A Method for Legacy Systems Maintenance by Mining Data Extracted from Source Code

Kai Chen	Christos Tjortjis	Paul Layzell
Dept. of Computation, UMIST	*Dept. of Computation, UMIST*	*Dept. of Computation, UMIST*
P.O. Box 88, Manchester,	*P.O. Box 88, Manchester,*	*P.O. Box 88, Manchester,*
M60 1QD, UK	*M60 1QD, UK*	*M60 1QD, UK*
K.Chen@student.umist.ac.uk	christos@co.umist.ac.uk	pjl@co.umist.ac.uk

Index terms- Legacy Systems Maintenance, Program Comprehension, Data Mining Tools, Data Preparation.

Abstract

This paper proposes a new method for understanding and maintaining legacy software systems. The method is based on the use of data mining for extracting interrelationships, patterns and groupings of code elements ranging from variables up to modules.

Data mining techniques have been previously used for producing high-level system organizations of source code and legacy systems remodularization. Clustering and association rules were used to get an overview of legacy systems, when attempting understanding, maintenance or re-engineering.

However, all previous approaches have addressed systems at a high level of files, programs and modules, failing to get an insight into systems at a lower level. The work presented here aims at addressing systems both at high and low level. It was motivated by the model of data mining in more conventional domains which requires data preprocessing prior to the application of algorithms.

The method comprises of a systematic data preparation stage for extracting a number of data models and the relevant databases before applying data mining. Its viability is evaluated in COBOL systems by deriving records about variables, keywords and other grammatical information which are then subjected to mining association rules. Detailed examples are given; conclusions and further work are also presented.

1. Introduction

The significance of legacy system maintenance and the importance of program comprehension as a part of it is well known and documented in the literature [3], [5], [12]. The importance of maintenance has risen dramatically over the past a few years so have the costs [10]. Some of the reasons are reported to be the fast and unplanned modifications, the lack of up-to-date documentation and the lack of experienced software maintainers. A recent survey of maintainers' needs indicated that automated methods are required to provide a quick grasp of a software system, to enable practitioners who are not familiar with a system, to commence maintenance with a level of confidence as if they had this familiarity[17].

Data mining is a technology traditionally used for prediction and understanding patterns and relations in databases. It has also been introduced to legacy program comprehension and used for producing high-level system organizations of source code [11], design recovery [9], [15], identification of data cohesive subsystems [13], legacy systems remodularization [2], modularity assessment, complexity detection and impact of changes prediction [16], [18].

However most of the proposed methods focus on systems at a high level of granularity, involving functions, files and classes even though maintainers also require understanding at a lower level, that of sentences and variables. Furthermore data pre-processing has mostly been ignored by these methods, despite the fact that it is considered to be an important step in "conventional" data mining [7].

The method proposed in this paper addresses both issues by introducing a separate data preparation step aiming at the extraction of both high and low level information from source code, followed by a data mining step. This information includes variables, reserved words and other grammatical tokens found in source code, at the level of paragraphs and sentences alike. This information is stored in a database which is then used as input for data mining tools in order to derive knowledge about the code.

COBOL systems were used to evaluate this method, as the majority of legacy systems are still in this language. The results of this method were assessed using an association rules tool. A discussion about the method's potential for facilitating maintenance concludes the paper. Further work includes applying the model in large-scale systems, different programming languages and the use of other data mining techniques such as clustering, link analysis and classification.

2. Background

2.1 Program comprehension and data mining

Program comprehension is a process that uses existing syntactic and semantic knowledge to acquire new knowledge that ultimately meets the goals of a code cognition task [12]. A way of reducing the difficulties of understanding is to chop the whole system into chunks. Syntactic and semantic knowledge can be described in models (chunks) that explicitly represent different dependency relationships between various software artefacts [6]. There are two different views of dividing of chunks. At system level, chopping means splitting up the system to form smaller chunks. At the entity level, chopping means grouping the entities to form larger chunks [19].

Data Mining aims at inferring useful relations and patterns from data that might not have been noticed or at least could not have been confirmed [8]. Data Mining achieves prediction and description by using several techniques such as classification, clustering, and association rules [7] and has recently been used to analyse software legacy systems.

Most of the proposed methods use clustering to group entities or chunks of code based on their interrelationships or similarity. Clustering algorithms can be used to extract relationships between chunks, and combine low-level chunks into high-level chunks [11], [19]. Thus software systems can be decomposed into subsystems, which provide maintainers with high-level structural information that helps them navigate through the numerous software components, their interfaces and their interconnections [11]. Another way of analysing legacy systems down to the relationships of smaller chunks is through using the calling structure [4]. Alternatively, data from the source code can be used to analyze the semantics, representing the structured record as a contiguous block of storage at runtime [1]. This approach uses the variables defined to trace the data flow, and the data flow to find out the semantics of source code.

2.2 Data preparation

The first step of data mining is pre-processing or data preparation which aims at preparing the data for mining. It comprises two activities: one is finding and assembling the data set the other is manipulating the data to enhance its utility for mining [14]. It requires understanding the objective of data mining in advance, and anticipating their likely results. The difference among algorithms should be known in advance and should be reflected on the preparation data in some degree. This can lead to appropriate data models. The target of data preparation is to produce a "mineable" data representation [14]. When dealing with source code, data preparation should focus on use of semantic and syntactic knowledge in order to include only the necessary data and produce a database of appropriate size.

2.3 Data preparation in COBOL

The objective of data preparation in COBOL is to extract information from code by taking into account the grammar and syntax of the language, setting up

different tables for different data mining algorithms, evaluating their quality and iteratively improving it. Successful COBOL data preparation should achieve the following goals:

✧ Extract the variables of COBOL programs.

✧ Generate a clear view of variable hierarchy which facilitates understanding programs and their structures.

✧ Design a database capturing code structure and variables.

As mentioned above (2.1), clustering is the most widely used technique, focusing on high level modules of source code. Proximity among modules depends on number of accessing and variables transfer or "call" among them. In other words focus is on modules naming and relationships existing. Data tables mainly store two types of data: module description and relationships description. Their contents can be binary showing whether a relationship between modules exists or not, or integer showing the number of times a relationship occurred. This number can represent module distance.

Module relationships can be variable transfer, accessing the same file and using the same sub-modules. Different kinds of relationships may be treated as either the same or having weights according to their kind. For example, accessing the same file can be more important than transferring a variable when determining module relationship, because file access always involves more than one variable, which have several attributes in common.

3. Approach

Facilitating, accelerating and increasing the accuracy of system and program level understanding motivated this work. Research is required to devise a method and generate an industrial scale tool to be used for identifying patterns in code, extracting interrelationships and predicting the impact of change.

3.1 Results usability

Maintainers analyse source code in order to understand it. Data preparation is a prerequisite before using data mining tools to generate results that facilitate program comprehension. Thus two types of results are anticipated.

Firstly, results representing the syntactic and semantic content of the source code. Relationships among variables and blocks of code should be identified. Therefore, it is useful to represent code by means of models or graphs, such as a variable relationship model (similar to an Entity-Relationship model), a variable - block relationship model (similar to the Object-Oriented model), or even models that convey a meaning similar to Data Flow Diagrams and flow chart.

Secondly results representing variable or block relationships, acquired by mining association rules. For example, a rule of the form: "if *SALARY* exists in a paragraph *P*, then *NAME* exists in paragraph *P*" with confidence x% and support y%, (each rule is characterized 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 [8])

3.2 Analysis

3.2.1 Understanding methods. Understanding COBOL code involves two aspects: a) variable relationships and b) relationships among different code elements (elements here mean variables and grammatical tokens). The relationship can be based on value and parameters transferring or control transferring. The understanding of these aspects facilitates understanding the whole structure and meaning of the source code. According to the attributes of COBOL, the way of understanding can be concluded as follows:

- **Process understanding**

In COBOL, sentences can be grouped together if they are e related rather than just collocated. Clustering or association rules can be applied here to produce groups of code blocks or variable relationships within a block of code. The importance of sentences is different when understanding source code; thus there are sentences which should be ignored. Grouping sentences together is a good way to avoid analysing unimportant sentences or the over-detailed analysis.

- **Data relationship understanding**

Understanding data relationships is a key factor for Entity-Relationship (ER) and Object Oriented (OO) models. Applying data mining on variables can produce information of how closely they are related to each other, and it can also give insight into relationships of different parts of the code. Variables in COBOL can be elementary and structured, which maybe further decomposed into elementary and structured. However, structured variables can be huge, and need to be divided into different data block. This work divides variables according to their original structure.

- **Data and block relationship understanding**

This part complements to the first two understandings. It is used to explore how different data relate to different parts of a program, how relevant is the

occurrence of data to the occurrence of others, and how relevant are the different parts of source code to variables. Both code blocks and data blocks need to be divided for this part.

3.2.2 Source code partitioning. The proposed three methods of dividing the source code is along the lines of source code understanding aiming at preserving its integrity. There are presents in the following.

- **Paragraph division**

The *PROCEDURE DIVISION* in COBOL code is composed of paragraphs. The advantage of this method is that it easily preserves the integration of data and their meaning. Usually, sentences in the same paragraph are more likely to have closer relationship. However this method ignores the details inside the paragraph.

- **Semantic division**

Paragraphs can be divided by grammatical factors. For example, a block of code can be divided by the grammatical tokens, such as "*for*", "*while*", "*end*", and so on. This way to divide code can preserve its integrity. A possible shortcoming of this method is that it may involve a complex algorithm.

- **Line division**

Code is composed by a number of lines. A line is a grammatical unit that can convey an integrated meaning. Dividing code in this way can be more straightforward, convenient and detailed. The algorithm required should be relatively easy to understand.

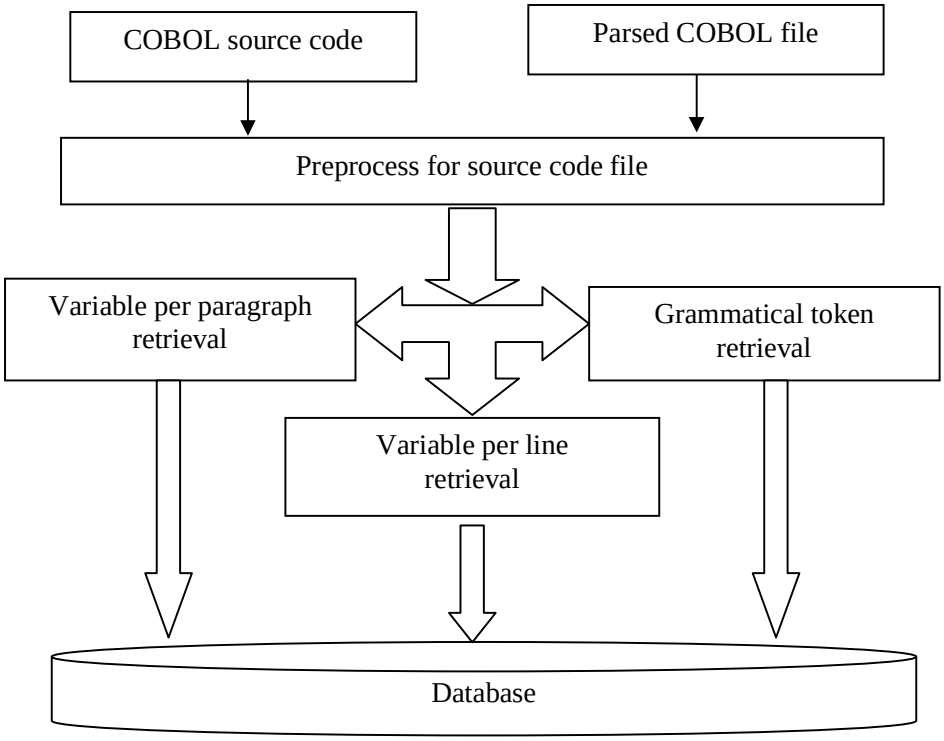

Figure 1: Overview of the system's structure

4. System design

4.1 System structure overview

The system can input either COBOL source code or files generated by a COBOL parser, developed in-house. The parser produces a full list of tokens contained inline with the associated line numbers. When given raw source code for input it generates two types of outputs. The first is a full list of variable names and

reserved words occurring within the same paragraph and the paragraph number, while the second is a similar list of variables and reserved words occurring within the very same line and the relevant line number. Thus two databases are populated when source code is input. One consists of one table per program file, containing one record per paragraph. The other contains one table per paragraph, having one record per line of code, excluding blanks and comments.

When a parsed COBOL source code file is input, the system generates a table containing a record of all grammatical tokens per line and the relevant line number. Figure 1 gives an overview of the structure of the system.

4.2 The main functional modules

The system consists of three main functional modules, as presented in Figure 1, one for retrieving information organized in paragraphs of code, one in lines within paragraphs and another retrieves grammatical information as produced by a parser. Their outputs are presented in the following.

Paragraph name	**Name of variable 1**	**Name of variable 2**	**Name of variable 3**	**..**
Paragraph name1	1	0	1	
Paragraph name2	1	1	1	
Paragraph name3	0	0	1	
Paragraph name4	0	1	0	
……				

Figure 2: The output for high-level data retrieval from source code

4.2.1 Variable per paragraph retrieval. The output of this module is a table with rows including a list of variables in each paragraph, the occurrences of variables and the relevant paragraph name. An example is given in Figure 2.

Relationships among variables can be found from such table, as well relationships among paragraphs by analyzing variable occurrences within them. A number of variables occurring in the same group of paragraphs is a strong indication that these paragraphs are related and may demonstrate similar functionality.

4.2.2 Variable per line retrieval. Similarly the output of this module is one table for each paragraph with rows including a list of variables per line, the occurrences of variables and the relevant line name. Obviously, the database produced contains tables as many as the number of paragraphs in the program.

4.2.3 Grammatical token retrieval. The output of this module is a single table with rows including a list of grammatical token per line and the relevant line name. This table can be used to identify relationships among lines by analysing the similarity of usage of grammatical token.

5. System Evaluation

Data Miner for Source Code[1] (*DMSC*) was used to evaluate the tables of variables produced from a COBOL program which reads employee salary information from a file, and then prints it out in a formal format.

5.1 Evaluate the table of paragraphs' variables

The following are a few rules found from the paragraphs' variable table by using *DMSC*.

✧ If "WS_WORK_AREAS___WS_LINE_CTR" in a paragraph then "OUT_REPORT_REC" also in the paragraph with 44% support, 100% confidence. **(1)**

✧ If "WS_WORK_AREAS___WS_DEPT_HOLD" in a paragraph then "OUT_REPORT_REC" also in the paragraph with 44% support, 100% confidence. **(2)**

These rules can be used to find out the relationship among variables. WS_WORK_AREAS___WS_LINE_CTR is used to set the blank between different lines in one record while WS_WORK_AREAS___WS_DEPT_HOLD is used to set the list separator for dividing records in different sets, such as employees' records in different department. OUT_REPORT_REC is used to print out the records. For this reason, they have close relationship.

Paragraphs can be grouped by using the clustering rules. The paragraphs can be grouped together by applying the same rules, which were discovered by using association rules in the previous step. For example, paragraphs 300-HEADING-RTN and 500-DEPT-

[1] DMSC is a tool for mining association rules developed by L. Sinos at UMIST.

BREAK were grouped together because both support the rule:

If 'WS_WORK_AREAS___WS_LINE_CTR in a paragraph then OUT_REPORT_REC also in the paragraph.

These two paragraphs have high possibility to have functions, algorithms, or logic in common. In this case, 300-HEADING-RTN can be PERFORMED FROM 500-DEPT-BREAK.

The objective of paragraphs' variable table is to help to understand variable relationships, and paragraph relationships by using data mining tools. As indicated this method satisfies the initial objective, by revealing variable and paragraph relationships.

5.2 Evaluate the table of lines' variables

The following are rules found at the paragraphs' variable table. They are similar to the ones used for evaluating the table of paragraphs' variables.

✧ If IN_EMPLOYEE_REC___IN_DEPT in a line then WS_WORK_AREAS___WS_DEPT_HOLD also in the paragraph with 9% support, 100% confidence. **(3)**

✧ If IN_EMPLOYEE_REC___IN_TERR in a line then WS_WORK_AREAS___WS_TERR_HOLD also in the paragraph with 9% support, 100% confidence. **(4)**

Variable relationships are easier to be found in lines than in paragraphs, as variables in neighbouring lines are more likely to have closer relationship. For example, according to the result, IN_EMPLOYEE_REC___IN_DEPT has direct relationship with WS_WORK_AREAS___WS_DEPT_HOLD. When we look at the source, the IN_EMPLOYEE_REC___IN_DEPT is used to directly compare to WS_WORK_AREAS___WS_DEPT_HOLD. The IN_EMPLOYEE_REC___IN_DEPT is the record that reads from the input file, while WS_WORK_AREAS___WS_DEPT_HOLD is for output. These two variables are used to judge the break of department and territory and print it out. While the IN_EMPLOYEE_REC___IN_DEPT is a break, then WS_WORK_AREAS___WS_DEPT_HOLD will be printed out. This is the reason why these two variables appear associated.

The lines can be clustered together by the similar rules they contain. The group of lines derived by *DMSC* is as follow.

✧ Lines 2 and 6 are grouped together. Both of them contain the same rule, that is rule **(3)**. Line 2 is "MOVE IN-DEPT TO WS-DEPT-HOLD". Line 6 is "WHEN IN-DEPT NOT = WS-DEPT-HOLD".

✧ Lines 3 and 8 are grouped together. Both of them contain the same that is rule **(4)**. Line 3 is "MOVE IN-TERR TO WS-TERR-HOLD". Line 8 is "WHEN IN-TERR NOT = WS-TERR-HOLD".

Lines 2 and 3 here are used to convey the value from the input record to the output record on some condition. Lines 6 and 8 are used to find out whether the value has been conveyed to output record or not.

If the value has been conveyed from IN-DEPT to WS-DEPT-HOLD and from IN-TERR to WS-TERR-HOLD, that means the condition of first record has been met, then the program need to print both the break of department and territory. In both cases, the clustering rules of lines implied all the relationships.

The objective of the lines' variable table is to help to understand variable relationships, and line relationships by using data mining tools. As indicated this method satisfies the initial objective, by revealing variable and line relationships.

5.3 Evaluate the table of lines' grammar tokens

No tools were identified suitable to evaluate the table for lines' grammar tokens. However this result can still be theoretically evaluated. The relationship of different lines can be estimated by simply calculating the grammatical tokens in common. The table contains both COBOL reserved and user-defined words. If there are more user-defined words in common in different lines that means there are more variables or paragraph names used in common in these lines. It also means these different lines have close relationship. If there are more reserved words in common in different lines, that means these different lines have more operation in common.

6. Conclusions and further work

6.1 Conclusions

Data mining can facilitate program comprehension and maintenance, even though the outcome is strongly influenced by the data preparation process which precedes it. However, few systematic data preparation methods have been proposed in this area. This work

addresses this deficiency by introducing a systematic method for data preparation, dealing with it as a separate process from data mining. Compared to other approaches this method provides more aspects of information at both a high and a low level and produces results which can be used by different data mining tools and algorithms, giving a more complete understanding of source code

Program comprehension is achieved by source code slicing, that is by extracting useful data from different slices of source code, and revealing the relationship of slices. Data preparation as proposed in this paper uses two approaches to slice the source code, and then extract both syntactic and semantic data from the slices of code. Using the results of the method, data mining tools can easily accomplish the mission of relationship revealing.

6.2 Future work

The proposed method broadens the area of tools achieving program comprehension. It is fully automated and can achieve more detail system understanding. However work can be further done in the following areas:

First, the model needs to be applied and tested in large-scale systems and even different programming languages. Then more information from source code can be extracted. More accurate block slicing can also be introduced by applying clustering tools to find optimum code partitions to be used for variable or token extraction. Finally, more data mining techniques such as clustering, link analysis and classification algorithms can be applied in search of more hidden knowledge.

References

[1] M. Andersson, "Searching for semantics in COBOL legacy applications", DS-7, *Reverse Engineering*, 1997.

[2] N. Anquetil and T. C. Lethbridge, "Experiments with Clustering as a Software Remodularization method", *Proc. 6th Working Conf. Reverse Engineering* (WCRE 99), IEEE Comp. Soc. Press, Oct. 1999, pp. 235-255.

[3] F. Balmas, H. Wertz and J. Singer, "Understanding Program Understanding", *Proc. 8th Int'l Workshop Program Comprehension* (IWPC 00), IEEE Comp. Soc. Press, 2000, pp. 256.

[4] E. Burd and M. Munro, "Evaluating the Use of Dominance Trees for C and COBOL", *Proc. Int'l Conf. Software Maintenance* (ICSM 99), IEEE Comp. Soc. Press, 1998.

[5] K. Erdös and H.M. Sneed, "Partial Comprehension of Complex Programs (enough to perform maintenance)", *Proc 6th Int'l Workshop Program Comprehension* (IWPC 98), IEEE Comp. Soc. Press, 1998, pp. 98-105.

[6] A.R. Fasolino and G. Visaggio, "Improving Software Comprehension Through an Automated Dependency Tracer", *Proc. 7th Int'l Workshop Program Understanding* (IWPC 99), IEEE Comp. Soc. Press, 1999.

[7] U. Fayyad, G. Piatetsky-Shapiro and P. Smyth, 'From Data Mining to Knowledge Discovery: an Overview', *Advances in Knowledge Discovery and Data Mining*, AAAI Press, 1996, pp. 1-34.

[8] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, 2000.

[9] T. Kunz and J. P. Black, "Using Automatic Process Clustering for Design Recovery and Distributed Debugging", *IEEE Transactions on Software Engineering*, vol. 21, no. 6, June 1995, pp. 515-527.

[10] K. Lano, *Reverse Engineering and Software Maintenance: a Practical Approach*, McGraw-Hill, 1994.

[11] S. Mancoridis, B. S. Mitchell, C. Rorres, Y. Chen and E. R. Gansner, "Using Automatic Clustering to Produce High-Level System Organisations of Source Code", *Proc. 6th Int'l Workshop Program Understanding* (IWPC 98), IEEE Comp. Soc. Press, 1998, pp. 45-53.

[12] A. Von Mayrhauser and A.M. Vans, "Program Comprehension During Software Maintenance and Evolution", *IEEE Computer*, vol. 28, no. 8, Aug. 1995, pp. 44-55.

[13] 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.

[14] D. Pyle, *Data Preparation for Data Mining*, Morgan Kaufmann, 1999.

[15] 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.

[16] Tjortjis C. and Layzell P.J., "Using Data Mining to Assess Software Reliability", *Suppl. Proc. IEEE 12th Int'l Symposium Software Reliability Engineering* (ISSRE2001), IEEE Comp. Soc. Press, 2001, pp. 221-223.

[17] Tjortjis C. and Layzell P.J., "Expert Maintainers' Strategies and Needs when Understanding Software: A Qualitative Empirical Study", *Proc. IEEE 8th Asia-Pacific Software Engineering Conf.* (APSEC 2001), IEEE Comp. Soc. Press, 2001, pp. 281-287.

[18] V. Tzerpos and R. Holt, "Software Botryology: Automatic Clustering of Software Systems", *Proc. 9th Int'l Workshop Database Expert Systems Applications* (DEXA 98), IEEE Comp. Soc. Press, 1998, pp. 811-818.

[19] T. A. Wiggerts, "Using Clustering Algorithms in Legacy Systems Remodularization", *Proc. 4th Working Conf. Reverse Engineering* (WCRE 97), IEEE Comp. Soc. Press, 1997, pp. 33-43.