Application of Data Mining Techniques for Improving Software Engineering

Wahidah Husain[1], Pey Ven Low[2], Lee Koon Ng[3], Zhen Li Ong[4],
School of Computer Sciences, Universiti Sains Malaysia
11800 USM, Penang
wahidah@cs.usm.my[1], {lpv10014[2], nlk1001[3], ozl1001[4]}@student.usm.my

***Abstract*—Computer software plays an important role in business, government, societies, and the sciences. To solve real-world problems, it is important to improve the quality and reliability of computer software. Software Engineering is the computing field concern with designing, developing, implementing, maintaining, and modifying software. Software Engineering data consists of sequences, graphs, and text.. In this paper, we study how data mining techniques can be applied in solving Software Engineering problems. These techniques are applied to detect problems such as bugs, to aid in pattern discovery, and to help developers deal with the complexity of existing software, in order to create more failure-free software. At the end of this paper, we suggest specific data mining techniques for each type of Software Engineering data.**

***Keywords*—Software Engineering, Data Mining, Text Mining, Sequence Mining, Graph Mining**

I. INTRODUCTION

The economies of all developed countries are dependent on software, because software controls systems that affect our daily needs. Thus, Software Engineering (SE) has become more and more important. SE concerns computer-based system development; this includes system specification, architectural design, integration, and deployment. With the increasing importance of SE, the difficulty of maintaining, creating and developing software has also risen. Challenges in SE include requirement gathering, systems integration and evolution, maintainability, pattern discovery, fault detection, reliability, and complexity of software development [1, 2]. SE data can be categorized into three types: sequences, graphs, and text [3].

Data mining is a process that employs various analytic tools to extract patterns and information from large datasets. Today, large numbers of datasets are collected and stored. Human are much better at storing data than extracting knowledge from it, especially the accurate and valuable information needed to create good software. Large datasets are hard to understand, and traditional techniques are infeasible for finding information from those raw data. Data mining helps scientists in hypothesis formation in biology, physics, chemistry, medicine, and engineering. The data mining process is shown in Fig. 1 below.

There are seven steps in the process: data integration, data cleaning, data selection, data transformation, data mining, pattern evaluation and knowledge presentation. Data mining techniques that can be applied in improving SE include generalization, characterization, classification, clustering, associative tree, decision tree or rule induction, frequent pattern mining, and etc. [5].

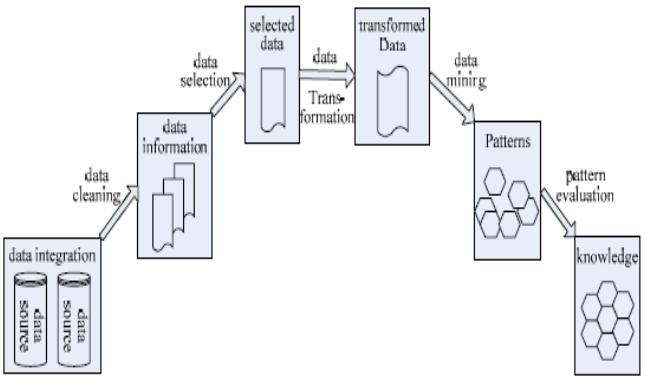

Figure 1. Data Mining Process [4]

The purpose of this study is to explore how data mining techniques can be applied to improve SE. Objectives of this study are:

- To review the concept of SE and data mining
- To determine the problems in SE
- To identify data mining techniques that can be applied to solve SE problems.

In this paper, we focus on the issues particular to each type of SE data, and the specific data mining techniques that can solve those problems. We suggest the best data mining technique(s) for each SE data type, which are sequences, graphs, and text.

II. BACKGROUND STUDY

SE has become increasingly important these days, and research on its problems has proposed various methods for improving SE. The study of SE problems has followed several approaches. Early research of Thayer et al. [6] was concerned with SE project management. They introduced planning

problems, organizing problems, staffing problems, and controlling problems as major challenges in this area. Ramamoorthy et al. [7] stated that as more complex software applications are required, programmers will fall further behind the demand. This causes the development of poor quality software and higher maintenance costs. The problems stated by Thayer are more closely tied to the processes of SE project management, while those introduced by Ramamoorthy are more closely tied to the limitations of human beings. Later work by Clarke [8] identified challenges in SE associated with the complexity of the software development process. Similar to our study, he stated that the complexity of software development causes the software to become harder to maintain. This leads to other problems such as software integrity and difficulty in detecting application bugs or flaws. A bug is a flaw in a computer program that can ultimately cause glitches, program failure or software destruction.

With the increasing importance of SE, data mining has become an important tool for solving SE problems. Data perturbation techniques for preserving privacy in data mining were proposed by Islam and Brankovic [9]. Aouf proposed the clustering technique to identify patterns in the underlying data [10]. Later work by Ma and Chan [11] suggested iterative mining for mining overlapping patterns in noisy data. The three data mining approaches discussed above are all clustering techniques. The data perturbation technique described by Islam and Brankovic [9] involves *adding* noisy data into some part of the dataset in order to preserve privacy, while Ma and Chan [11] were concerned with the *elimination* of noisy data to enable extracting valuable information.

Different types of data require different data mining techniques. In this paper, we present specific problems that exist in SE and apply a specific data mining technique to solve each of them.

III. RESEARCH METHODOLOGY

In this research, we decided to use a descriptive approach as our research methodology. The descriptive approach is primarily used for gathering data, with attempting to generate rich descriptions and explanations of the object of study. The project may also include gathering opinions about the desirability of the present state of things. It relies on both qualitative and quantitative data. The research methodology of our study is shown in Fig. 2 below.

First, we collect data from past literature: journals, magazine, written documents, sources from the internet, books and published papers. After identifying the problem in SE (reliability of software and complexity of development processes), we analyze the data we collected about the various data mining techniques, and make a decision as to which technique is best suited to solve our problem. Finally, we propose the best data mining technique to solve SE problem.

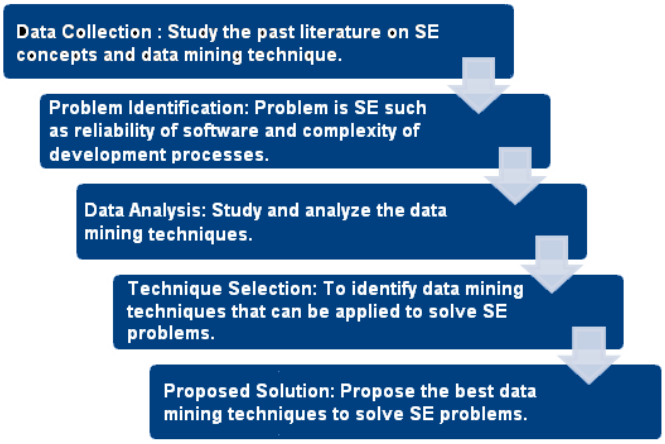

Figure 2. Research Methodology Phase

IV. ANALYSIS AND FINDINGS

SE data can be categorized into three categories as follows [3]:

- Sequences, such as execution traces collected at run-time, and static traces extracted from source code
- Graphs, such as dynamic call graphs extracted from source code
- Text, such as code comments, documentation, and bug reports.

When problems such as bugs or flaws arise in systems or software, it is difficult for developers or programmers to determine the cause(s). Data mining is a valuable tool for solving SE problems. Data mining techniques can be applied to solve problems in all three categories of SE data. The following sub-topics review and describe data mining techniques that can solve problems in each type of SE data.

A. Sequence Data Mining

Examples of SE sequence data include method-call data that is dynamically collected during program execution, or statically extracted from source code. The challenge inherent in SE sequence data is the difficulty of extracting sequential patterns and information from the source code [3] or program during program execution [12] for software bug or flaw detection. Data mining techniques that can be applied to solve these problems are frequent itemset mining (FIM) and frequent sequence mining (FSM). FIM and FSM are types of frequent pattern mining that use alternative support counting technique [3].

Fig. 3 shows the process of frequent pattern mining on program source code. A program source code is composed of elements written in a particular programming language. To mine sequential data from source code, we need to divide it into different units, such as tags, blocks, function, and classes, for further mining. The information extracted from different units of source code can be used to remove the wrong source code [13]. In this case, FIM can be used to characterize the importance of program elements. FIM is used to mine the

frequency of program elements, and to extract the procedural rules of fault and bug detection. FIM is only suitable for mining static traces or paths extracted from source code as it does not reflect the sequential order of information in the mined pattern [3, 14]. FIM uses a bottom-up approach and follows the principle wherein an itemset X of variables can only be frequent if all its subsets are also frequent [14]. Thus, it is not suitable for application to run-time data [13].

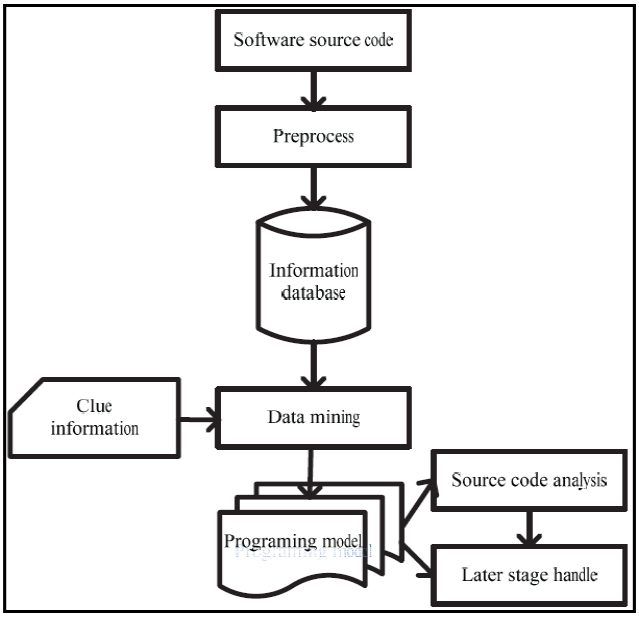

Figure 3. The Procedure of Frequent Pattern Mining on Source Code [13]

Run-time paths capture the ordered sequence of components or events that take place in an application [14]. The information can be used to analyze the overall software design. However, a run-time path can be very long, and numerous different run-time paths may exist in a system [14]. FSM can identify the frequent sequences that occur across different run-time paths [14]. FSM can be applied to the run-time paths [14] during program execution to identify the repeating sequences, patterns, and design flaws that lead to poor system performance. The common problem in systems, especially enterprise systems, is performance degradation when systems retrieve data from databases. Run-time paths contain performance information of the application, corresponding to users' actions. In solving performance issues of an application, it is more accurate to identify resource intensive method sequences than frequent sequences [14]. Applying FSM to mine the run-time paths not only can identify the resource intensive methods, but also give the sequences in which they occurred. The sequential information extracted by FSM allows developers to quickly locate and correct the flaws in an application.

FIM helps developers determine the important elements of a particular source code. This enables them in future to create and develop more effective and failure-free software that has similar requirements. Besides this, the detection of bugs and flaws in the source code of an application helps in system maintenance, and produces more reliable software. FSM saves developers time in searching for and fixing software bugs or faults during program execution. This improves software reliability and maintenance. Besides this, system performance can be enhanced as systems require less time for retrieving data from databases.

B. Graph Data Mining

Most SE data can be represented as graphs. Dynamic call graphs generated during execution of a program and static call graphs generated from program source code are instances of SE data represented in graphs [3]. A graph is an expressive representation for SE data and is useful for modeling complicated structures and their relationships [15]. Mining graph data has great potential to help in software development. Thus, graph data mining is often an active research area in SE.

As graph data are usually large and complex, it is hard for the human eye to uncover patterns in them and classify the patterns to ease program bug analysis [16]. Application of graph classification can solve this problem, since the technique automates the identification of subgraph patterns from graph data and constructs a graph classification model for automating the classification process.

Software is unlikely to ever be failure-free. The more failures and bugs encountered in the software, the more unreliable the software is. By detecting bugs and failures and pinpointing their origin software programmers or developers can fix them and thus increase the software reliability. There are two types of software bugs: crashing and non-crashing [16]. Crashing bugs are those that cause program execution to halt under non-ordinary conditions. Examples of crashing bugs include inputting characters into a field of integer data type, or referring to a null pointer. Non-crashing bugs do not terminate the program [16] but may result in errors in execution or output. An example of a non-crashing bug would be logical errors such as the final result generated by the program is not the result that it should be. Non-crashing bugs are more difficult to be detected as they have no crashing point -- that is, the program continues running past the point of error [16].

In order to disclose traces of non-crashing bugs, graph classification technique is used. Software behavior graphs from program execution comprise the data to be mined. Fig. 4 shows an example of behavior graphs. These graphs represent one correct run and one incorrect run of a program. The node of the graph represents the function of the program while the edge represents the function calls. However, the number of frequent subgraphs mined from behavior graphs is often large which may result in low performance of graph mining and classification [16] due to the difficulties and time consuming nature of mining huge numbers of frequent subgraphs. Closed frequent subgraphs are a legitimate substitute for frequent graphs for classification purposes, as the number of closed

frequent subgraphs generated is lower. The mined closed frequent subgraphs are used for classifier construction [15]. A classifier is built at each checkpoint for every function in a program to detect the location of the bug [16].

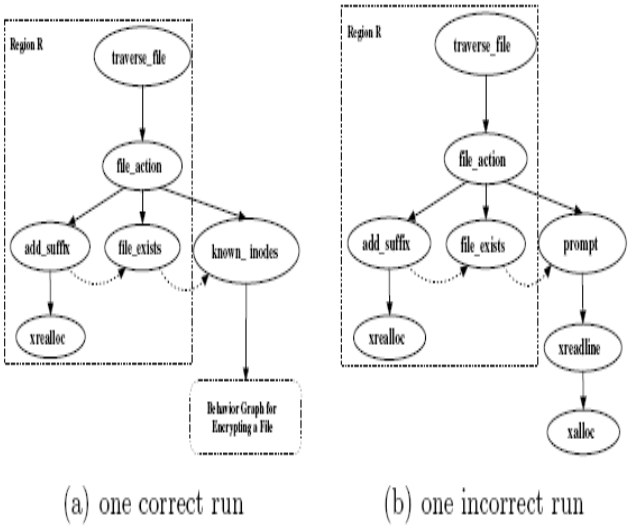

Figure 4. Example of behavior graphs [16]

Bug detection is achieved by monitoring for classification accuracy boost. The classification accuracy of each classifier will remain low before the bug is triggered [3], because the classifier lacks information about the behavior of the bug. When the bug is triggered, the classification accuracy of the classifier at that checkpoint increases, and therefore the location of that classifier is probably the location of the bug [16]. This eases the process of debugging for non-crashing bugs, since programmers thus can deal with the bugs directly without going through the difficulty and time to search for the bugs on their own.

C. Text Data Mining

Currently, approximately 80% of information is stored in computers as text [17]. Example of SE text data includes bug reports, project reports, e-mails and code comments [3]. The majority of software used every day generates bug reports (BR). BRs are compiled from various sources such as the research and development sector and also from end user feedback. When a bug report is sent in, it is labeled as either a security bug report (SBR) or non-security bug report (NSBR). SBRs have a higher priority than NSBR. Gegick et al [17] stated that correctly labeling SBRs and NSBRs is critical to good security practice since a delay in identifying and fixing security bugs has the potential to cause serious damage to software-system stakeholders.

In reality, during bug report classification SBRs are often mislabeled as NSBRs. This is partially caused by the lack of human knowledge of security domain issues. Such errors can cause serious damage to the software system due to delays in ascertaining authenticity and in rectifying the bugs. Though people on the security team are able to manually inspect NSBR submitted to identify potentially mislabeled SBR, this exercise is time consuming and often either not feasible or not regular practice [17].

To solve this problem, we can apply a technique known as text mining on natural-language description. Text data mining refers to the discovery of unknown information and potentially useful knowledge from a collection of texts, by automatically extracting and analyzing information [18]. A key factor is to extract the appropriate information and link it together to form new facts to be explored further. The Natural Language Description Technique combines linguistics and computer science to enhance the interactions between computers and natural languages [19].

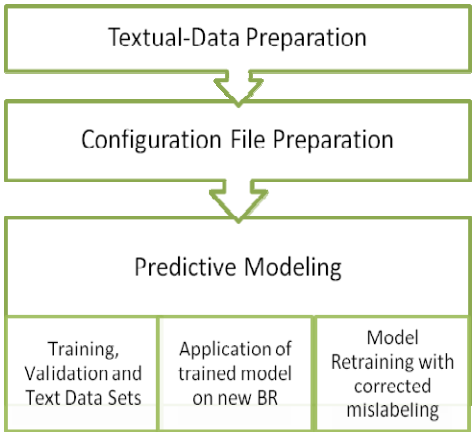

Figure 5. Bug Reports Classification using Text Data Mining [17]

Fig. 5 shows the process of bug report classification using text data mining's Natural Language Description Technique. The initial step is to obtain a set of labeled BR data that contains textual descriptions of bugs correctly labeled to indicate whether they should be classified SBR or NSBR. Labeling of the BR data set is required for creating and evaluating a Natural Language Predictive Model.

The next step is to create three configuration files used in text mining: start list, stop list, and synonym list. A stop list contains terms such as articles, prepositions, and conjunctions that are not relevant in text mining. If a term in the document is found in the stop list it is not entered into matrix. Terms in the start list, on the other hand, are highly relevant to SBRs, therefore their appearance in a BR increases the probability that the BR is an SBR [17]. The synonym list contains terms with the same meanings (e.g., “buffer overrun” and “buffer overflow” have the same meaning). Terms in a synonym list are treated equally in text mining. Thus, for example, a less frequently used term that is associated with SBRs may be assigned more weight in the prediction model than a more-frequent term that is not associated with SBRs, or a term that is not itself frequently associated with SBRs may receive more priority in the prediction model if it is a synonym with a term that is frequently used in SBRs. The third step is to train, validate, and test the predictive model that estimates the probability that a BR is an SBR. [17].

Text mining on natural-language descriptions can be applied to BRs to determine whether those reports are SBR or NSBR. Software engineers can use this technique to automate the classification process of BRs from huge bug databases to save time and reduce dependency on human efforts, as well as to improve the priority ranking of each SBR and ensure that the described security bug receives the appropriate effort and a timel fix, thereby improving the software.

D.Discussion of the Three Data Mining Techniques

In sections A, B, and C above, we discussed the data mining techniques that can be used to solve problems in each type of SE data (sequences, graphs, and text) Hence, we have shown that the different nature of data demands different data mining techniques in solving SE problems. Data mining techniques that can be applied to solve certain SE data problems might not be able to solve other SE data problems. The data mining techniques appropriate for solving sequence data problems are frequent itemset mining (FIM) and frequent sequence mining (FSM), while the data mining technique best suited for solving SE graph data problems is classification, and text mining on natural language technique is the most useful tool for solving text data problems.

V.CONCLUSION

In this paper, we have discussed the application of data mining for solving of a number of Software Engineering problems. A number of problems encountered in the SE field, such as bug occurrence, high cost of software maintenance; unclear requirements and so on can reduce software quality and productivity. We outlined the data mining techniques that can be applied to various types of SE data in order to solve the challenges posed by SE tasks such as programming, bug detection, debugging, and maintenance. Our studies show that data mining techniques have proven to be effective for improving SE by increasing software reliability and quality.

As to future direction, new mining techniques or algorithms are needed in order to solve unclear software requirements in mining SE data. Another challenge is the ability to mine combination types of SE data (e.g., text and graphs together) for more informative patterns. Further research needs to be done to explore the potential of mining combination data to solve numerous real-world problems in SE.

ACKNOWLEDGMENT

The authors wish to thank Universiti Sains Malaysia (USM) for supporting this research.

REFERENCES

[1] J. Clarke, “Refomulating software as s search problem,” *IEEE Proc.: Software Vol. 150, No.3*, pp. 161-175, June 2003.

[2] X. L. Fern, C. Komireddy, V. Gregoreanu, and M. Burnett, “Mining problem-solving strategies from HCL data,” *ACM Trans. Computer-Human Interaction*, Vol. 17, No. 1, Article 3 pp. 1-22, March 2010.

[3] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, “Data mining for software engineering,” IEEE Computer Society August 2009, pp. 55-62, 2009.

[4] Y. Chen, X. H. Shen, P. Du, and B. Ge, “Research on software defect prediction based on data mining,” *2nd International Conference on Computer and Automation Engineering (ICCAE) 2010*, Vol. 1, pp. 563-567, Apr. 2010.

[5] R. W. DePree, “Pattern recognition in software engineering,” *IEEE Computer 1983*, pp. 48-53, 1983.

[6] R. H. Thayer, A. Pyster, and R. C. Wood, “Validating solutions to major problems in software engineering project management,” *IEEE Computer Society*, pp. 65-77, 1982.

[7] C. V. Ramamoorthy, A. Prakash, W. T. Tsai, and Y. Usuda, “Software engineering: problems and perspectives,” *IEEE Computer Society*, pp. 191-209, Oct 1984.

[8] J. Clarke et al., “Refomulating software engineer as a search problem,” *IEEE Proceeding Software*., Vol. 150, No. 3, pp. 161-175, June 2003.

[9] M. Z. Islam and L. Brankovic, “Detective: a decision tree based categorical value clustering and perturbation technique for preserving privacy in data mining,” *Third IEEE Conference on Industrial Informatics (INDIN)*, pp. 701-708, 2005.

[10] M. Aouf, L. Lyanage, and S. Hansen, “Critical review of data mining techniques for gene expression analysis,” *International Conference on Information and Automation for Sustainability (ICIAFS) 2008* , pp. 367-371, 2008.

[11] P. C. H. Ma and K. C. C. Chan, “An iterative data mining approach for mining overlapping coexpression patterns in noisy gene expression data,” *IEEE Trans. NanoBioscience*, Vol. 8 No. 3, pp. 252-258, Sept 2009.

[12] B. Bhasker, “An algorithm for mining large sequence in databases,” *Communications of the IBIMA*, Vol. 6, pp. 149-153, 2008.

[13] C. M. Zong and Z. L. Li, “Applying data mining techniques in software development,” *2nd IEEE Int. Conf.*, pp. 535-538. Apr. 2010.

[14] T. Parsons, J. Murphy, and P. O Sullivan, “Applying frequent sequence mining to identify design flaws in enterprise software systems,” *Machine Learning and Data Mining in Pattern Recognition*, 5th Int. Conf., pp. 261-275, MLDM 2007.

[15] J. Han, and M. Kamber, *Data mining concepts and techniques*, 2nd Edition, The Morgon Kaufman Series in Data Management Systems, 2005.

[16] C. Liu et al., “Mining behavior graphs for “backtrace” of noncrashing bugs,” *Proc. of 2005 SIAM Int. Conf. on Data Mining (SDM'05)*, pp. 286-287, 2005.

[17] M. Gegick, P. Rotella and T. Xie, “Identifying security bug reports via text mining: an industrial case study,” *7th IEEE Working Conf. Mining Software Repositories (MSR)*, Cape Town, 2010.

[18] G. Vishal and S. L Gurpreet, “A survey of text mining techniques and applications,” *Journal of Emerging Technologies in Web Intelligence*, Vol. 1, No. 1, August 2009.

[19] X.Y. Wang, Z. Lu, T. Xie, A. John and S. Jiasu, “An approach to detecting duplicate bug reports using natural language and execution information,” *ACM/IEEE 30th Int. Conf. on Software Engineering, ICSE 08*, 2008