Volume 4, Issue 5, May 2014 **ISSN: 2277 128X**

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Mining Frequent Patterns From Bug Repositories

Kiran Kumar B.
Dept. of IT,
KITS, Warangal,
INDIA

Jayadev Gyani
Dept. of CSE,
JITS, Narsampet,
INDIA

Narsimha G.
Dept. of CSE,
JNTUH College of Engineering,
INDIA

Abstract- Mining bug repositories is an emerging area of research. Bug repositories are major source of knowledge. Different data mining algorithms can be applied on these repositories to extract useful and interesting patterns. Finding frequently occurring bugs helps the developers to plan to overcome the occurrences of the bugs in future releases. In this paper we propose a method to mine frequent patterns from the bug repositories. Our method considers bug summary information extracted from the bug repository on which text mining techniques are applied to find frequent patterns from the bug summary.

Keywords: Frequent Patterns, Bug repositories, Text Mining.

I. INTRODUCTION

The goal of Software Engineering (SE) is to improve software productivity and quality. Mining software engineering data has recently emerged as a promising way to meet this goal. SE data can be used to gain understanding of software development, predict, plan, understand various aspects of a project, support future development and project management activities. Types of SE data consists of software requirements specification documents, configuration management data, source code, compiled code with execution traces, bug repositories, mailing lists. Due to the increasing abundance of such data, different data mining algorithms like frequent pattern mining, classification, clustering can be applied on this data to discover interesting patterns.

In this paper, we propose a method to mine frequent itemsets from the bug repositories. Bug or defect is reported by the quality analysts, test engineers, and users of the software product. These reported bugs are stored in the bug repositories which are managed and tracked by different tools such as Bugzilla, JIRA etc. Bug database can be used to extract patterns, which can be used by the software developers to eliminate such bugs in the code in the future projects. In this paper, we used bug_summary attribute of the bug database and text and frequent pattern mining algorithms are applied on bug database to find frequent patterns. These frequent patterns can be used to find the associations among the terms appearing in the bug report. Our paper is organized as follows. Section II focuses on related work. Section III, discusses proposed work. Experimentation and results are shown in section IV. Section V shows conclusions and future scope and references are given in Section VI.

II. RELATED WORK

Bug repositories can be used to mine different patterns. Jaweria Kanwal and Onaiza Maqbool proposed a method using support vector machines for managing bug repositories through bug report prioritization [4]. Naresh Kumar Nagwani, Ashok Bhansali proposed a model to predict software bug complexity using bug estimation and clustering [9]. Philipp Schugerl, Juergen Rilling, Philippe Charland proposed a method for quality assessment in bug repositories [6]. Leon Wu, et.al. developed a tool BUGMINER, which is used to derive useful information from historic bug report database using data mining. This information is used to do completion check and redundancy check on a new or given bug report, and to estimate the bug report trend using statistical analysis [5]. Michael Gegick, Pete Rotella, Tao Xie developed a approach that applies text mining on natural-language descriptions of bug reports to train a statistical model on already manually-labeled bug reports to identify security bug reports that are manually-mislabeled as not-security bug reports [8]. Security engineers can use the model to automate the classification of bug reports from large bug databases to reduce the time that they spend on searching for security bug reports. Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik and Jiasu Sun proposed an approach for detecting duplicate bug reports using natural language and execution information [11]. When a new bug report arrives, its natural language information and execution information are compared with those of the existing bug reports. Then, a small number of existing bug reports are suggested to the triager (A triager is a person who decides whether a report should be worked on and who should work on it) as similar bug reports to that of new bug report. Amir Michail, Tao Xie proposed a method to help users avoid bugs in GUI applications [3]. Users would use the application normally and report bugs that they encounter to prevent anyone from encountering those bugs again. Syed Nadeem Ahsan et.al presented an approach to mine the bug-fix effort data from the history of developer's bug fix activities [10]. Ahmed Lamkanfi et.al. proposed a method for predicting the severity of a reported bug by analyzing its textual description [2]. Adrian Schroter, Nicolas Bettenburg, and Rahul Premraj identified the support of usefulness of stack traces to ECLIPSE developers by examining key patterns in the resolution of bug reports that contained stack traces [1]. Marco D'Ambros, Michele Lanza and Romain Robbes presented a benchmark for defect prediction, in the form of a publicly available data set consisting of several software systems, and provide an extensive comparison of the explanative and predictive power of well-known bug prediction approaches [7].

All the above methods are focused on application of text mining, classification and clustering techniques on the bug databases. In this paper, we are using text mining and apriori algorithm to extract frequent patterns from the bug repositories.

III. PROPOSED WORK

Bug repository is the database which stores the information relating to the bugs. Typical attributes of bug repositories are: BugID (A unique identification number), EntryTime (Time the bug is reported), ModifiedTime (Time the bug is modified), Type (Type of the bug), Status (Typical values for status are NEW, DUPLICATE, FIXED, RESOLVED etc.), Severity (Bug complexity), Summary (Description of the bug) etc. From this database, we are considering bug summary attribute as source of our work. The process involved in the proposed method is depicted in Fig.1.

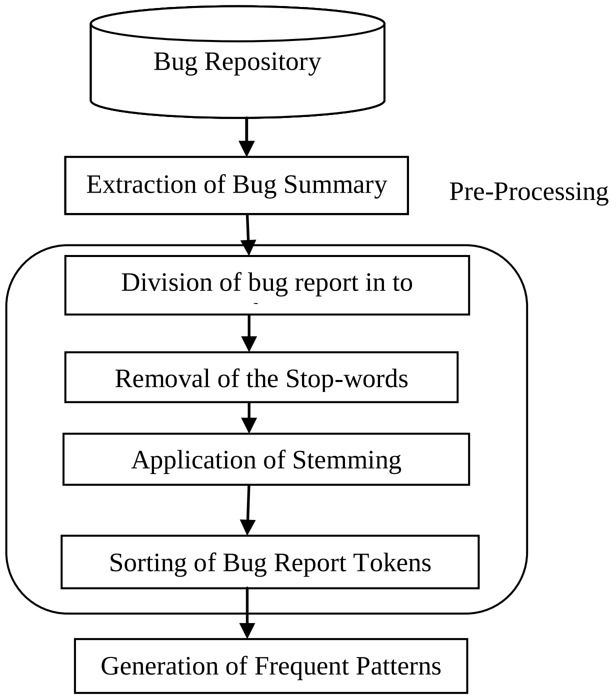
Bug Repository
Extraction of Bug Summary
Pre-Processing
Division of bug report in to
Removal of the Stop-words
Application of Stemming
Sorting of Bug Report Tokens
Generation of Frequent Patterns

Fig.1 Frequent Pattern Mining from Bug Database.

Table I: Sample Bug Database.

ID	**SUMMARY**
1	"Does not make Toast"
2	"Innodb autoincrement stats los on restart"
3	"SHOW SESSION VARIABLES LIKE 'x' != SELECT @@SESSION.x"
4	"SQLDescribeParam returns the same type information"
5	"Lost Data with delayed insert / mysqlhotcopy or lock and flush"
6	"Wrong case sensitivity for table names when in ""ANSI"" mode"
7	"Parser allows naming of PRIMARY KEY"
8	"changing hostname confuses master or slave"
9	"Can't use @user_variable as FETCH target in stored procedure"
10	"fields-escaped-by only allows single character separator!"
11	"wrong value for stmt->field_count"
12	"Error 2003 - can't connect to MYSQL server on localhost "

13	"Rollback in stored procedure doesn't close cursor"
14	"Incomplete lower_case_table_names=2 implementation"
15	"max_heap_table_size affects creation of disk-based temporary tables"
16	"Floating point conversions are inconsistent."
17	"Traditional mode: CAST allows invalid value"
18	"Traditional mode: BIGINT range not correctly delimited"
19	"Traditional: MOD with 0 divisor should return error"
20	"Traditional: FLOAT and DOUBLE out-of-range values are accepted"

.

The description of the each bug summary is treated as one text document. Sample bug summary of MySQL bug repository is shown in Table I The bug description is first divided into number of tokens and some of the tokens (stop-words) are not useful in the analysis. Stop-words are the words, which have low significance in the analysis of the bug reports and so these are eliminated. Stop-words can be a, an, and, are etc. The removal of stop words doesn't make any difference in the programming. For example, when a user logged into the system with incorrect credentials an error message will be displayed as "Your access is denied". By this message, the user will come to know that he/she has entered the incorrect credentials. In the above contest, the system displays an error message such as "Access denied", the user finds the same meaning as above. From this case, the observed result is the presence of tokens "your" and "is" doesn't make any change in the process of accessing the system. The bug database after removal of stop-words is shown in the Table II.

Table II. Database after removal of stop-words.

ID	**SUMMARY**
1	make Toast
2	Innodb autoincrement stats los restart
3	SHOW SESSION VARIABLES SELECT SESSION.x
4	SQLDescribeParam returns type information
5	Lost Data delayed insert mysqlhotcopy lock flush
6	Wrong case sensitivity table names ANSI mode
7	Parser naming RIMARY KEY
8	changing hostname confuses master slave
9	user_variable FETCH target stored procedure
10	fields-escaped-by single character separator
11	wrong stmt->field_count
12	Error 2003 – connect MYSQL server localhost
13	Rollback stored procedure 700oesn't close cursor
14	Incomplete lower_case_table_names2 implementation
15	max_heap_table_size affects creation disk-based temporary tables
16	Floating point conversions inconsistent.
17	Traditional mode: CAST invalid
18	Traditional mode: BIGINT range correctly delimited
19	Traditional: MOD 0 divisor return error
20	Traditional: FLOAT DOUBLE out-of-range values accepted

The tokens left after removal of stop-words are passed to the stemming process. Stemming is the process of trimming the derived words to their root word. The stem need not be identical to the actual root word, but should definitely match the same stem. For example, consider the words read, reads, reading, readable, all these words are derived from the root word "read". When applying the process of stemming to these words, they are replaced with its stem word "read" that means all the derived words are unified to its base word. In this paper, we used Porter Stemmer algorithm to stem the bug tokens. These stemmed tokens are sorted in their lexicographic order in order to apply Apriori algorithm. The sorted stemmed tokens are shown in Table III.

Table III. Stemmed tokens

ID	ITEMS
1	make,toast
2	autoincrement,innodb,los,restart,stats
3	select,session,session,show,variables,x
4	informateon,returns,sqldescribeparam,type
5	data,delaied,flush,insert,lock,lost,mysqlhotcopi
6	ansi,case,mode,names,sensitiveti,table,wrong
7	kei,nameng,parser,primary
8	changing,confuses,hostname,master,slave
9	fetch,procedure,stored,target,user,variable
10	by,character,escaped,fields,separater,single
11	count,field,stmt,wrong
12	connect,error,localhost,mysql,server
13	close,cursor,doesnt,procedure,rollback,stored
14	case,implementateon,incomplete,lower,names,table
15	affects,based,creation,disk,heap,max,size,table,temporary
16	conversions,floateng,inconsistent,point
17	cast,invalid,mode,traditional
18	bigint,correctli,delimited,mode,range,traditional
19	divisor,error,mod,return,traditional
20	accepted,double,float,of,out,range,traditional,values

Mining frequent patterns is one of the fundamental operations in data mining applications for extracting interesting patterns from databases. Let I= $\{i_1, i_2 \ldots i_n\}$ be a set of items. Let D be a set of database transactions where each transaction T is a set of items and ||D|| be the number of transactions in D. Given X= $\{i_j \ldots i_k\} \subseteq$ I ($j \leq k$ and $1 \leq j, k \leq n$) is called a pattern. The support of a pattern X in D is the number of transactions in D that contains X. Pattern X will be called frequent if its support is no less than a user specified minimum support threshold. We used Apriori algorithm on the stemmed tokens to find the frequent tokens appearing in the bug database. The summary of frequent items generated with minimum support of 10 is shown in Table IV.

Table IV: Frequent itemsets

Project: MySQL

Frequent Item	Support
federated,server	10

error,mysql,server	10
procedure,stored,variable	10
case,lower,names	10
case,lower,table	10
case,names,table	15
case,database,sensitive	10
lower,names,table	10
index,optimizer,queri	10
db,install,mysql	10
defaults,mysqld,option	10
option,install,mysql	10
case,lower,names,table	10

IV. EXPERIMENTATION AND RESULTS

We experimented with bug reports of Firefox project, which are available at https://**bugzilla**.mozilla.org/. We considered Build Configuration component of the Core product and the results are shown in fig 6. We also experimented with the bug reports of Eclipse project, which are available at *https://bugs.**eclipse**.org/bugs/*. We considered bug reports of Chart component and Report Designer component of Business Intelligence and Reporting Tools (BIRT) Product and the results are shown in Table V and Table VI respectively.

Table V: Frequent Patterns of Firefox bug reports.

Project : Firefox, Product:Core
Componetnent: Build Config

Frequent Item	**Support**
android,mobile	10
android,build	11
in,makefile	43
in,configure	10
config,mk	15
c,build	12
error,make	10
error,build	27
linux,build	10
make,build	19
make,target	13
work,doesn’t	13
mk,rules	15
build,doesn’t	17
build,files	21
build,add	15
build,moz	52
build,system	33
build,fails	19
build,move	21
build,failure	12
build,js	16
build,firefox	10
build,mach	26
rule,target	10
files,moz	12
check,configure	12
builds,windows	13
moz,move	21
clobber,bug	10
js,src	16

Table VI: Frequent Patterns of Eclipse bug reports.

Project: Eclipse, Product: BIRT
Component: Chart

Frequent Item	Support
axis,x	7
axis,chart	13
chart,type	7
chart,legend	5
chart,area	7
chart,bar	7
chart,label	7
chart,builder	13
chart,series	8
chart,data	16
chart,birt	10
chart,interactiveti	5
chart,line	5
chart,set	5
chart,support	6
chart,format	5
chart,bps	5
data,points	7

Project: Eclipse, Product: BIRT
Component: ReportDesigner

Frequent Items	Support
birt,table	9
birt,report	13
birt,designer	7
birt,binding	6
birt,column	5
birt,data	5
editor,property	10
editor,report	6
preview,report	5
report,designer	9
report,file	5
report,design	9
report,layout	6
report,item	6
report,items	5
report,data	5
report,show	6
report,template	5
layout,view	6
outline,view	6
set,data	7
binding,column	6
binding,data	10
column,data	6
builder,expression	8
error,message	5
tct,tvt	7

V. CONCLUSIONS

The frequent terms generated from the bug databases can be used to know the type of bugs generated and are used by the developers to plan the code in the future project, which can reduce the maintenance cost. These patterns can be used to cluster the bug records based on the similarity of tokens and also used to suggest the resolution process for similar bugs.

REFERENCES

1. Adrian Schroter, Nicolas Bettenburg, Rahul Premraj, "Do Stack Traces Help Developers Fix Bugs?" MSR-2010 pp:118-121
2. Ahmed Lamkanfi_, Serge Demeyer_y, Emanuel Gigery, Bart Goethalsz, "Predicting the Severity of a Reported Bug" MSR-2010, pp: 1-10
3. Amir Michail, Tao Xie "Helping Users Avoid Bugs in GUI Applications", ICSE'05, May 15-21, 2005, St. Louis, Missouri, USA
4. Jaweria Kanwal, Onaiza Maqbool, "Managing Open Bug Repositories through Bug Report Prioritization Using SVMs", Proceedings of the 4th International Conference on Open-Source Systems and Technologies (ICOSST 2010) 22-24 December, 2010 © ICOSST 2010, pp: 1-7
5. Leon Wu, Boyi Xie, Gail E. Kaiser, and Rebecca J. Passonneau, "BUGMINER: Software Reliability Analysis Via Data Mining of Bug Reports" *SEKE, Knowledge Systems Institute Graduate School, (2011),* pp: 95-100
6. Philipp Schugerl, Juergen Rilling, Philippe Charland, "Mining Bug Repositories – A Quality Assessment", CIMCA.2008, pp:1105-1110
7. Marco D'Ambros, Michele Lanza, "An Extensive Comparison of Bug Prediction Approaches" MSR 2010, pp: 31-41
8. Michael Gegick, Pete Rotella, Tao Xie, "Identifying Security Bug Reports via Text Mining: An Industrial Case Study" MSR 2010, pp 11-20
9. Naresh Kumar Nagwani, Ashok Bhansali, "A Data Mining Model to Predict Software Bug Complexity Using Bug Estimation and Clustering", 2010 International Conference on Recent Trends in Information, Telecommunication and Computing, pp:13-17
10. Syed Nadeem Ahsan, Muhammad Tanvir Afzal, Safdar Zaman, Christian Gutel, Franz Wotawa, "Mining Effort Data From The OSS Repository of Developer's Bug Fix Activity", Journal of IT in Asia, Vol 3 (2010) pp:67-80
11. Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik and Jiasu Sun, "An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information" ICSE'08, May 10–18, 2008, Leipzig, Germany