+A+A'+I+I'+T+T'+W+W'	0.265	O+O'+S+S'	0.145	N+N'	–	–	√
mifluz-0.25.0	0.561	C+C'+A+A'+I+I'+T+T'+W+W'	0.264	O+O'+S+S'	0.133	N+N'	–	–	√
mpfr-2.4.0	0.533	C+C'+A+A'+I+I'+T+T'+W+W'	0.277	O+O'+S+S'	0.147	N+N'	–	–	√
mtools-4.0.10	0.420	I+I'+O+O'+T+T'+S+S'	0.207	C+C'+A+A'	0.156	N+N'	0.155	W+W'	√
readline-4.3	0.333	I+I'+O+O'+T+T'+S+S'	0.227	C+C'+A+A'	0.174	N+N'	0.125	W+W'	√
rush-1.5	0.378	I+I'+O+O'+T+T'+S+S'	0.344	C+C'+A+A'+W+W'	0.164	N+N'	–	–	√
uucp-1.07	0.369	C+C'+A+A'+W+W'	0.336	I+I'+O+O'+T+T'+S+S'	0.155	N+N'	–	–	√
vim-6.0	0.375	I+I'+O+O'+T+T'+S+S'	0.241	C+C'+A+A'	0.161	N+N'	0.118	W+W'	√
wget-1.12	0.375	C+C'+A+A'+W+W'	0.360	I+I'+O+O'+T+T'+S+S'	0.160	N+N'	–	–	√
zile-2.3.10	0.364	C+C'+A+A'+I'+W+W'	0.303	I+O+O'+T+T'+S+S'	0.166	N+N'	–	–	×

√: indicate all end-slice-based and metric-slice-based metrics are in the same principal components
×: indicate all end-slice-based and metric-slice-based metrics are in different principal components

based metrics are in the same principal components. "×" indicates that there exist paired metrics in different principal components.

From Table 8, we have the following observations. First, in most data sets, slice-based cohesion metrics are clustered into four distinct orthogonal components, which describe around 88% ~ 98% of the variance in the data. Second, for each of the Coverage, MaxCoverage, Overlap, SBFC, and NHD metrics, end-slice-based and metric-slice-based versions fall into the same component in all the 40 data sets. For the MinCoverage and WFC metrics, end-slice-based and metric-slice-based versions fall into the same component in 39 over all 40 data sets. In 38 over all 40 data sets, end-slice-based and metric-slice-based Tightness fall into the same component. From Table 9, we have the similar observations. The core observation from Tables 8 and 9 is that, for each cohesion metric, there is little difference between the end-slice-based and metric-slice-based versions. In essence, they measure the same cohesion dimensions. Therefore, in practice, for each metric, there is no need to collect both the end-slice-based version and the metric-slice-based version at the same time, as they are redundant.

5. Discussions

In this section, we further discuss conclusions. First, we analyze whether our conclusions will change if the potentially confounding effect of module size is removed. Second, we investigate why end-slice-based and metric-slice-based cohesion metrics generally have similar metric values.

5.1. Will our conclusions change if the potentially confounding effect of module size is excluded?

In this study, we did not take into account the potentially confounding effect of module size on the associations between each pair of slice-based cohesion metrics [25–27]. Therefore, it is unknown whether our conclusions will change if the potentially confounding effect of module size is excluded. In the following, for each slice-based cohesion metric, we use the method proposed by Zhou *et al.* [27] to remove the potentially confounding effect of module size and then rerun the analyses for the cleaned metric. More specifically, for each slice-based cohesion metric c, the corresponding cleaned metric c' is obtained by the following two steps. At the first step, we build a univariate linear regression model with the module size SLOC against c'. At the second step, we subtract the predicted values by the regression model from c to obtain the cleaned metric c'.

Fig. 3 presents the results from correlation analysis for the cleaned end-slice-based and metric-slice-based cohesion metrics. In this figure, all the coefficients have a p-value less than 0.001.

Table 11
PCA results for metrics at statement level after removing the confounding effect of module size.

System	PC1		PC2		PC3		PC4		All
	%Var	Metric	%Var	Metric	%Var	Metric	%Var	Metric	
acm-5.1	0.371	I+I'+O+O'+T+T'+S+S'	0.233	C+C'+A+A'	0.164	N+N'	0.130	W+W'	√
apr-1.0.0	0.371	I+I'+O+O'+T+T'+S+S'	0.252	C+C'+A+A'	0.145	N+N'	0.111	W+W'	√
bash-2.05	0.357	I+I'+O+O'+T+T'+S+S'	0.234	C+C'+A+A'	0.174	N+N'	0.124	W+W'	√
cairo-1.0.0	0.356	I+I'+O+O'+T+T'+S+S'	0.248	C+C'+A+A'	0.150	N+N'	0.137	W+W'	√
cim-5.1	0.401	I+I'+O+O'+T+T'+S+S'	0.257	C+C'+A+A'	0.184	N+N'	0.091	W+W'	√
dico-2.0	0.415	C+C'+A+A'+I+I'+T+T'+W+W'	0.309	O+O'+S+S'	0.164	N+N'	–	–	√
direvent-5.0	0.409	C+C'+A+A'+I+I'+W+W'	0.330	O+O'+T+T'+S+S'	0.167	N+N'	–	–	√
freeipmi-1.2.8	0.401	I+I'+O+O'+T+T'+S+S'	0.214	C+C'+A+A'	0.163	N+N'	0.129	W+W'	√
gawk-4.0.0	0.365	I+I'+O+O'+T+T'+S+S'	0.360	C+C'+A+A'+W+W'	0.163	N+N'	–	–	√
glib-2.18.4	0.462	C+C'+A+A'+I+I'+T+T'+W+W'	0.254	O+O'	0.167	N+N'	0.088	S+S'	√
global-6.2	0.379	I+I'+O+O'+T+T'+S+S'	0.350	C+C'+A+A'+W+W'	0.168	N+N'	–	–	√
glpk-4.40	0.411	C+C'+A+A'+I+I'+T'+W+W'	0.314	O+O'+T+S+S'	0.142	N+N'	–	–	×
gmp-5.0.0	0.485	C+C'+A+A'+I+I'+T+T'+W+W'	0.316	O+O'+S+S'	0.151	N+N'	–	–	√
gnats-4.2.0	0.377	I+I'+O+O'+T+T'+S+S'	0.250	C+C'+A+A'	0.176	N+N'	0.130	W+W'	√
gnu-pw-mgr-1.0	0.455	C+C'+I+I'+O+O'+T+T'+S+S'	0.214	A+A'	0.159	N+N'	0.119	W+W'	√
gnuit-4.9.5	0.418	C+C'+A+A'+I+I'+T+T'+W+W'	0.318	O+O'+S+S'	0.163	N+N'	–	–	√
gsl-1.8	0.529	C+I+I'+O+O'+T+T'+W+W'+S+S'	0.345	C'+A+A'+N+N'	–	–	–	–	×
gstreamer-1.0.1	0.436	C+C'+A+A'+I+I'+T+T'+W+W'	0.244	O+O'	0.158	N+N'	0.105	S+S'	√
idutils-4.2	0.372	I+I'+O+O'+T+T'+S+S'	0.316	C+C'+A+A'+W+W'	0.159	N+N'	–	–	√
libcm-0.1.1	0.330	I+I'+O+O'+T+T'+S+S'	0.318	C+C'+A+A'+W+W'	0.176	N+N'	–	–	√
libgdata-0.8.0	0.400	C+C'+A+A'+W+W'	0.324	I+I'+O+O'+T+T'+S+S'	0.170	N+N'	–	–	√
libgsasl-1.8.0	0.338	I+I'+O+O'+T+T'+S+S'	0.288	C+C'+A+A'	0.192	N+N'	0.136	W+W'	√
libiconv-1.10	0.450	C+C'+A+A'+I+I'+T+T'+W+W'	0.335	O+O'+S+S'	0.165	N+N'	–	–	√
libosip2-2.2.0	0.408	C+C'+A+A'+I+I'+W+W'	0.339	O+O'+T+T'+S+S'	0.153	N+N'	–	–	√
librep-0.16	0.319	C+C'+I+I'+T+T'+W+W'	0.267	O+O'+S+S'	0.198	A+A'	0.167	N+N'	√
libsecret-0.8	0.462	C+C'+A+A'+I+I'+T+T'+W+W'	0.265	O+O'+S+S'	0.157	N+N'	–	–	√
libunistring-0.9.2	0.448	C+C'+A+A'+I+I'+W+W'+N+N'	0.393	O+O'+T+T'+S+S'	–	–	–	–	√
lightning-2.0.0	0.396	I+I'+O+O'+T+T'+S+S'	0.273	C+C'+A+A'	0.191	N+N'	0.080	W+W'	√
m4-1.4.12	0.367	I+I'+O+O'+T+T'+S+S'	0.323	C+C'+A+A'+W+W'	0.184	N+N'	–	–	√
make-4.0	0.353	I+I'+O+O'+T+T'+S+S'	0.349	C+C'+A+A'+W+W'	0.158	N+N'	–	–	√
mdk-1.0	0.363	I+I'+O+O'+T+T'+S+S'	0.325	C+C'+A+A'+W+W'	0.160	N+N'	–	–	√
mifluz-0.25.0	0.561	C+C'+A+A'+I+I'+T+T'+W+W'	0.269	O+O'+S+S'	0.136	N+N'	–	–	√
mpfr-2.4.0	0.526	C+C'+A+A'+I+I'+T+T'+W+W'	0.287	O+O'+S+S'	0.148	N+N'	–	–	√
mtools-4.0.10	0.414	I+I'+O+O'+T+T'+S+S'	0.209	C+C'+A+A'	0.163	N+N'	0.155	W+W'	√
readline-4.3	0.322	I+I'+O+O'+T+T'+S+S'	0.244	C+C'+A+A'	0.177	N+N'	0.122	W+W'	√
rush-1.5	0.370	I+I'+O+O'+T+T'+S+S'	0.343	C+C'+A+A'+W+W'	0.173	N+N'	–	–	√
uucp-1.07	0.387	C+C'+A+A'+W+W'	0.311	I+I'+O+O'+T+T'+S+S'	0.152	N+N'	–	–	√
vim-6.0	0.361	I+I'+O+O'+T+T'+S+S'	0.250	C+C'+A+A'	0.165	N+N'	0.115	W+W'	√
wget-1.12	0.365	I+I'+O+O'+T+T'+S+S'	0.361	C+C'+A+A'+W+W'	0.167	N+N'	–	–	√
zile-2.3.10	0.375	C+C'+A+A'+I+I'+W+W'	0.276	O+O'+T+T'+S	0.172	N+N'	0.099	S'	×

√: indicate all end-slice-based and metric-slice-based metrics are in the same principal components
×: indicate all end-slice-based and metric-slice-based metrics are in different principal components

The results from Fig. 3 reveal that, after removing the potentially confounding effects of module size, each end-slice-based metric is still strongly statistically correlated with the corresponding metric-slice-based version. Tables 10 and 11 summarize the results from PCA with respect to data-token level and statement level after removing the potentially confounding effect of module size. As can be seen, the results in Tables 10 and 11 are very similar to those results in Tables 8 and 9. Overall, after excluding the potentially confounding effect of module size, we still find that end-slice-based and metric-slice-based metrics measure the same cohesion dimensions.

5.2. Why do end-slice-based and metric-slice-based cohesion metrics have similar metric values?

In Section 4, the core observation is that end-slice-based and metric-slice-based cohesion metrics generally have similar metric values. In this section, we investigate the reason behind this observation. Fig. 4 shows for each data set the average percent of statements or data-tokens in a metric slice that are not included in the corresponding end slice. As can be seen, the average percent is less than 10% on almost all the data sets. In particular, for many data sets, the average percent is even less than 5%. In other words, for most modules, there is only a very small difference between their metric slices and end slices. This is exactly the reason that end-slice-based cohesion metrics have similar values to metric-slice-based cohesion metrics in our study.

6. Related work

Since slice-based cohesion metrics have been proposed, there is a debate whether end slice or metric slice should be used. However, little work has been focused on determining which type of slice is preferable for computing slice-based cohesion metrics. Meyers and Binkley [11,15] were the first to conduct a large-scale empirical study of five statement-level end-slice-based cohesion metrics: *Coverage, MaxCoverage, MinCoverage, Tightness*, and *Overlap*. They analyzed the relations between slice-based cohesion metrics and code-size metrics and found that slice-based cohesion metrics provided a unique view of a program. Furthermore, from the results of two longitudinal studies, Meyers and Binkley found that slice-based cohesion metrics were able to quantify the deterioration of a program as it evolved. In their study, Meyers and Binkley used end slice to compute slice-based cohesion metrics. However, they also pointed out that there was a need to examine the utility of end-slice-based and metric-slice-based cohesion metrics in practice. In this sense, our study can be regarded as an extension to their work.

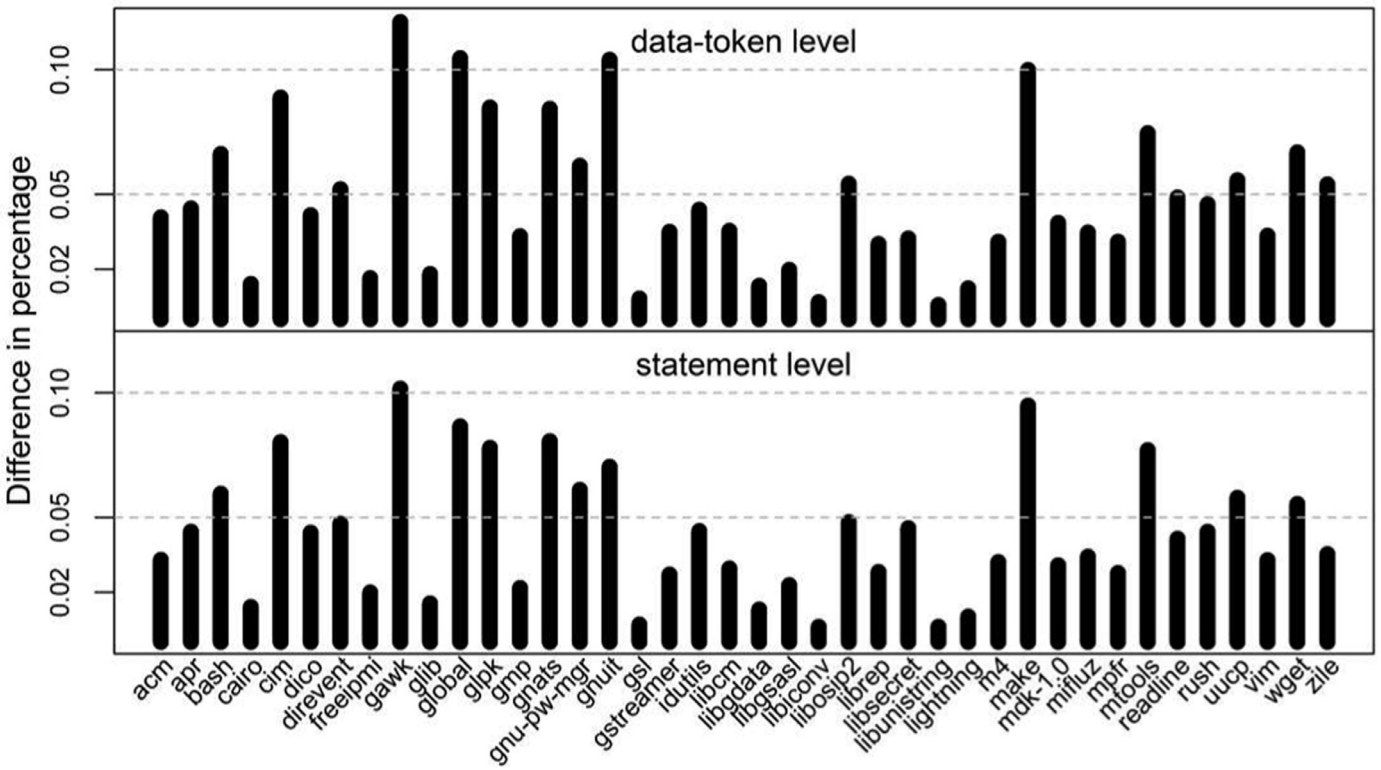

Fig. 4. Difference between a metric slice and the corresponding end slice.

Black *et al.* [18] empirically investigated the ability of two statement-level end-slice-based cohesion metrics, namely *Tightness* and *Overlap*, to distinguish faulty and not-faulty functions. Based on nineteen versions of a small program called Barcode, their results showed that faulty functions tended to have lower *Tightness* and *Overlap* values compared with not-faulty functions. In addition, they also found that, compared with *Overlap, Tightness* had a stronger association with fault-proneness. Dallal [8] proposed a data-token-level end-slice-based cohesion metric called *SBFC*. She used six very small programs developed by students to examine the correlations between *SBFC* and the other three data-token-level metrics (*SFC*, *WFC*, and *A*). She found that there was a strong correlation between *SBFC* and these metrics and hence concluded that SBFC provided a useful alternative to them. Counsell *et al.* [5] proposed a novel end-slice-based cohesion metric called *NHD* and compared it with *Tightness*/*Overlap*. Based on Barcode, they found that *NHD* had a lower correlation to module size (measured by SLOC) compared with *Tightness* and *Overlap*. Yang *et al.* [12] empirically investigated metric-slice-based cohesion metrics in effort-aware fault-proneness prediction. Based on five industrial-size open-source systems, they found that slice-based cohesion metrics were not redundant with respect to the most commonly used code/process metrics. Furthermore, when used together with the baseline code/process metrics, slice-based cohesion metrics can statistically significantly improve the effectiveness of effort-aware fault-proneness prediction. However, all the above literatures were not devoted to the comparison of end-slice-based and metric-slice-based cohesion metrics. In this study, we are the first to perform thorough empirical studies on this subject.

7. Threats to validity

In this section, we analyze the most important threats to the construct, internal, external, and conclusion validity of our study. Construct validity is the degree to which the variables used in a study accurately measure the concept they purport to measure. Internal validity is the degree to which conclusions can be drawn about the causal effect of independent variables on the dependent variables. External validity is the degree to which the findings of a study can be generalized to data not used in that particular study. Conclusion validity is the degree to which the conclusions on the relationships in our data are reasonable.

7.1. Construct validity

The variables used in this study are the end-slice-based and metric-slice-based cohesion metrics. Based on the source code analysis tool Frama-C, we developed two plug-ins called INFERCON and SLIBCOM to compute the end-slice-based and metric-slice-based cohesion metrics. To ensure the metrics were correctly calculated. First, we used INFERCON and SLIBCOM to collect the metric data for a number of functions (developed by ourselves) in which different control flows and data flows were implemented. We found that INFERCON and SLIBCOM always produced reliable results in our test. Second, we randomly examined a small number of C functions in some of these systems and found that the metric data produced by INFERCON and SLIBCOM were reliable. However, in our study, the INFERCOM plugin infers the contract for a called function conservatively. This assumption will possibly cause the slices to be much larger than it would be if using proper points-to analysis. Currently, it is unclear on this impact on the data-flow analysis precision by value analysis. We hope to investigate this impact in the future.

7.2. Internal Validity

In this study, the threat of the internal validity is the unknown effect of excluding the functions that have an “undefined” value. In

our study, when a function cannot be analyzed within 30 minutes or has no output variable reported by Frama-C, the corresponding slice-based cohesion metrics were assigned as “undefined”. These functions with “undefined” metric values were excluded from subsequent analyses (in Section 4). However, according to Table 5, we can find that only a few functions with “undefined” metric values excluded from the subject systems. Therefore, this exclusion should not have a large influence on our conclusions.

7.3. External Validity

Our experiments are based on forty widely used open-source C systems. The first threat to the external validity of this study is that our findings may not be generalized to other systems, especially closed-source systems. The second threat to the external validity of this study is that our findings are restricted to only one language and slice-based cohesion metrics at function level. Researcher has developed many slice-based cohesion metrics at class level [28,29]. We hope that researchers will replicate our study across a wide variety of systems as well as class level slice-based cohesion metrics in the future.

7.4. Conclusion Validity

In this study, our conclusions are derived from the data sets collected from forty widely used open-source C systems. These forty systems are non-trivial software belonging to different problem domains which contain 44,360 modules in total. Therefore, we believe the conclusion validity is acceptable.

8. Conclusion and future work

In this paper, we perform a comparative study to investigate the relationship between end-slice-based and metric-slice-based cohesion metrics. Based on forty open source systems, we find that end-slice-based and metric-slice-based metrics essentially measure the same software cohesion dimensions. This finding suggests that practitioners could choose only end slice or metric slice to compute slice-based cohesion metrics rather than using both of them. Therefore, we suggest using end-slice-based cohesion metrics, as they have a lower cost in data collection. Furthermore, we find that along with the growth of the module size, the average metric values show a decreasing trend. That is to say, small modules tend to be more cohesive than large modules. In the future work, we will replicate our experiments using a wide variety of systems including closed-source systems and systems programmed in other languages.

Acknowledgements

This work is supported by the National Key Basic Research and Development Program of China (2014CB340702), the National Natural Science Foundation of China (61432001, 61272082, 61300051, and 61321491), the Natural Science Foundation of Jiangsu Province (BK20130014), the Climbing Plan of Nanjing University, and the program A for Outstanding PhD candidate of Nanjing University.

References

[1] Y. Press, L.L. Constantine, Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design, 1 edition, Prentice Hall, Englewood Cliffs, N.J, 1979.
[2] M. Page-Jones, The Practical Guide to Structured Systems Design: 2nd Edition, Yourdon Press, Upper Saddle River, NJ, USA, 1988.
[3] L.M. Ott, J.J. Thuss, Slice based metrics for estimating cohesion, in: Software Metrics Symposium, 1993. Proceedings., First International, 1993, pp. 71–81.
[4] J.M. Bieman, L.M. Ott, Measuring functional cohesion, IEEE Trans. Softw. Eng. 20 (8) (1994) 644–657.
[5] S. Counsell, T. Hall, D. Bowes, A theoretical and empirical analysis of three slice-based metrics for cohesion, Int. J. Softw. Eng. Knowl. Eng. 20 (05) (2010) 609–636.
[6] M. Harman, S. Danicic, B. Sivagurunathan, B. Jones, Y. Sivagurunathan, Cohesion metrics, In 8th International Quality Week (San Francisco), May 29th, 1995.
[7] H.D. Longworth, Slice based program metrics Ph.D. thesis, Michigan Technological University, USA, 1985.
[8] J. Al Dallal, Software similarity-based functional cohesion metric, IET Softw. 3 (1) (2009) 46–57.
[9] M. Weiser, Program Slicing, IEEE Trans. Softw. Eng. SE-10 (4) (1984) 352–357.
[10] L.M. Ott, J.M. Bieman, Program slices as an abstraction for cohesion measurement, Inf. Softw. Technol. 40 (11–12) (1998) 691–699.
[11] T.M. Meyers, D. Binkley, An Empirical Study of Slice-based Cohesion and Coupling Metrics, ACM Trans. Softw. Eng. Methodol. 17 (1) (2007) 2:1–2:27.
[12] Y. Yang, Y. Zhou, H. Lu, L. Chen, Z. Chen, B. Xu, H. Leung, Z. Zhang, Are slice-based cohesion metrics actually useful in effort-aware post-release fault--proneness prediction? an empirical study, IEEE Trans. Softw. Eng. 41 (4) (2015) 331–357.
[13] P.D. Green, P.C.R. Lane, A. Rainer, S. Scholz, An Introduction to Slice-Based Cohesion and Coupling Metrics, University of Hertfordshire, 2009 Technical report 488.
[14] K.J. Ottenstein, L.M. Ottenstein, The program dependence graph in a software development environment, in: Proceedings of the First ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, New York, NY, USA, 1984, pp. 177–184.
[15] T.M. Meyers, D. Binkley, Slice-based cohesion metrics and software intervention, in: 11th Working Conference on Reverse Engineering, 2004. Proceedings, 2004, pp. 256–265.
[16] M. Weiser, Program slicing, in: Proceedings of the 5th International Conference on Software Engineering, Piscataway, NJ, USA, 1981, pp. 439–449.
[17] D. Bowes, S. Counsell, T. Hall, Calibrating program slicing metrics for practical use., Presented at the 2009 Testing: Academic and Industrial Conference - Practice and Research Techniques, Windsor , United Kingdom, 2009.
[18] S. Black, S. Counsell, T. Hall, D. Bowes, Fault analysis in OSS based on program slicing metrics, in: Proceedings of the 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, Washington, DC, USA, 2009, pp. 3–10.
[19] L.M. Ott, J.M. Bieman, Effects of software changes on module cohesion, in: Conference on Software Maintenance, 1992. Proceerdings, 1992, pp. 345–353.
[20] J. Al Dallal, Measuring the discriminative power of object-oriented class cohesion metrics, IEEE Trans. Softw. Eng. 37 (6) (2011) 788–804.
[21] J. Al Dallal, Fault prediction and the discriminative powers of connectivity-based object-oriented class cohesion metrics, Inf. Softw. Technol. 54 (4) (2012) 396–416.
[22] P. Cuoq, F. Kirchner, N. Kosmatov, V. Prevosto, J. Signoles, B. Yakobowski, Frama-C: a software analysis perspective, in: Proceedings of the 10th International Conference on Software Engineering and Formal Methods, Berlin, Heidelberg, 2012, pp. 233–247.
[23] R. Ott, M. Longnecker, An introduction to statistical methods and data analysis, Thomson Learning, DUXBURY, USA, 2001.
[24] J. Carpenter, J. Bithell, Bootstrap confidence intervals: when, which, what? A practical guide for medical statisticians, Stat. Med. 19 (9) (2000) 1141–1164.
[25] K. El Emam, S. Benlarbi, N. Goel, S.N. Rai, The confounding effect of class size on the validity of object-oriented metrics, IEEE Trans. Softw. Eng. 27 (7) (2001) 630–650.
[26] Y. Zhou, H. Leung, B. Xu, Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness, IEEE Trans. Softw. Eng. 35 (5) (2009) 607–623.
[27] Y. Zhou, B. Xu, H. Leung, L. Chen, An in-depth study of the potentially confounding effect of class size in fault prediction, ACM Trans. Softw. Eng. Methodol. 23 (1) (2014) 10:1–10:51.
[28] Y. Zhou, J. Wang, J. Zhao, H. Lu, B. Xu, A novel class cohesion measure based on slices, Proceedings of the 10th International Software Metrics Symposium (METRICS 2004), late breaking paper, Chicago, Illinois, USA, 2004. 14–16 September.
[29] Y. Zhou, B. Xu, J. Zhao, H. Yang, ICBMC: an improved cohesion measure for classes, in: Presented at the Proceedings of the 18th International Conference on Software Maintenance, 2002, pp. 44–53.