Using Data Mining to Assess Software Reliability

Christos Tjortjis and Paul Layzell
Department of Computation, UMIST
P.O. Box 88, Manchester, M60 1QD, UK
Email: {christos, pjl}@co.umist.ac.uk

Index terms- Reliability assessment tools, Software Maintenance, Data Mining, Cluster analysis, Association rules.

Abstract

The paper investigates the applicability of data mining in software reliability assessment and maintenance. The proposed methodology comprises three steps. First the input models are defined by selecting parts of the source code, such as functions, routines and variables, to populate a database. Then Clustering is applied to identify sub-sets of source code that are grouped together according to custom-made similarity metrics. Finally Association rules are used to establish inter-group and intra-group relationships. Experimental results show that the methodology can assess modularity, detect complexity and predict the impact of changes.

1. Introduction

Assessing software reliability and testability is a major challenge related to testing and maintenance. Testability is a measurement of structural complexity, which has no operational definition and can neither have an absolute value nor can it reveal every weakness in reliability [10]. However, it can be used to assess the impact of changes. High testability makes the validation phase more efficient and improves maintenance and comprehension.

Cognitive complexity is influenced by structural properties, such as coupling and cohesion. Systems composed of highly coupled classes are more fault-prone, and hard to comprehend and maintain. Coupling Between Object Classes is a measure which depends on classes' usage of methods or attributes that belong to other classes, where uses can mean as a member, method local variable, or parameter type [1]. Another way for predicting fault-prone modules is to exploit metrics using data mining[1] [5].

Data mining is suitable to support software reliability assessment, as it achieves results for large collections of data even when limited background knowledge is available. It has also been applied to get a better understanding of source code. Examples include identification of data cohesive subsystems [8], recovery and maintenance of software system structures [7] and architectural design recovery [9].

2. Data mining for assessing software reliability and facilitating maintenance

2.1. Aims and objectives

This work aims at facilitating software reliability assessment, maintenance and comprehension by identifying fault-prone modules and predicting the impact of changes. Objectives include the definition of data representation models for source code, and the application of appropriate data mining techniques. Mining data derived from code, rather than metrics can be used to explore module complexity and interrelationships, in a manner similar to methods based on coupling and cohesion.

2.2. Input models

Defining input models involved representing programs as a number of entities each consisting of several

[1] *Data Mining* means applying data analysis and discovery algorithms that produce a particular enumeration of patterns over the data [3]. It incorporates several techniques, such as *Clustering*, which partitions a data set into mutually exclusive groups and *Association rules* which return relationships among sets of items where presence of an item in a record implies the presence of others in the same record.

attributes. Two models were used to extract data from code and populate a database suitable for data mining. One model for C/C++, where entities are functions and attributes are defined according to the use and types of parameters and variables, and the types of returned values. The model for COBOL caters for a medium level involving paragraphs as entities, and a low level, where entities are merely lines of code. Attributes in both levels are binary depending on the presence of user-defined and language-defined identifiers.

2.3. Clustering

Data Mining Code Clustering (*DMCC*) is the approach taken for grouping C/C++ functions, based on their similarity, into clusters, which represent subsystems. A prototype tool was developed using an agglomerative hierarchical clustering algorithm [4] and custom-made similarity metrics founded on the association coefficient paradigm. The tool utilised additional information about interrelationships amongst attributes. It was evaluated using systems of various sizes, of up to 18 modules and 118 functions. Experiments resulted in multi-layered high-level abstractions of given systems, as a number of subsystems consisting of "similar" functions. Interrelationships amongst functions were identified likewise.

DMCC produces systems' overviews, which aid comprehension. Grouping program components into subsystems reduces the perceived complexity thus facilitating maintenance. Complexity can be detected by identifying subsystems which consist of comparatively large number of functions. Large, complex and strongly interrelated subsystems are likely to be fault-prone.

Software reliability assessment is supported by automatically deriving a meaningful decomposition of source code into several subsystems, identifying the interfaces connecting them, and determining the role each subsystem plays in performing a service. This can further help to modify existing code in a manner consistent with the original structure and understand the overall impact of such modifications. For any changes, especially related to parameter usage within the body of a function, the software engineer should consider the possibility of affecting other "similar" functions. This supports fast code modification risk assessment, even prior to regression tests.

DMCC can also be used to improve systems cohesion and coherence by increasing modularity. This improves reliability and can be done in two ways. First by relocating functions into modules where they more "naturally" belong, i.e. amongst similar functions. Second, by adjusting the processing performed within functions to better reflect the functionality designed to be encapsulated within.

2.4. Association rules

Mining *association rules* is suitable for binary attributes derived from COBOL programs. A parsing tool using code as input was developed to populate a database. Another tool based on *Apriori*, a well-established algorithm, was developed to scan the database and derive association rules among attributes [2]. The tool groups paragraphs together if they contain a user-defined number of strong rules. This grouping is an alternative to clustering paragraphs according to their "similarity", analogously to *DMCC*.

Experiments with COBOL programs of up to 1000 lines of code highlighted the potential of the method in identifying groups of variables and/or reserved words which tend to appear in the same module, thus implying that they are interrelated. The method also identifies programming styles, by exposing patterns related to the presence of variables and reserved words in paragraphs. Finally, grouping paragraphs gives an insight into modularity thus facilitating the prediction of the impact of changes and software reliability assessment.

Irregularities in the observed patterns suggest that some parts of the program are exceptional, and may require further testing. Rules and patterns obviously need expert examination, interpretation and validation. Ongoing work involves expert consultation on the significance of this methodology and the validity of the patterns derived by Clustering and Association rules [6]. Further experimentation for fine-tuning of the tools is also required to optimise performance.

References

[1] R. Agrawal and R. Srikant, 'Fast Algorithms for Mining Association Rules', *Proc. 20th Int'l Conf. Very Large DataBases* (VLDB 94), 1994, pp. 487-499.

[2] S. Benlarbi, N. Goel, K. El Emam and S. Rai 'Thresholds for OO Measures', *Proc. 11th Int'l Symposium Software Reliability Engineering* (ISSRE 2000), IEEE Comp. Soc. Press, 2000, pp. 24-37.

[3] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, 'From Data Mining to Knowledge Discovery: an Overview', *Advances*

[2] Each rule is characterised by its *confidence*, i.e. the proportion of times the rule is correct, and its *support*, i.e. the proportion of times the rule applies. A rule with high confidence and support is *strong* [1].

in Knowledge Discovery and Data Mining, AAAI Press, 1996, pp. 1-34.

[4] K. Jain and R. C. Dubes, *Algorithms for Clustering Data*, Prentice-Hall, 1988.

[5] T. Khoshgoftaar, E. Allen and R. Shan, 'Improving Tree-based Models of Software Quality with Principal Components Analysis', *Proc. 11th Int'l Symposium Software Reliability Engineering* (ISSRE 2000), IEEE Comp. Soc. Press, 2000, pp. 198-209.

[6] M. Li, C. Smidts and R. Brill, 'Ranking Software Engineering Measures Related to Reliability Using Expert Opinion', *Proc. 11th Int'l Symposium Software Reliability Engineering* (ISSRE 2000), IEEE Comp. Soc. Press, 2000, pp. 246-258.

[7] S. Mancoridis, B.S. Mitchell, Y. Chen and E.R. Gansner, 'Bunch: A Clustering Tool for the Recovery and Maintenance of Software System Structures', *Proc. Int'l Conf. Software Maintenance* (ICSM 99), IEEE Comp. Soc. Press, 1998, pp. 50-59.

[8] C.M. de Oca and D.L Carver, 'Identification of Data Cohesive Subsystems Using Data Mining Techniques', *Proc. Int'l Conf. Software Maintenance* (ICSM 98), IEEE Comp. Soc. Press, 1998, pp.16-23.

[9] K. Sartipi, K. Kontogiannis and F. Mavaddat, 'Architectural Design Recovery Using Data Mining Techniques', *Proc. 2nd European Working Conf. Software Maintenance Reengineering* (CSMR 2000), IEEE Comp. Soc. Press, 2000, pp. 129-140.

[10] Y. Le Traon, F. Oaubdesselam and C. Robach 'Analyzing Testability on Data Flow Designs', *Proc. 11th Int'l Symposium Software Reliability Engineering* (ISSRE 2000), IEEE Comp. Soc. Press, 2000, pp. 168-173