2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications

An Analysis of the 'Inconclusive' Change Report Category in OSS Assisted by a Program Slicing Metric

S. Counsell, T. Hall and E. Nasseri
Department of Computing
Brunel University, Uxbridge, UK
steve.counsell@brunel.ac.uk

D. Bowes
Dept. of Computer Science
University of Hertfordshire, Hatfield, UK
d.h.bowes@herts.ac.uk

Abstract

In this paper, we investigate the Barcode open-source system (OSS) *using one of Weiser's original slice-based metrics* (*Tightness*) *as a basis. In previous work, low numerical values of this slice-based metric were found to indicate fault-free* (*as opposed to fault-prone*) *functions. In the same work, we deliberately excluded from our analysis a category comprising 221 of the 775 observations representing 'inconclusive' log reports extracted from the OSS change logs. These represented OSS change log descriptions where it was not entirely clear whether a fault had occurred or not in a function and, for that reason, could not reasonably be incorporated into our analysis. In this paper we present a methodology through which we can draw conclusions about that category of report.*

1. Introduction

The area of program slicing has developed into a software engineering topic spawning a range of research studies in a number of disciplines [4, 5, 6, 14, 15, 16, 21]. Program slicing has also been used as a basis for measuring software cohesion and a notable set of slice-based metrics for cohesion were first proposed by Weiser in [20]. In previous work by the authors [7], we explored whether two slice-based metrics (Tightness and Overlap [20]) could tell us anything about the propensity of functions to be fault-prone or not. The Tightness metric showed some promise in its ability to discriminate between fault-free and fault-prone functions; we analyzed multiple versions of the Barcode open-source system (OSS) on the basis of those two metrics and found that low values of the Tightness metric were indicative of fault-free functions. In the same study however, we were obliged to omit 221 of the 775 reports manually extracted from Barcode logs on the basis that they were deemed 'inconclusive'. In other words, they suggested that a fault *may* have occurred in the function's code, but we could not say with any *certainty* that this was actually the case. In this paper, we provide an analysis of the 'inconclusive' category and pose the question: can statistical analysis and the earlier finding related to Tightness help us assess whether this category was more indicative of fault-prone rather than fault-free functions?

2. Preliminaries

2.1. Metric definition

The metric that we explore in this paper (Tightness) was originally proposed by Weiser [20] and in this paper we use the same formal definition of the metric. Before defining that metric, and in common with our earlier work [7], we first describe the necessary formal underpinnings of a slice's components proposed by Ott and Thuss [17, 18] (and which we adopt in this paper). We denote a set of variables used by a function K as V_K and V_Z as the subset of V_K representing output (return), input, global and printf variables (i.e. variables used in printf statements. K represents a program 'function', defined as a unit of code under consideration. We further note that in the OO paradigm, this would equate to a class, the level at which OO cohesion metrics have tended to be applied in past studies [1, 8, 9, 11]. We denote a slice SL_i as that obtained for $v_i \in V_Z$ and SL_{int} as the intersection of SL_i over all $v_i \in V_Z$.

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

Tightness measures the extent of interaction between the slices of a function.

2.2. Metric/fault extraction

The CodeSurfer tool [10] was used to extract the Tightness metric from multiple versions of Barcode, an OSS written in C for processing barcode data. Nineteen versions of Barcode were studied as part of our analysis. Fault data was extracted manually using the on-line report logs of the system by two researchers. Henceforward, we describe functions that contain at least one fault in any single version as 'fault-prone' and those that contained zero faults (in any single version up until the current date) as 'fault-free'. Initially, and as reported in [7], the dataset was partitioned into just these two categories. However, in that previous work, 221 of the 775 functions had to be classed as 'inconclusive'.

3. Tightness metric

The focus of the paper is to explore one key research question, based on the three categories extracted from Barcode: *Do the characteristics of the inconclusive category for the Tightness metric have a greater similarity with fault-prone or fault-free functions* (*or neither*)? In the previous study [7], low values of Tightness were found to be indicative of fault-free functions. Preliminary scrutiny of the inconclusive data revealed a disproportionately large number of high Tightness values in that category. On that basis, there is reason to suspect that the inconclusive category *tends* more towards fault-prone functions. In this paper, we explore that possibility further.

3.1. Summary data

Table 1 contains the summary data (number of functions (N) in each category, mean, maximum, minimum, standard deviation (SD) and median) for the Tightness metric for all functions in the three categories (fault-prone, fault-free and inconclusive).

Table 1. Summary data for Tightness

Tightness	N	Mean	Max	Min	SD	Med.
Fault-prone	372	*0.38*	1.00	0	0.37	*0.28*
Fault-free	150	*0.32*	0.99	0	0.32	*0.21*
Inconclusive	221	*0.36*	0.99	0.02	0.28	*0.33*

A noticeable trend from Table 1 is the relative closeness of the inconclusive values to the fault-prone values. The mean for inconclusive values is 0.36, which is closer to that of the mean for the set of fault-prone functions by 0.02. The most revealing statistic is that for the median for inconclusive values (it is 0.12 above that of fault-free functions, but only 0.05 above that of the fault-prone function values). Tentatively, it would seem that values in the inconclusive category are more similar to fault-prone than fault-free functions.

3.2. Inconclusive values and Tightness

Figure 1 shows the values of the Tightness metric for inconclusive functions (in the form of a scatter plot). The relatively fewer values in the range 0-6 – 1.00 compared with those in the lower ranges (0-0.599) is evident from the figure. Small 'pockets' of values are also noticeable in the 0.2-0.4 range. Table 2 shows the breakdown of the frequencies and the percentage of values in each category.

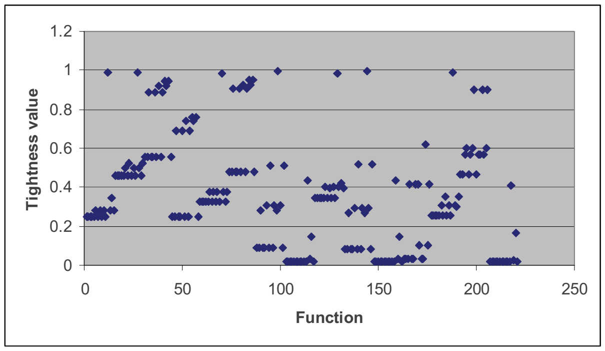

Figure 1. Tightness values (inconclusive functions)

Table 2 shows that the majority of Tightness values are in the lower range of the Tightness metric. In fact, inconclusive values tend to occur in the range 0.2-0.4. This is where the similarity between fault-prone and inconclusive functions is found to be strongest (median values in Table 1 are 0.28 and 0.33, respectively). The low number of overall values in the range 0.60-1.00 in the three categories is reflective of the fact that in most industrial systems intersection of variable usage is invariably small. Barcode is no exception in this sense. By definition, if the SL_{int} value is low, then the Tightness metric value will be low also.

Table 2. Frequency of Tightness values (inconclusive)

Range/Category	Total	0 – 0.199	0.2– 0.399	0.4– 0.599	0.6-0.799	0.8-1.00
Fault-prone	372	172 (46.24%)	65 (17.47%)	55 (14.78%)	26 (6.99%)	54 (14.52%)
Fault-free	150	63 (42.00%)	24 (16.0%)	19 (12.67%)	2 (1.33%)	42 (28.0%)
Inconclusive	221	65 (29.41%)	69 (31.22%)	52 (23.53%)	11 (4.98%)	24 (10.86%)

In the spirit of the discussion in the previous section, sound software engineering practice for achieving high cohesion and low fault-proneness would suggest a high intersection of variables (i.e., all variables use each other frequently). This is certainly in keeping with the contemporary view of high cohesion according to both the LCOM and CAMC metrics [1, 9, 12]. It is of note therefore that the highest percentage of values reflecting this characteristic of functions (0.8-1.00 category from Table 2) *does actually belong to the fault-free functions* (42 functions, representing 28.0%). That said, we believe that there is a mismatch between theory and practice. We suggest that what the contemporary view of cohesion does not take into consideration is that high interaction of variables which contributes positively to the ideal cohesion value in the cases of LCOM and CAMC, also increases the complexity of a function, the need to untangle the logic of such a function and hence increase the potential for faults. High cohesion in a function according to current thinking and definitions does not necessarily mean that the function will be less liable to faults. It might actually mean the reverse.

3.3 Fault-prone versus fault-free

Figure 2 shows the relationship between fault-free and inconclusive functions and shows very little relationship between the two sets of Tightness values. The R^2 (Pearson's parametric coefficient) value is just 0.02 (rounded).

Figure 3 shows the scatter plot for the fault-prone functions and shows a stronger relationship between the inconclusive and fault-free set of functions. The Pearson's R^2 value in this case is 0.14, significant at the 1% level (two tailed test) [19]. The result tentatively supports the view that the set of inconclusive functions has more similarity with fault-prone functions than it does with fault-free functions.

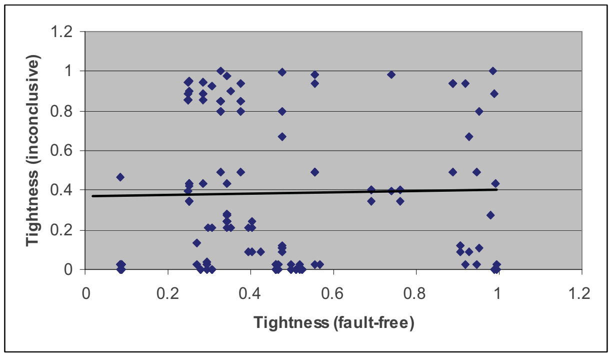

Figure 2. Inconclusive versus fault-free (Tightness)

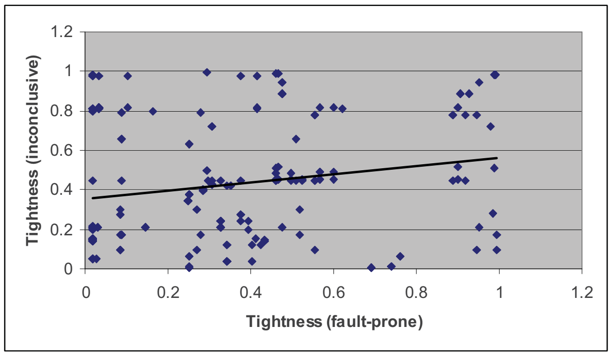

Figure 3. Inconclusive vs. fault-prone (Tightness)

3.4. Statistical support

To support our analysis, we correlated the category of inconclusive values against fault-prone and then fault-free functions using Spearman's and Kendall's, non-parametric, correlation coefficients. The correlation between fault-prone and Spearman's rank correlation coefficient was 0.22 (significant at the 1% level); Kendall's coefficient was 0.17, also significant at the 1% level. On the other hand, for the correlation between fault-free and inconclusive values, Spearman's coefficient was only -0.03 and Kendall's -0.03, neither of which were significant. Correlation therefore supports the view that the inconclusive category is more strongly related to the fault-prone category and what is more correlated significantly.

4. Conclusions and future work

The relevance to a developer of the research presented is that, based on the evidence, they should minimise the number of variables (but more importantly the interactions) in a function so as to obtain simple functions; by doing that, (according to our analysis) we believe this will lead to low values of the Tightness metric and less likelihood of a fault in that function. They should accompany as few variables as possible with keeping their functions small in size.

Future work will focus on extending the empirical study to other systems and to compare the results with other cohesion metrics for which have the data (e.g., the NHD of Counsell et al. [11] and the remaining metrics from the set originally proposed by Weiser [20]). Finally, it would be interesting to see how the values of the Tightness metric change in the presence of an active re-engineering or refactoring strategy [13].

Acknowledgements

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

References

[1] Bansiya, J., Etzkorn, L., Davis, C., and Li, W. A class cohesion metric for object-oriented designs. Journal of Object-Oriented Programming 11(8), pp. 47-52, 1999.

[2] Bieman, J., and Ott, L. Measuring functional cohesion. IEEE Trans. on Software Eng. 20, 8 (1994), pp. 644-657.

[3] Binkley, D. Gold, N. and Harman, M. An empirical study of static program slice size. ACM Trans. Software Engineering Methodology (TOSEM) 16(2):1-32, 2007.

[4] Binkley, D., Harman, M., and Krinke, J., Empirical study of optimization techniques for massive slicing. ACM Trans. Program. Lang. Syst. 30(1): (2007)

[5] Binkley D and Harman M., Locating dependence clusters and dependence pollution, IEEE International Conf. on Soft. Maintenance, Budapest, Sept. 2005 pages 177-186.

[6] Binkley, D., Harman, M., Raszewski, I., and Smith, C. An empirical study of amorphous slicing as a program comprehension tool. Proc. of the Intl. Workshop on Program Comprehension, Limerick, Ireland, pp. 161-170, 2000.

[7] Black, S. Counsell, S, Hall, T, Bowes, D, Fault Analysis in OSS Based on Program Slicing Metrics. EUROMICRO-SEAA 2009, pages 3-10, Patras, Greece.

[8] Briand, L., Daly, J., and Wust, J. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering Journal 3(1), 65-117, 1998.

[9] Chidamber, S., and Kemerer, C. A metrics suite for object oriented design. IEEE Trans. on Software Engineering 20(6) (1994), 467-493.

[10] www.grammatech.com/products/codesurfer/

[11] Counsell, S., Swift. S. and Crampton J. The Interpretation and Utility of Three Cohesion Metrics for Object-Oriented Design. ACM Transactions on Software Engineering and Methodology, 15(2):123 – 149, 2006.

[12] Counsell, S., Bowes D., and Hall T., Evolutionary Cohesion Metrics: The Empirical Contradiction. Proceedings of The Psychology of Programming Interest Group (PPIG), Open University, January 2009.

[13] Fowler, M. Refactoring (Improving the Design of Existing Code). Addison Wesley, 1999.

[14] Horwitz, S, Reps, T. and Binkley, D., Interprocedural Slicing Using Dependence Graphs. ACM Transactions on Programming Language and Systems, 12(1): 26-60, 1990.

[15] Meyers, T and Binkley, D. Slice-based Cohesion Metrics and Software Intervention, Proc. Working Conf. on Reverse Engineering, Delft, Netherlands, pages 256-265.

[16] Meyers, T. and Binkley, D. An empirical study of slice-based cohesion and coupling metrics. ACM Trans. on Software Engineering and Methodology, 17(1), 2007.

[17] Ott L, Thuss J., (1993) Slice based metrics for estimating cohesion; Proc Software Metrics, 71–81, Baltimore, US.

[18] Ott L. and Thuss, J., The relationship between slices and module cohesion. Proceedings of International Conference on Software Engineering, Pittsburgh, US, 1989, pages 198-204.

[19] Snedecor, G., and Cochran, W. Statistical Methods, 8th ed. Iowa State University Press, Ames, Iowa, 1989.

[20] Weiser, M. Program slicing. Proceedings Int. Conf on Soft Eng., San Diego, 1981. IEEE Press, pp. 439-449.

[21] Weiser M (1982) Programmers use slices when debugging, Comm. of the ACM, 25(7):446-452, July 1982