ISSN:2320-0790

An International Journal of Advanced Computer Technology

Applying Data Mining on Execution Trace Log File for Improving Maintainability

Neha Koria[1], Meena Sharma[2]

Department of Computer Engineering, Institute of Engineering & Technology, Devi Ahilya University, Indore (M.P.) India,

[1]korian.me@gmail.com, [2]mrsharma.iet@dauniv.ac.in

Abstract: Software Engineering is a domain which has, in a short span of time, provided a vast scope for researchers. It has become an important part in software development process since people and organizations mostly rely on advanced software systems. Advanced software system requires the skills and directed efforts during the development phase and thus needs to be engineered. This has increased the competition for better software development which in turn has aroused an urgent need to emphasize on improving the software performance and quality. In our research, we apply data mining on software engineering data to enhance the maintainability of the system. The execution trace log files are used as the software engineering data. The mining algorithm identifies the most frequently accessed data from the logs. Analyzing the result of mining algorithm along with the logger levels in the log file the error prone area of the code is identified. Once the sensitive part of the code is recognized more emphasis on this part of the code would ensure minimum defects and errors in this code during various stages of software development life cycle thus improving the overall quality and performance of the software system.

Keywords: Software Engineering, Software Quality Attributes, Software Maintainability, Execution trace Log File (Log File), Data Mining, Data Mining Algorithm, Frequent Pattern Mining Algorithm, Log Parser, Logger Level.

I. Introduction

Software Engineering is an approach to develop software by applying our efforts in a correct and a systematic manner. This systematic manner aids to get the desired results in the best optimized way. It is mainly concerns with processes of software development, development of tools and methods to support software production, and getting results of the required quality within the schedule and budget [8]. It seems effortless in documentations but with practical implementations occur numerous problems. The growing business complexity, heightened competitiveness and market pressures on software development have raised the levels of software development. Since the race to bring more functionality to market, in less time, too easily compromises quality and thus the need to maintain the quality of the software system in the urgent need of the hour.

Software Quality Attributes are the measures that represent a system's anticipated behavior within the environment for which it was developed. Here we address one of the non-functional attribute that support in the enhancement of the performance and the quality of the software, the maintainability. Maintenance is the key concern for the software developers. Maintenance is assumed to be merely a defect fixing task, but actually approximately 80% of the efforts are used for non-corrective task. Thus putting in efforts to effectively maintain the software enhances the software engineering processes.

The diverse and immense data of software engineering provides excellent research platform. It can be broadly classified as sequences, graphs and text Mining Software Engineering data has recently emerged as a solution to meet the goal of improving the software productivity and quality due to two main reasons, the increasing profusion of such data and its usefulness in solving various real-world problems [1].

Thus with the intention of providing an easy and efficient technique towards achieving the desired quality of a software system, we propose a new approach through which we had put forward what we perceived, to be the most suitable and seemingly conceivable solution. The methodology followed is quite elementary yet very comprehensible. We have focused on the sequence data in software engineering for our research. The execution trace log files are used as the software engineering data on which we would apply the mining algorithm.

II. Background

Software Engineering is a well-researched area in the present scenario. Software Engineering is a growing field. It is often useful to think of it in three dimensions, each dimension being concerned with one particular aspect [6]. The first dimension comprises of the process, methods, and tools required in developing the software. The second dimension highlights the management techniques required to control software projects, to scrutinize the efficiency of the development processes successfully. The third dimension takes care of as to how the non-functional attributes of the software can be achieved.

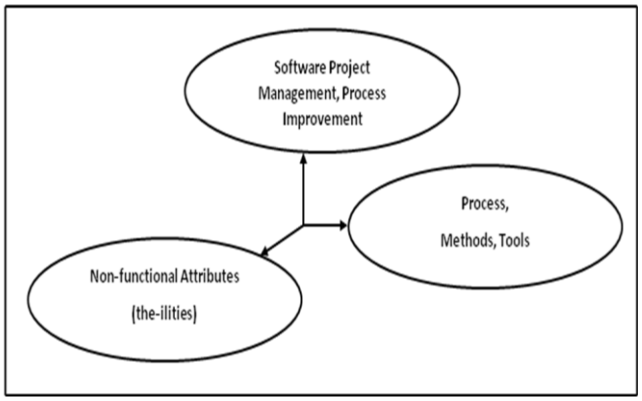
Software Project Management, Process Improvement
Process, Methods, Tools
Non-functional Attributes (the-ilities)

Figure 1. The 3-Dimensions of Software Engineering [6]

The earliest DACS state-of-art-report highlights a thorough survey of data mining techniques, with emphasis on applications to software engineering [2]. Using well established data mining techniques, practitioners and researchers can explore the potential of this valuable data in order to better manage their projects and to produce higher-quality software systems that are delivered on time and within budget [8].

Software Quality Attributes provide the means for measuring the fitness and aptness of a product. The desired attributes for a good software system are *Reliability, Efficiency, Security, Maintainability, Supportability, Performance, Usability, etc.* During the complete process of a software development, at each phase, efforts are made to achieve these attributes.

There are various algorithm and approaches advised to apply on the software engineering data. For mining the sequences we can use Iterative pattern mining, Temporal rule mining, Sequence-diagram and FSM mining, Sequence association rule mining, The graphs can be mined using other approaches as Discriminative graph mining and Graph classification. Text mining algorithms include text clustering, classification, and matching. The specific type of algorithm can be applied to the indented software engineering data [1].

Software maintenance is considered a very important and complex stage in software lifecycle normally consuming 50-70% of the total effort allocated to a software system [10], [11]. Various attempts have been made to enhance the software maintenance by applying different approaches of data mining on assorted software engineering data.

III. Motivation

Modern software engineering is an inherently human-centric activity. From requirements, architecture and design through development, testing and maintenance, the inputs, processes and outputs are primarily created, evaluated and performed by humans [5]. Through our research, we intend to reduce this human effort in the maintenance process. Taking into consideration the preventive maintenance, we suggest a semi-automated solution that analyzes the log file to identify the frequently used section of the code. This is used with the logger level to identify the sensitive code that would, in future, create defects or faults in the system. The result helps the maintenance programmers and developer to pay more attention towards this sensitive code instead of going through the complete code.

The execution trace log files are being maintained for each software system. Commonly these are accessed when an error occurs in the system. The motivation to analyze these log files is to increase their utility and to use them to enhance software quality.

IV. Present Scenario

In the present scenario, the maintenance programmer deals with the error that occurs in the system. The steps involved are:

- Investigate each request.
- Confirm it, by reproducing the situation and checking its validity.
- Scrutinize it to recommend a solution,
- Document the request and the solution proposal,
- Finally, obtain all the required authorizations to apply the modifications.

There are a number of processes, activities and practices that are unique to maintainers [9], for example:

- Transition: a controlled and synchronized sequence of activities during which a system is transferred gradually from the developer to the maintainer.
- Service Level Agreements (SLAs) and specialized (domain-specific) maintenance contracts negotiated by maintainers.

- Modification Request and Problem Report Help Desk: a problem-handling process used by maintainers to prioritize documents and route the requests they receive.

V. Architecture

The generalized approach for mining of Software Engineering data involves five main steps [1]. Initially the software engineering data is being collected and investigated. This is then accompanied by determining the software engineering task to assist. These two steps work in parallel. The next step is the preprocessing of the data. Preprocessing involves extracting the relevant data from the raw software engineering data. This data is further preprocessed by cleaning and appropriately formatting it for the mining algorithm.

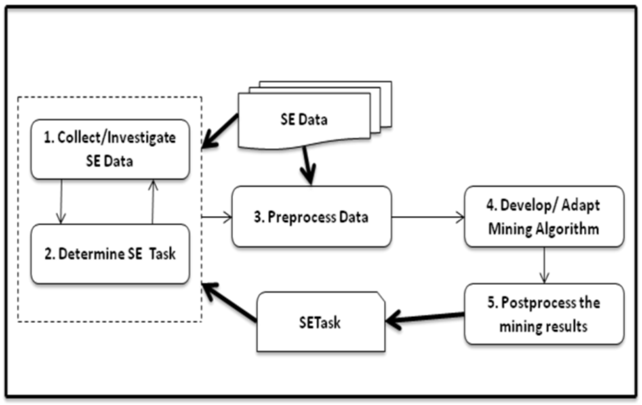
SE Data
1. Collect/Investigate SE Data
2. Determine SE Task
3. Preprocess Data
4. Develop/ Adapt Mining Algorithm
5. Postprocess the mining results
SETask

Figure 2. The Architecture for Mining Software Engineering Data [1]

In the next step a mining algorithm is being adopted which satisfies the requirements analyzed during the first two steps. Once the data is preprocessed into the database table and dataset are created any mining algorithm can be used to mine the data according to the software engineering task. Finally the result from the previous step is transformed into proper format that would assist the software engineering task.

VI. Proposed Architecture

Initially the logs as generated by the execution of the software system are saved in the file with a specified name format. These are then preprocessed using the log parser, to make them ready to be used with data mining algorithm. The log parser parses these into a database table. It specifically stores the logger level, package name, class name, method name, and the log message. The database table represents each entry of the log file in the form of database transactions. This provides handy data for applying data mining algorithm, instead of having an overhead of repeatedly parsing the log file. Once the database table is created we can identify distinct packages, classes and methods. These are then mapped to an individual unique identifier.

The mapping is done since it would be a tedious job to read the long names of packages, classes and methods while applying the mining algorithm.

Now to make the data, mining algorithm ready, after the mapping is completed, we combine the data fields to form item sets. The package, class, methods and logger level forms the complete item steps.

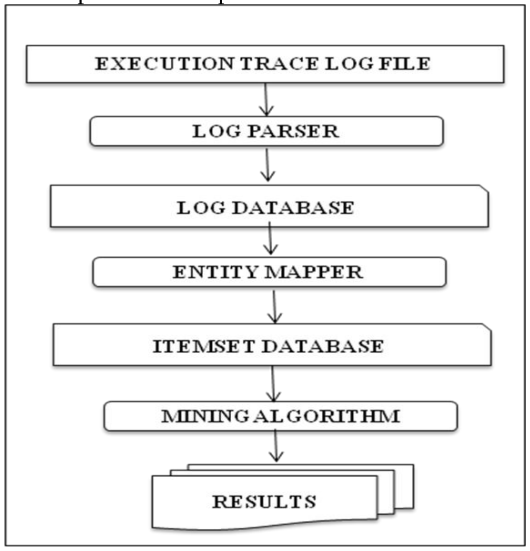

Figure 3. The Proposed Architecture

This completes the preprocessing task. Frequent Pattern Mining algorithm is used for mining these item sets. This enables us to identify the item set that occur frequently in the database. On identifying the frequent item set, we post process the results into the software engineering data to assist software engineering task. We use the logger level in the log file to identify the error prone code. The final results are the names of the packages, classes and methods that occur repeatedly. These help the maintenance and developer teams to emphasize on this section of the code more while making changes or updating the software system in future.

VII. Assumption

During the implementation of the proposed architecture the following assumptions have been stipulated:

1. The execution trace log file is saved in a specified name format.
2. The date wise logs are maintained.
3. All logger levels are enabled.
4. Each line in the log file is in the same format,
5. All classes and methods have sufficient logging.
6. All error scenarios are properly logged.
7. The item-sets generated contain fixed number of items in each transaction.

VIII. Implementations

In the implementation of the proposed architecture we use the execution trace log file of a web application which is account receivables module for a telecom company. This application manages the customer (telecom subscribers) financial accounts and is used telecom service providers (e,g At&T etc)

This application handles critical actions related to customer financial transactions like payment, payment back-out, credit, debit, dispute, etc. Such a system needs to be full proof and huge efforts go into maintenance of such critical module.

We have taken log files where multiple users are using the system in production.

A. Log Parser

The software engineering domain experts may lack the knowledge required to design the mining algorithms whereas the data mining researches may lack the background to understand the applicability of data mining in software engineering. Thus it is required to provide an easy to use data format to the data mining experts to apply their research on the software engineering data.
The Log Parser helps to solve the above mentioned problem. It parses the log file by reading the individual line of the execution trace log file and tokenizes it into the fields and records of the database table. This correctly stores the logger level, package name, class name, method name, and the log message with respect to a particular execution trace. This data is then preprocessed to make it more effective to apply the mining algorithm directly.

The following shows the entries in the execution trace log file which are then parsed into the database table.

```
[DEBUG] [Mar 13 01:37:16] [Thread-19] [mentor.revmgmt.AR1GeneralLogHandler.mentor.ar.datalayer] [AbstractPM.<init>] Setting operatorId - batch: 0

[DEBUG] [Mar 13 01:37:16] [Thread-18] [mentor.revmgmt.AR1GeneralLogHandler] [PartitionHelperBaseCustomization.setTransactionLogPartitions] PARTITION_PAR == 1
```

```
mysql> select * from logdata limit 2 \G
************ 1. row ************
id: 1
date: 2012-03-13 01:37:16
logger_level: DEBUG
package_name:mentor.revmgmt.AR1GeneralLogHandler.mentor.ar.datalayer
class_name: AbstractPM
method_name: <init>
log_message: SettingoperatorId-batch:0

************ 2. row ************
id: 2
date: 2012-03-13 01:37:16
logger_level: DEBUG
package_name:mentor.revmgmt.AR1GeneralLogHandler
class_name: PartitionHelperBaseCustomization
method_name: setTransactionLogPartitions
log_message: PARTITION_PAR==1

2 rows in set (0.00 sec)
```

B. Mining Algorithm

Different data mining algorithms produce patterns that reflect different levels of information, and which algorithm to choose depends on the specific Software Engineering task's mining requirements. Here we need to identify the most accessed part of the code and thus use the frequent pattern mining algorithm. This algorithm helps us to identify the frequently occurring flow of packages, classes and method from the execution trace log file.

We have used the Apriori Algorithm for mining frequent item-sets. It uses the property that a subset of a frequent item-set must also be a frequent item-set. The invocation of a method is dependent on the invocation of the class and the invocation of a class depends on the package that contains it. Thus, if a method is identified as frequently occurring the concerned class and package would also be accessed frequently. The package and class corresponds to a specific workflow of the software application. Thus as a result of the mining algorithm we obtain the workflows that occur frequently in the execution of the system. Analyzing the result with the logger levels we identify the erroneous workflow that needs to be taken care of the most.
For implementation we have considered the execution trace log file of more than 45K transactions. This file is scanned, parsed into transactions and then mapped into item-sets. The package name, class name, method name and logger levels are mapped as Pi, Ci, Mi and Li respectively. To highlight how the proposed system works, here we consider a small set of 10 item-sets. The table below represents the item-sets as generated by processing the execution trace log file.

ID	Item-Sets
Id1	P1, C1, M1, L1
Id2	P2 ,C2, M2, L1
Id3	P1, C1, M1, L1
Id4	P1, C3, M3, L1
Id5	P2, C4, M4, L1
Id6	P2, C5, M5, L1
Id7	P2, C5, M5, L1
Id8	P3, C6, M6, L1
Id9	P1, C1, M1, L1

Applying the Apriori algorithm on the above set of it generates the frequent item-sets. The frequent item-sets of size 1: [p1], [p2], [c1], [c5], [m1], [m5], [l1]. These help in generating the candidate item-sets for size-2. On pruning those we get the next frequent item-sets, [c1, l1], [c1, m1], [p1, c1], [c5, m5], [p2, c5], [m1, l1], [m5, l1], [p1, l1], [p2 , l1], [p1, m1], and, [p2, m5]. The items in the above item-sets are used to generate the next candidate set. The frequent item-sets of size-3 obtained are [c1, m1, l1], [p1, c1, l1], [p1, c1, m1], [p2, c5, m5], [p1, m1, l1], and [p2, m5, l1]. Finally these combine to form the item-sets of size-4. The final frequent item-set generated is [p1, c1, m1, l1].

Since we need the frequent workflows in the application execution, we have focused on the last set of the frequent item-sets generated. This set represents the complete workflow, containing the package, class, method and the logger level. The item-sets are mapped again in terms of software engineering data. This is achieved by replacing the unique identifier by the original names of the corresponding package, class, and method along with the logger level. This is done to generate the final results that would aid the maintenance and developers to identify the part of the code that needs to be more emphasized.

Since the final frequent item-set is [p1, c1, m1, l1]. These are replaced by the original name with respect to the software engineering data. P1 is the package name alias for <<mentor.revmgmt.AR1GeneralLogHandler.mentor.ar.data layer>>. C1 represents <<AbstractPM>> class name. M1 is method name alias for <<init>>. L1 represents <<Debug >> logger level. These are replaced and the final output is generated as below:

The frequent execution workflow considering 10 transactions from execution trace log file is as below:

Package Name :
mentor.revmgmt.AR1GeneralLogHandler.mentor.ar.dat alayer

Class Name:
AbstractPM

Method Name:
init

Logger_level:
Debug

IX. Conclusion

We have aimed to enhance the maintenance process for a software system. The research work intends to increase the utilization of the execution trace log files. The execution trace log file is generated by the activities performed by the end user in real time. Applying mining algorithm on this data to enhance the quality of the software application proves beneficial since these are the transaction performed by the end user. This would result in data error free interactions of the end user with the software application.

The usage mining algorithm reduces the manual efforts involved in analyzing the log files when an error is detected. The sensitive area of the code is identified; the developers and the maintenance programmer can work towards improving this part of the code only. The functionalities of the modules are checked continuously and not only when an error occurs. We use a preventive maintenance technique to improve the quality of the software system.

X. References

[1] Tao Xie, Suresh Thummalapenta, David lo, Chao Liu,"*Data Mining for Software Engineering*", IEEE Computer, August 2009, pp. 55-62.

[2] Manoel Mendonca, "*Mining Software Engineering Data: A Survey, DACS State-of-the-Art Report*", University of Maryland, Department of Computer Science, Nov 1999.

[3] A.V.Krishna Prasad, Dr.S.Rama Krishna, "*Data Mining for Secure Software Engineering - Source Code Management Tool Case Study*", International Journal of Engineering Science and Technology, Vol. 2 (7), 2010, 2667-2677.

[4] NATO Science Committee, "*Software Engineering*", Report on a conference, Garmisch, Germany

[5] Raymond P.L. Buse, Caitlin Sadowski, Westley Weimer, "*Benefits and Barriers of User Evaluation in Software Engineering Research*", OOPSLA'11, Portland, Oregon, USA, October 22–27, 2011

[6] Prof. D. Vernon, "*Course Notes*", Khalifa University, http://www.vernon.eu/courses/David_Vernon_Software_Engineering_Notes.pdf

[7] Software Maintenance and Re-engineering, CSE2305 Object-Oriented Software Engineering,

http://www.csse.monash.edu.au/~jonmc/CSE2305/Topics/13.25.SWEng4/html/text.html

[8] Tao Xie, Jian Pei, Ahmed E. Hassan, "*Mining Software Engineering Data*".

[9] Alain April, Jane Huffman Hayes, Alain Abran, Reiner Dumke, "*Software Maintenance Maturity Model (SMmm): The software maintenance process model*", J. Softw. Maint. And Evolution 2004.

[10] Pigoski T.M., Practical Software Maintenance: Best Practices for Managing your Software Investment, Wiley Computer Publishing, 1996.

[11] Sommerville, Software Engineering, 6th ed., Harlow,Addison-Wesley, 2001.