Using Program Slicing to Identify Faults in Software:

Sue Black[1], Steve Counsell[2], Tracy Hall[3], Paul Wernick[3],

[1] Centre for Systems and Software Engineering, London South Bank University, 103 Borough Road, London SE1 0AA, UK.,
blackse@lsbu.ac.uk
[2] Department of Information Systems and Computing, Brunel University, Uxbridge, Middlesex UB8 3PH, UK.
steve.counsell@brunel.ac.uk
[3] School of Computer Science, University of Hertfordshire, Hatfield, Herts +44 (0)1707 284 782,
{t.hall,p.d.wernick}@herts.ac.uk

Abstract. This study explores the relationship between program slices and faults. The aim is to investigate whether the characteristics of program slices can be used to identify fault-prone software components. Slicing metrics and dependence clusters are used to characterise the slicing profile of a software component, then the relationship between the slicing profile of the component and the faults in that component are then analysed. Faults can increase the likelihood of a system becoming unstable causing problems for the development and evolution of the system. Identifying faultprone components is difficult and reliable predictors of fault-proneness not easily identifiable. Program slicing is an established software engineering technique for the detection and correction of specific faults. An investigation is carried out into whether the use of program slicing can be extended as a reliable tool to predict fault-prone software components. Preliminary results are promising suggesting that slicing may offer valuable insights into fault-proneness.

Categories and Subject Descriptors
D.2.8 [**Software Engineering**]: Metrics– *product metrics*

General Terms
Measurement

Keywords
Program slicing, slicing metrics, fault proneness, software quality.

1. PROBLEM OVERVIEW AND RELATED WORK

Faults in software systems continue to be a major problem. Many systems are delivered to users with excessive faults. Newspaper headlines regularly report faults in major pieces of expensive software (eg Microsoft's search engine, Search Beta, Computer Weekly, 16/11/04). The source of these faults is many and varied, however, technical errors in code remain a significant class of faults. This is despite a huge amount of development effort going into fault reduction in terms of quality control and testing. It has long been recognised that seeking out fault-prone parts of the system and targeting those parts for increased quality control and testing is an effective approach to fault reduction.

A limited amount of valuable work in that area has been carried out previously [15]. Despite this it is difficult to identify a reliable approach to identifying fault-prone software components. A variety of fault prediction models and techniques have been developed, which Fenton and Neil review [4]. They report that previous approaches are primarily based on complexity and size. In particular previous work analyses complexity data using metrics such as Halstead [5] and McCabe [8], and size data using metrics such as KLOC and Function Points [1]. We argue that our novel application of program slicing may make a significant contribution to improving the reliability of detecting fault-prone software components. Program slicing offers a rich insight into the semantic complexities of the structure of a component of code. Such complexity is reported as a likely source of faults [15] meaning that the characteristics of program slices offer a more direct, and potentially more accurate understanding of where faults are likely to occur.

Program slicing is a well-recognised technique [12,11,14] that is used mainly at source code level to highlight code statements that impact upon other statements. It is an analysis technique that extracts all statements relevant to the computation of a given variable. The original motivation for program slicing was to locate faults during debugging. The idea was that the slice would contain the fault, but would not contain lines of code that could not have caused that fault. Computing program slices reduces the relevant part of a program for easier identification of faults. Much research has been carried out using program slicing to identify faults contained within code by reducing the amount of code that needs to be inspected into relevant slices [7]. These slices are then analysed using a static analysis tool (such as Codesurfer) to find faults.

Two recent developments in program slicing underpin our ability to apply program slicing to fault prediction:

1. Meyers and Binkley's [9] large scale collection of metrics data which characterises program slices in open source systems. Meyers and Binkley applied and validated metrics initially proposed by Weiser [13] and extended by Ott and Thuss [10]. The emergence of reliable automatic code analysis tools for collecting large scale data has now made this progress possible.
2. Binkley and Harman's [3] use of program slicing to compute dependence clusters. Dependence clusters are formally defined as the solution to a reachability problem over a program's System Dependence Graph [6]. Slices produced from a program are plotted on a graph in monotonically increasing order of slice size, giving a graphic illustration of the presence of clusters as sheer-drop cliff faces.

We use these two approaches to generate a profile of the program slices in a software component. We then relate this profile to the faults in that software component.

2. BRIEF DESCRIPTION OF PRELIMINARY RESULTS

There are two parts to this exploratory study:

1. **Formulate hypotheses of the relationship between slice characteristics and faults.** This part of the study is complete and reported in Appendix One.
2. **Test our hypotheses on samples of software components.** For this exploratory study we use Open Source Systems (OSS) to initially test our hypotheses. We present analysis based on three sets of data collected from the Barcode open source system (Barcode is available from the Freshmeat OSS portal). Barcode is a small system with fault data available for 9 minor releases. Appendix 2 shows that the 3 data sets collected for Barcode are:
   - Slicing metrics data
   - Dependence cluster data

- Fault data.

The 3 data sets for Barcode in Appendix 2 show that it is possible to collect and analyse data to explore our hypothesis. This small system does not reveal major relationships between the characteristics of the slices (Figure 1) and the faults reported (Figure 2). Barcode is a small OSS with minor faults being reported, we would, therefore not necessarily expect significant relationships between the variables to emerge. Figures 1 and 2 do however confirm a loose relationship between system size and faults. An interesting result is the potential relationship between dependence clusters, faults and system size. The data presented in Figure 3 relates only to version .98 of Barcode. Generating dependence clusters for previous versions may give insights into the growth of interdependence in the system. Such interdependence growth is likely to correlate with increased faults. These results demonstrate that appropriate data can be mined from an OSS portal and that analysis of larger OSS systems can now be carried out.

3. DISCUSSION OF IMPACT OF RESULTS AND TENTATIVE FUTURE INVESTIGATIONS

This work is at a preliminary stage and requires considerably more empirical analysis to generate reliable conclusions. In the short term we plan to extend our analysis of OSS systems. Assuming the OSS results continue to be promising we then intend to investigate the relationship between slices and faults in an industrial setting with a collaborator.

The results of this larger scale work could make a significant contribution to reducing faults in software. If we can find a relationship between the characteristics of program slices and faults this will enable developers to effectively target improvement activities towards those parts of the software. This means that more faults will be detected and so quality will be improved. It also means that an effort effective, and therefore cost, effective approach is possible

4. ACKNOWLEDGMENTS

Sue Black would like to thank Professor Mark Harman and King’s College London for hosting her sabbatical and London South Bank University for the ‘Promising Researcher’ funding.

5. REFERENCES

[1] Albrecht, Allan J., "Measuring Application Development Productivity", Proceedings SHARE/GUIDE IBM Applications Development Symposium, Monterey, CA., Oct 14-17, 1979.

[2] Black, S. E., "Computing ripple effect for software maintenance", Journal of Software Maintenance and Evolution: Research and Practice 2001;13:263-279.

[3] Binkley D and Harman M, “Locating dependence clusters and dependence pollution”, IEEE International Conference on Software Maintenance, Budapest, Hungary, Sept 26th -292005.

[4] Fenton, N. E. and Neil, M. 1999. A Critique of Software Defect Prediction Models. *IEEE Trans. Softw. Eng.* 25, 5 (Sep. 1999), 675-689

[5] Halstead, M.H.: *Elements of Software Science*. Prentice-Hall, Inc., New York, 1977

[6] Horwitz, S., Reps, T., Binkley, D. 1988. Interprocedural slicing using dependence graphs. In *Proceedings of the ACM SIGPLAN 1988 Conference on Programming Language Design and Implementation* (Atlanta, Georgia, United States, June 20 24, 1988). ACM Press, New York, NY, 35-46.

[7] Kusumoto, S., Nishimatsu, A., Nishie, K., and Inoue, K. 2002. Experimental Evaluation of Program Slicing for Fault Localization. *Empirical Softw. Engg.* 7, 1 (Mar. 2002), 49-76.

[8] McCabe(1976) : Complexity Measure, IEEE Transactions on Software Engineering, Volume 2, No 4, pp 308-320, Dec 1976

[9] Meyers T, Binkley D. (2004) *Slice-Based Cohesion Metrics and Software Intervention* Proceedings of the IEEE Eleventh Working conference on Reverse Engineering, Delft University, the Netherlands 9-12 November 2004

[10] Ott L, Thuss J (1993) Slice based metrics for estimating cohesion; Proc. First intern Software Metrics Symposium, 71–81

[11] Tip, F. 1994 *A Survey of Program Slicing Techniques.*. Technical Report. UMI Order Number: CS-R9438., CWI (Centre for Mathematics and Computer Science).

[12] Weiser, M. 1981. Program slicing. In *Proceedings of the 5th international Conference on Software Engineering* (San Diego, California, United States, March 09 -12, 1981). IEEE Press

[13] Weiser M (1982) Programmers use slices when debugging, Communications of the ACM, v.25 n.7, p.446-452, July 1982

[14] Xu, B., Qian, J., Zhang, X., Wu, Z., and Chen, L. 2005. A brief survey of program slicing. *SIGSOFT Softw. Eng. Notes* 30, 2 (Mar. 2005), 1-36.

[15] Yu P, Systa T, Muller H. "Predicting Fault-Proneness using OO Metrics: An Industrial Case Study," *csmr,* p. 0099, Sixth European Conference on Software Maintenance and Reengineering, 2002.

Appendix One. Hypotheses

Hypothesis 1: short program slices have fewer faults

It is likely that a long section of program logic relating to one variable is more complex than a shorter run of logic. More lines of code will need to be written and the relationships between the lines will become more complex. Programs with short slices are likely to have fewer errors than those with longer slices.

This hypothesis relates to the *coverage* slicing metric [1]:

Coverage: "… compares the length of slices to the length of the entire program. Coverage might be expressed as the ratio of mean Barcode version slice length to program length. A low coverage value [indicates] a long program with many short slices. We expect that high values for coverage correlate to longer runs of code, and thus to higher expected fault rates.

Hypothesis 2: code common to many program slices has fewer faults

Code common to many parts of a program is likely to have been tested by the developers, and executed by the system's users, more frequently than code that is only used in one part of the program. The many examinations and readings of the common code during development, during testing as well as in normal operation of the system in use, is likely to identify faults more quickly than is the case for code common to many slices. This hypothesis relates to the following slicing metrics [1]:

overlap: "the mean of the ratios of non-unique to unique statements in each slice. A high overlap might indicate very interdependent code." This measures the number of lines of code that appear only in that slice and the number of lines of code that also appear in other slices. The ratio of these two values is calculated. [1]

We expect high overlap to correlate to lower fault rates. However it is also possible that a high value might indicate complex, interleaved code that might actually be more fault-prone.

tightness: "measures the number of statements which are in every slice, expressed as a ratio over the total program length. The presence of relatively high tightness might indicate that all the slices in a subroutine really belonged together because they all shared certain activities."

Again, we expect high tightness to correlate to lower fault rates.

Hypothesis 3: cliff faces in dependence clusters indicate faults

Faults are more likely to occur in source code that is highly interdependent. The presence of a set of statements in a program that are all mutually interdependent should be viewed with caution; dependence clusters in code indicate that there may be ripple effects [2] if that code is changed. A dependence cluster is deemed to be significant if it is over a 10% threshold i.e. 10% or more of a program's slices are in the cluster, this shows up as a cliff drop on a Monotonic Size-slice Graph [3]. A significant dependence cluster is therefore representative of fault-proneness.

[1] Descriptions for slicing metrics are taken from Weiser [12]

Appendix Two. Data Collected for Barcode

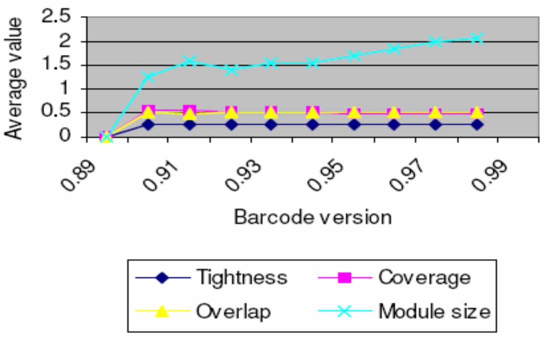

Figure 1. Slicing data for Barcode
(Reproduced from [8])

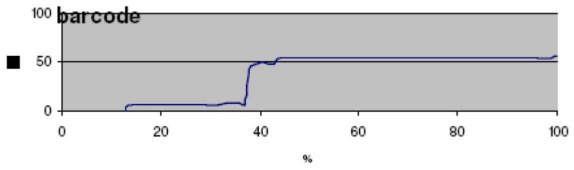

Figure 2. Dependence clusters for Barcode v.98
(Reproduced from [2])

This graph shows all slices contained within the barcode program plotted by increasing size. The cliff face shows that approximately 40% of all the slices are the same size.

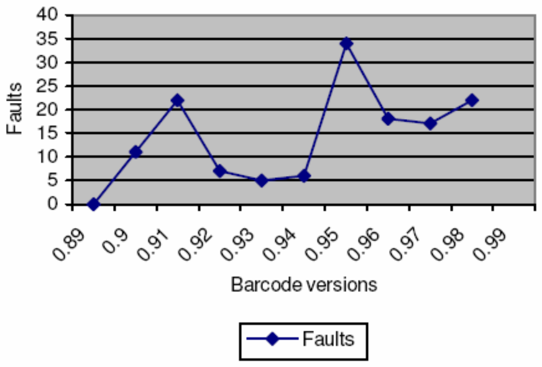

Figure 3. Faults reported for Barcode
(data from the fault log for Barcode on the Freshmeat OSS Portal