www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 3 Issue 6 June, 2014 Page No. 6722-6725

Application of Data Mining Techniques on Software Engineering Data for Software Quality

Pooja Arora
BCIIT
Delhi, India
poojadhanrajani@gmail.com

***Abstract*—The processes of Software engineering are complex and produces large number and variety of artifacts. The potential of data mining technique on this large valuable data is to better manage the software projects and to produce high-quality software systems that are delivered on time and within budget. This paper present the latest research in mining software engineering data, software engineering task helped by data mining, kind of software engineering data, data mining techniques used in software engineering.**

***Index Terms*—Software Engineering, Data mining, software Quality**

I. INTRODUCTION

Software Engineering data contains large amount of information about software project's status, progress and evolution. Working with Nokia, Gall et al. [4] have shown that software repositories can help developers change legacy systems by pointing out hidden code dependencies. Working with Bell Labs and Avaya, Graves et al. [5] andMockus et al. [8] demonstrated that historical change information can support management in building reliable software systems by predicting bugs and effort. Working on open source projects, Chen et al. [3] have shown that historical information can assist developers in understanding large systems. SE data can be used to gain empirically-based understanding of software development. It can be also used to prdict, plan and understand various aspects of a project and support future development and project management activities.

Types of SE Data
- Historical data
- Multi-run and multi-site data
- Source code data

Data Mining Techniques in SE
- Association rules and frequent patterns
- Classification
- Clustering

Software engineering data can be broadly categorized into:

- Sequences such as execution traces collected at runtime, static traces extracted from source code, and cochanged code locations. Examples of mining algorithms used here are Frequent Item set /Sequence/ Partial ordering mining, sequence matching/clustering/classification. Examples of software engineering tasks here are Programming, maintenance, bug detection and debugging.
- Graphs such as dynamic call graphs collected at runtime and static call graphs extracted from source code;Examples of mining algorithms used here are Frequent Sub-graph mining, Graph matching/clustering/classification. Examples of software engineering tasks here are bug detection and debugging.
- Text such as bug reports, e-mails, code comments, and documentation. Examples of mining algorithms used here are Text Matching/ Clustering/Classification. Examples of software engineering tasks here are Maintenance, Bug Detection and Debugging.

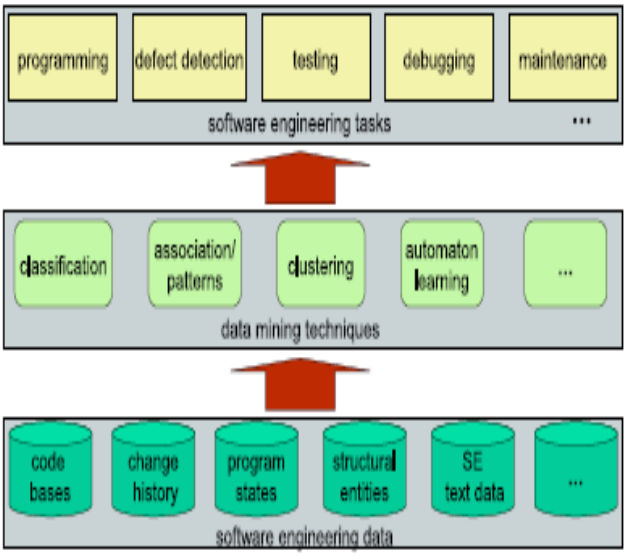

Figure 1. overview of mining SE data

2. SOFTWARE ENGINEERING DATA

SE data concerns the 3Ps: people, processes, and products. People include software developers, testers, project managers, and users. Processes include various development phases and activities such as requirements, design, implementation, testing, debugging, maintenance, and deployment. Products can be structured, such as source code (including production and test code), or nonstructured,
such as documentation and bug reports. As the first column of Table 1 shows, SE data can be broadly categorized into
• *sequences* such as execution traces collected at runtime,
static traces extracted from source code, and co-changed code locations;
• *graphs* such as dynamic call graphs collected at runtime and static call graphs extracted from source code; and
• *text* such as bug reports, e-mails, code comments, and documentation.
To improve both software productivity and quality, software engineers are increasingly applying data mining algorithms to various SE tasks. For example, such algorithms can help engineers figure out how to invoke APImethods provided by a complex library or framework with insufficient documentation. In terms of maintenance, such algorithms can assist in determining what code locations must be changed when another code location is changed.
Software engineers can also use data mining algorithms to hunt for potential bugs that can cause future in-field failures as well as identify buggy lines of code (LOC) responsible for already-known failures. The second and third columns of Table 1 list several example data mining algorithms and the SE tasks to which engineers apply them.

Software Engineering Data	Mining Algorithms	Software Engineering Tasks
Sequences: execution/static traces, co-changes, etc.	association rule mining, frequent itemset/subseq/ partial-order mining, seq matching/clustering/classification, etc.	programming, maintenance, bug detection, debugging, etc.
Graphs: dynamic/static call graphs, program dependence graphs, etc.	frequent subgraph mining, graph matching/clustering/ classification, etc.	bug detection, debugging, etc.
Text: bug reports, emails, code comments, documentations, etc.	text matching/clustering/ classification, etc.	maintenance, bug detection, debugging, etc.

Table 1. Software engineering data, mining algorithms, and software engineering tasks [1]

3. MINING METHODOLOGY FOR SOFTWARE ENGINEERING DATA

Figure 3 shows an overview of the five main steps in mining SE data. Software engineers can start with either a problem-driven approach (knowing what SE task to assist) or a data-driven approach (knowing what SE data to mine), but in practice they commonly adopt a mixture of the first two steps: collecting/investigating data to mine and determining the SE task to assist. The three remaining steps are, in order, preprocessing data, adopting/adapting/developing a mining algorithm, and post processing/applying mining results. Preprocessing data involves first extracting relevant data from the raw SE data—for example, static method- call sequences or call graphs from source code, dynamic method-call sequences or call graphs from execution traces, or word sequences from bug report summaries.
This data is further preprocessed by cleaning and properly formatting it for the mining algorithm. For example, the input format for sequence data can be a sequence database where each sequence is a series of events.
The next step produces a mining algorithm and its supporting tool, based on the mining requirements derived in the first two steps. In general, mining algorithms[1] fall into four main categories:
• *frequent pattern mining*—finding commonly occurring patterns;
• *pattern matching*—finding data instances for given patterns;
• *clustering*—grouping data into clusters; and
• *classification*—predicting labels of data based on already-labeled data.
The final step transforms the mining algorithm results into an appropriate format required to assist the SE task. For example, in the preprocessing step, a software engineer replaces each distinct method call with a unique symbol in the sequence database being fed to the mining algorithm.
The mining algorithm then characterizes a frequent pattern with these symbols. In postprocessing, the engineer changes each symbol back to the corresponding method call. When applying frequent pattern mining, this step also includes finding locations that match a mined pattern—for example, to assist in programming or maintenance—and finding locations that violate a mined pattern—for example, to assist in bug detection.

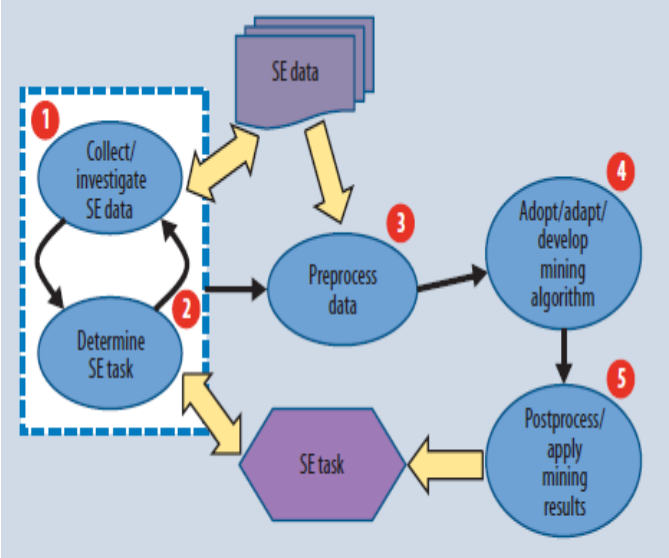

Figure 3. Five basic steps for mining software engineering data

4. DATA MINING FOR SOFTWARE ENGINEERING

Due to its capability to deal with large volumes of data and its efficiency to identify hidden patterns of knowledge, data mining has been proposed in a number of research work as mean to support industrial scale software maintenance, debugging, testing. The mining results can help software engineers to predict software failures, extract and classify common bugs, identify relations among classes in a libraries, analyze defect data, discover reused patterns in source code and thus automate the development procedure. In general terms, using data mining practitioners and researchers can explore the potential of software engineering data and use the mining results in order to better manage their projects and to produce higher quality software systems that are delivered on time and on budget. In the following sections we discuss the main features of mining approaches that have been used in software engineering and how the results can be used in the

software engineering life cycle. We classify the approaches according to the software engineering tasks that they help and the mining techniques that they use.

4.1. Requirement elicitation and tracing

In this section we discuss how data analysis techniques can contribute to educe or trace system requirements. The works for requirement analysis refers to data mining in its broadest sense, including certain related activities and methodologies from statistics, machine learning and information retrieval.

4.1.1. Classification

A recent approach, presented in [6], has focused on improving the extraction of high level and low level requirements using information retrieval. More specifically, they consider the documents' universe as being the union of the design elements and the individual requirements and they map the problem of requirements tracing into finding the similarities between the vector space representations of high leveland low level requirements, thus reducing it into an IR task. As an expansion of this study, in [7], the authors focused on discovering the factors that affect an analysts' behavior when working with results from data mining tools in software engineering. The whole study was based on the verified hypothesis that *the accuracy of compute regenerated candidate traces affects the accuracy of traces produced by the analyst*. The study presents how the performance of tools that extract high level and low level requirements through the use of information retrieval, affects the time consumed by an analyst to submit feedback, as well as her performance. Results reveal that data mining systems exhibiting low recall result in a time consuming feedback from the analyst. In parallel, high recall leads to a large number of false positive thus prompting the analyst cut down large number of requirements, dimming recall. Overall reported results reveal that the analyst tends to balance precision and recall at the same levels.

4.1.2. Data summarization

From another perspective, text mining has been used in software engineering to validate the data from mailing lists, CVS logs, and change log files of open source software. In [11] they created a set of tools, namely SoftChange2, that implements data validation from the aforementioned text sources of open source software. Their tools retrieve, summarize and validate these types of data of open source projects. Part of their analysis can mark out the most active developers of an open source project. The statistics and knowledge gathered by SoftChange analysis has not been exploited fully though, since further predictive methods can be applied with regard to fragments of code that may change in the future, or associative analysis between the changes' importance and the individuals (i.e. were all the changes committed by the most active developer as important as the rest, in scale and in practice?).

4.2. Development analysis

This section provides an overview of mining approaches used to assist with development process.

4.2.1. Clustering

Text mining has also been used in software engineering for discovering development processes. Software processes are composed of events such as relations of agents, tools, resources, and activities organized by control flow structures dictating that sets of events execute in serial, parallel, iteratively, or that one of the set is selectively performed. Software process discovery takes as input artifacts of development (e.g. source code, communication transcripts, etc.) and aims to elicit the sequence of events characterizing the tasks that led to their development. In [12] an innovative method of discovering software processes from open source software Web repositories is presented. Their method contains text extraction techniques, entity resolution and social network analysis, and it is based on the process of entity taxonomies. Automatic means of evolving the taxonomy using text mining tasks could have been levied, so that the method lacks strict dependency on the taxonomy's actions, tools, resources and agents. An example could be the use of text clustering on the open software text resources and extraction of new candidate items for the taxonomy arising from the clusters' labels.

In [6], they used as text input the Apache developer mailing list. Entity resolution was essential, since many individuals used more than one alias. After constructing the social graph occurring from the interconnections between poster and replier, they made a social network analysis and came to really important findings, like the strong relationship between email activity and source code level activity. Furthermore, social network analysis in that level revealed the important nodes (individuals) in the discussions. Though graph and link analysis were engaged in the method, the use of node ranking techniques, like PageRank, or other graph processing techniques like Spreading Activation, did not take place.

4.2.2. Classification

Source code repositories stores a wealth of information that is not only useful for managing and building source code, but also provide a detailed log howthe source code has evolved during development. Information regarding the evidence of source code refactoring will be stored in the repository. Also as bugs are fixed, the changes made to correct the problem are recorded. As new APIs are added to the source code, the proper way to use them is implicitly explained in the source code. Then, one of the challenges is to develop tools and techniques to automatically extract and use this useful information.

In [2], a method is proposed which uses data describing bug fixes mined from the source code repository to improve static analysis techniques used to find bugs. It is a two step approach that uses the source code change history of a software project to assist with refining the search for bugs.

The first step in the process is to *identify the types of bugs* that are being fixed in the software. The goal is to review the historical data stored for the software project, in order to gain an understanding of what data exists and how useful it may be in the task of bug findings. Many of the bugs found in the CVS history are good candidates for being detected by statistic analysis, NULL pointer checks and function return value checks.

The second step is to *build a bug detector* driven by these findings. The idea is to develop a function return value checker based on the knowledge that a specific type of bug has been fixed many times in the past. Briefly, this checker looks for instances where the return value from a function is used in the source code before being tested. Using a return value could mean passing it as an argument to a function, using it as part of calculation, dereferencing the value if it is a pointer or overwriting the value before it is tested. Also, cases that return values are never stored by the calling function are checked. Testing a return value means that some control flow decision relies on the value. The checker does a data flow analysis on the variable holding the returned value only to the point of determining if the value is used before being tested. It simply

identifies the original variable the returned value is stored into and determines the next use of that variable. Moreover, the checker categorizes the warnings it finds into one of the following categories: – Warnings are flagged for return values that are completely ignored or if the return value is stored but never used. – Warnings are also flagged for return values that are used in a calculation before being tested in a control flow statement. Any return value passed as an argument to a function before being tested is flagged, as well as any pointer return value that is dereferenced without being tested .However there are types of functions that lead the static analysis procedure to produce false positive warnings. If there is no previous knowledge, it is difficult to tell which function does not need their return value checked. Mining techniques for source code repository can assist with improving static analysis results. Specifically, the data we mine from the source code repository and from the current version of the software is used to determine the actual usage pattern for each function. In general terms, it has been observed that the bugs cataloged in bug databases and those found by inspecting source code change histories differ in type and level of abstraction. Software repositories record all the bug fixed, from every step in development process and thus they provide much useful information. Therefore, a system for bug finding techniques is proved to be more effective when it automatically mines data from source code repositories.

4.3. Testing

The evaluation of software is based on tests that are designed by software testers. Thus the evaluation of test outputs is associated with a considerable effort by human testers who often have imperfect knowledge of the requirements specification. Data mining approaches can be used for extracting useful information from the tested software which can assist with the software testing. Specifically, the induced data mining models of tested software can be used for recovering missing and incomplete specifications, designing a set of regression tests and evaluating the correctness of software outputs when testing new releases of the system. A regression test library should include a minimal number of tests that cover all possible aspects of system functionality.

To ensure effective design of new regression test cases, one as to recover the actual requirements of an existing system. Thus, a tester has to analyze system specifications, perform structural analysis of the system's source code and observe the results of system execution in order to define inputoutput relationships in tested software.

5. CONCLUSION

This paper presents good understanding of existing research on mining SE data. This discuss the categorization of the existing research in this field into three major perspectives: data sources being mined, tasks being assisted, and mining techniques being used.

6. REFERENCES

[1] T. Xie, S. Thummalapenta, D. Lo, and C. Liu. Data ining for software engineering. *IEEE Computer*, 42(8):35–42, August 2009.

[2] C.C.Williams and J.K.Hollingsworth,Automatingmining of source code repositories to improve bug finding techniques, *IEEE Transactions on Software Engineering* **31**(6) (2005), 466–480.

[3] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail. CVSSearch: Searching through source code using CVS comments. In *Proceedings of the 17th Interna- tional Conference on Software Maintenance*, pages 364–374, Florence, Italy, 2001.

[4] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the 14th International Conference on Software Maintenance*, pages 190–198, Bethesda, Washington D.C., 1998.

[5] T. L. Graves, A. F. Karr, J. S. Marron, and H. Siy. Predicting fault incidence using software change history. *IEEE Trans. Softw. Eng.*, 26(7):653–661, 2000.

[6] J. Huffman Hayes, A. Dekhtyar and J. Osborne, Improving requirements tracing via information retrieval. In *Proceedings of the International Conference on Requirements Engineering*, 2003.

[7] J. Huffman Hayes, A. Dekhtyar and S. Sundaram, Text mining for software engineering: How analyst feedback impacts final results. In *Proceedings of International Workshop on Mining Software Repositories* (*MSR*), 2005.

[8] A. Mockus, D. M. Weiss, and P. Zhang. Understanding and predicting effort in software projects. In *Proceedings of the 25th International Conference on Software Engineering*, pages 274–284, Portland, Oregon, 2003.

[9] M. Acharya, T. Xie, J. Pei, and J. Xu. Mining API patterns as partial orders from source code: From usage scenarios to specifications. In *Proceedings of 6th joint meeting of the European Software Engineering Conference and the ACMSIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2007)*, pages 25–34, September 2007.

[10] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, and P. Devanbu. The promises and perils of mining git. In *roceedings of the 6th IEEE International Working Conference on Mining Software Repositories (MSR 2009)*,pages 1–10, May 2009.

[11] D. German and A. Mockus, Automating the measurement of open source projects. In Proceedings of the 3rd Workshop on Open Source Software Engineering, 25th International Conference on Software Engineering (ICSE03),2003.

[12] C. Jensen andW. Scacchi, Datamining for software process discovery in open source software development communities.In *Proceedings of International Workshop on Mining Software Repositories* (*MSR*), 2004.