The Way Ahead for Bug-fix time Prediction

Meera Sharma
Department of Computer Science
University of Delhi
Delhi, India
meerakaushik@gmail.com

Madhu Kumari
Department of Computer Science
University of Delhi
Delhi, India
mesra.madhu@gmail.com

V.B.Singh
Delhi College of Arts & Commerce,
University of Delhi
Delhi, India
vbsinghdcacdu@gmail.com

***Abstract*— The bug-fix time i.e. the time to fix a bug after the bug was introduced is an important factor for bug related analysis, such as measuring software quality or coordinating development effort during bug triaging. Previous work has proposed many bug-fix time prediction models that use various bug attributes (number of developers who participated in fixing the bug, bug severity, bug-opener's reputation, number of patches) for predicting the fix time of a newly reported bug. In this paper, we have investigated the associations between bug attributes and the bug-fix time. We have proposed two approaches to apply association rule mining. In the first approach, we have used Apriori algorithm to predict the fix time of a newly coming bug based on the bug's severity, priority summary terms and assignee. In second approach, we have used *k*-means clustering method to get groups of correlated variables followed by association rule mining inside each cluster. We have collected 1,695 bug reports of three products namely AddOnSDK, Thunderbird and Bugzilla of Mozilla open source project to mine association rules. Results show that for given set of bug attributes, we can predict the bug-fix time for newly coming bugs which will help in software quality improvement. A large number of association rules having high confidence and support with higher severity and priority as antecedents and short bug-fix time as consequent show that more important bugs are fixed without any delay. This information is useful in determining software quality. We also observe that our approach for bug-fix time prediction will be helpful in bug triaging by assigning a bug to the most potential and experienced assignee who will solve the bug in minimum time period. This will again help in software quality improvement. In nutshell, we can say that association rule mining based bug-fix time prediction can help managers to improve the software development process.**

Keywords—Bug-fix time; Apriori algorithm; Association rule mining; k-means Clustering

I. INTRODUCTION

Bug-fix time prediction is useful in software quality prediction [1] or in coordinating effort during bug triaging to maintain the software systems effectively [2]. In literature efforts have been made to construct many bug-fix time prediction models, based on machine learning algorithms, on both open source and commercial projects [3-5].

A bug report is characterized by many attributes like summary, priority, severity and assignee. The textual description of a bug reported by users is known as its summary. Bug priority tells about the importance and order of bug fixing in comparison of other bugs with P1 as the highest and P5 as the lowest priority. The bug severity can be defined as: (i) the impact of bugs on the functionality of the software (business point of view) (ii) the impact of bugs on developer means how much time a bug will take in fixing. In this paper, we consider the bug severity from the business point of view. It is measured according to different levels from 1(blocker) to 7(trivial). These levels are defined in repositories as 1 for highest and 7 for lowest. Assignee is a person to whom the bug is assigned to work on.

To the best of our knowledge, no approach has been proposed till now to mine association rules among different bug attributes for bug-fix time prediction. In software development this can help the managers to improve the process in terms of cost and resources. We have proposed an approach for bug fix time prediction based on other bug attributes namely summary terms, priority, severity and assignee. We have applied association rule mining by using Apriori algorithm and *k*-means clustering followed by Apriori algorithm. For experiment of the proposed approach we have used 1,695 bug reports of AddOnSDK, Thunderbird and Bugzilla products of Mozilla open source project. Association rule mining was first explored by [7] which is the base of our prediction method.

In a database, the interesting correlations, frequent patterns, associations or casual structures among the attributes can be discovered by using association rule mining. Let C is a database of transactions and each transaction T is a set of items. An association rule is an expression $A \Rightarrow D$, where A is called antecedent and D is called consequent. $A \Rightarrow D$ reveals that whenever a transaction T contains A, then T also contains D with a specified confidence and support. The confidence of a rule is defined as percentage/fraction of the number of transactions that contain $A \cup D$ to the total number of transactions that contain A. It is a measure of the rule's strength or certainty [8]. Support of a rule is defined as the percentage/fraction of transactions that contain $A \cup D$ to the total number of transactions in the database. It corresponds to statistical significance or usefulness of the rule. Minimum support count is defined as the number of transactions required for an item set to satisfy minimum support. Association rule mining generates all association rules that have a support greater than minimum support $min.Supp(A \Rightarrow D)$, in the database, i.e., the rules are frequent. The rules must also have confidence greater than minimum confidence $min.Conf(A \Rightarrow D)$, i.e., the rules are strong.

In a wide range of science and business areas association rule mining can be applied successfully. Several performance studies have resulted in better accuracy for associative classification than state-of-the-art classification methods [9-18].

Clustering is a partitioning method in which a group of data points is partitioned into a small number of clusters. In *k*-means clustering algorithm, the function *k*-means partitions data into *k* mutually exclusive clusters, and returns the index of the cluster to which it has assigned each observation. Unlike hierarchical clustering, *k*-means clustering operates on actual observations (rather than the larger set of dissimilarity measures), and creates a single level of clusters. The distinctions mean that *k*-means clustering is often more suitable than hierarchical clustering for large amount of data.

The successful use of association rule mining in various fields motivates us to apply it to the open source software bug data set [9-18].

The organization of rest of the paper is as follows. Section 2 gives the description and preprocessing of data. Section 3 describes the model building. Section 4 presents the results. Section 5 discusses about related work. Section 6 tells about the threats to validity and finally section 7 concludes the paper with future research directions.

II. DATA SETS DESCRIPTION AND DATA PREPROCESSING

We collected bug reports from Bugzilla bug tracking system with status "verified", "resolved" and "closed" and resolution "fixed" because only these types of bug reports contain the consistent information for the experiment. We have compared and validated the collected bug reports against general change data (i.e. CVS or SVN records). Number of bug reports collected in the observed period is given in table I.

TABLE I. PRODUCTWISE NUMBER OF BUG REPORTS

Product	Bug reports	Observation period
Bugzilla	964	Sept. 1994-June 2013
Thunderbird	115	Apr. 2000-Mar. 2013
Add-on SDK	616	May 2009-Aug. 2013

In order to apply association rule mining, we have quantified different bug attributes namely severity, priority, summary, assignee and fix time.

We have preprocessed the bug summary attribute to extract terms in RapidMiner tool [19] with the help of following steps:

Tokenization: the process of breaking a stream of text into words, phrases, symbols, or other meaningful elements called tokens is called '*tokenization'*. We have considered a word or a term as a token.

Stop Word Removal: words which are commonly used in the text but do not carry useful meaning like prepositions, conjunctions, articles, verbs, nouns, pronouns, adverbs, adjectives are called stop words. We have removed all the stop words from bug summary.

Stemming to base stem: the process of converting derived words to their base word (stem) is known as stemming. Standard Porter stemming algorithm can be utilized for stemming [20].

Feature Reduction: tokens of minimum 3 and maximum 40 occurrences have been considered because most of the data mining algorithm may not be able to handle large feature sets.

Weight by Information Gain or InfoGain: it is helpful in determining the importance or relevance of the term. It helps in selection of top few terms in the data set.

We have made a workflow in RapidMiner to extract a set of terms from bug summary attribute. We have taken tokenize mode as non-letters and in filter tokens parameter we have set min chars value as 3 and max chars value as 50. We used English dictionary to filter the stop words.

III. MODEL BUILDING

Our study consists of following steps:

1. **Data Extraction**
   a. From CVS repository: https://bugzilla.mozilla.org/, downloaded bug reports for 3 products of Mozilla open source project.
   b. Store the downloaded bug reports in excel file for further processing.
2. **Data Pre-processing**
   a. In RapidMiner developed a workflow to extract individual terms of bug summary.
3. **Data Preparation**
   a. For different severity and preiority levels, we have taken numeric values from 1 to 7 and from 8 to 12.
   b. Assigned a numeric value from 13 to 43 to top 30 terms based on InfoGain.
   c. For each assignee take a unique numeric value.
   d. Filtered bug-fix time for 0 to 99 days as maximum number of bugs has fix time in this range only. Define three bug-fix time ranges: 0 to19 days, 20 to 64 days and 65 to 99 days. Assign a numeric value from 1 to 3 to these three ranges.
4. **Association Rule Mining and Clustering**
   a. ARMADA (Association Rule Miner And Deduction Analysis) is a Data Mining tool of MATLAB software that extracts Association Rules from numerical data files using a variety of selectable techniques and criteria [21]. We have applied Apriori algorithm by using ARMADA tool. As a result we get association rules for bug-fix time prediction with severity, priority, summary terms and assignee as antecedents.
   b. We have applied k-means clustering algorithm in SPSS(Statistical Package for Social Sciences) software followed by Apriori algorithm for each resulting cluster by using MATLAB software

with minimum confidence 20% and minimum support 7%.

5. Testing and Validation

Assess the resulting association rules in terms of different performance measures namely support and confidence.

IV. Results and Discussion

In this paper, we have proposed two approaches to apply association mining. In first approach, we have mined the association rules for bug-fix time prediction with bug severity, priority, summary terms and assignee as antecedents by applying Apriori algorithm of ARMADA tool in MATLAB software. We have considered association rules with minimum confidence 20% and minimum support 7% for AddOnSDK and Bugzilla products. In thunderbird product we have very less number of bug reports as a result of which we get association rules with minimum confidence 20% and support 3%. All the 3 datasets have more than 100 rules. For this reason, we do not list them all, but instead we present top 5 rules based on the highest confidence. In table II we have presented top five association rules of AddOnSDK product for three defined ranges.

TABLE II. Top five Association rules for AddOnSDK

Association Rules (minimum support=7%, minimum confidence=20%)
Bug-fix time 0-19 days
1. Priority {P1} ∧ Assignee { Alexandre Poirot} ∧ Term {con} ∧ Term {test} ∧ Term {content} ∧ Term{fail} ⇒ Bug-fix time {0-19 days} @ (10%, 100%)
2. Severity {Major} ∧ Priority {P1} ∧ Term {con} ∧ Term {test} ∧ Term {content} ∧ Term {fail} ⇒ Bug-fix time {0-19 days} @ (8%, 100%)
3. Severity {Major} ∧ Priority {P1} ∧ Assignee {Alexandre Poirot} ∧ Term {con} ∧ Term{content} ∧ Term{fail} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
4. Priority{P1} ∧ Assignee { Alexandre Poirot } ∧ Term {con} ∧ Term{content} ∧ Term{fail} ⇒ Bug-fix time {0-19 days} @ (11%, 100%)
5. Severity{Major} ∧ Priority {P1} ∧ Term{con} ∧ Term {content} ∧ Term {fail} ⇒ Bug-fix time {0-19 days} @ (9%, 100%)
Bug-fix time 20-64 days
1. Severity {Major} ∧ Priority {P1} ∧ Term {win} ∧ Term {window} ∧ Term {updat} ∧ Term {privat} ⇒ Bug-fix time {20-64 days} @ (7%, 100%)
2. Severity {Major} ∧ Assignee {Will Bamberg} ∧ Term {doc} ∧ Term {document} ∧ Term {page} ⇒ Bug-fix time {20-64 days} @ (7%, 100%)
3. Severity {Major} ∧ Priority{P1} ∧ Term {mod} ∧ Term {modul} ∧ Term {privat} ⇒ Bug-fix time {20-64 days} @ (7%, 100%)
4. Severity {Major} ∧ Term {mod} ∧ Term {modul} ∧ Term {privat} ⇒ Bug-fix time {20-64 days} @ (8%, 100%)
5. Severity {Major} ∧ Term {modul} ∧ Term {privat} ⇒ Bug-fix time {20-64 days} @ (8%, 100%)
Bug-fix time 65-99 days
1. Severity {Major} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (9%, 31%)
2. Term {text} ⇒ Bug-fix time {65-99 days} @ (9%, 29%)
3. Severity {Major} ∧ Term {con} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (7%, 27%)
4. Term {con} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (7%, 25%)
5. Priority {P1} ∧ Term {tab} ⇒ Bug-fix time {65-99 days} @ (8%, 23%)

The first association rule is a six antecedent rule, which reveals that a bug with priority *P1*, assignee *Alexander Poirot* and summary containing terms *con, test, content* and *fail* can have a fix time of *0 to 19 days* with a significance of 10 percent and a certainty of 100 percent. Second association rule means that a bug with severity *Major*, priority *P1*, and summary containing terms *con, test, content* and *fail* can have a fix time of *0 to 19 days* with a significance of 8 percent and a certainty of 100 percent. Third rule shows that a bug with severity *Major*, priority *P1* and summary containing terms *con, content* and *fail* can have a fix time of *0 to 19 days* with a significance of 7 percent and a certainty of 100 percent. Rule four reveals that 11 percent of the bugs in the bug data set have priority *P1*, assignee *Alexandre Poirot,* summary containing terms *con, content, fail* and bug-fix time of *0 to 19 days.* 100 percent of the bugs in the bug data set that have *priority P1*, assignee *Alexandre Poirot,* summary containing terms *con, content, fail* also have bug-fix time of *0-19 days.* The fifth rule shows that the bug having severity *Major*, priority *P1* and summary containing terms *con, content* and *fail* can have bug-fix time of *0 to 19 days* with a significance of 9 percent and a certainty of 100 percent. Similarly we have interpreted association rules of other bug-fix time ranges.

We have shown top five association rules to predict bug-fix time for Thunderbird product in table III.

TABLE III. Top five Association rules for Thunderbird

Association Rules (minimum support=3%, minimum confidence=20%)
Bug-fix time 0-19 days
1. Severity {Major} ∧ Term {add} ∧ Term {icon} ∧ Term {address} ⇒ Bug-fix time {0-19 days} @ (3%, 100%)
2. Severity {Major} ∧ Priority {P3} ∧ Term {text} ∧ Term {box} ⇒ Bug-fix time {0-19 days} @ (3%, 100%)
3. Severity {Major} ∧ Priority {P3} ∧ Term {window} ∧ Assignee {Andreas Nilsson} ⇒ Bug-fix time {0-19 days} @ (3%, 100%)
4. Term {tool} ∧ Term {toolbar} ∧ Assignee {Blake Winton} ⇒ Bug-fix time {0-19 days} @ (3%, 100%)
5. Term {config} ∧ Term {auto} ∧ Assignee {Blake Winton} ⇒ Bug-fix time {0-19 days} @ (3%, 100%)
Bug-fix time 20-64 days
1. Severity {Major} ∧ Assignee {David} ∧ Term {move} ∧ Term {remov} ⇒ Bug-fix time {20-64 days} @ (3%, 100%)
2. Term {add} ∧ Term {pre} ⇒ Bug-fix time {20-64 days} @ (3%, 100%)
3. Term {mail} ∧ Term {move} ∧ Term {remov} ⇒ Bug-fix time {20-64 days} @ (3%, 75%)
4. Assignee {David} ∧ Term {move} ⇒ Bug-fix time {20-64 days} @ (3%, 75%)

5.	Assignee {David} ∧ Term {messag} ⇒ Bug-fix time {20-64 days} @ (3%, 75%)
Bug-fix time 65-99 days	
1.	Priority {P1} ∧ Term {mail} ⇒ Bug-fix time {65-99 days} @ (5%, 63%)
2.	Severity {Major} ∧ Term {thunderbird} ∧ Assignee {Mark Banner} ⇒ Bug-fix time {65-99 days} @ (3%, 60%)
3.	Severity {Major} ∧ Assignee {Mark Banner} ⇒ Bug-fix time {65-99 days} @ (4%, 50%)
4.	Assignee {Mark Banner} ⇒ Bug-fix time {65-99 days} @ (5%, 38%)
5.	Severity {Major} ∧ Term {mail} ⇒ Bug-fix time {65-99 days} @ (3%, 38%)

The first association rule is a four antecedent rule, which reveals that a bug with severity *Major*, and summary containing terms *add, icon* and *address* can have a fix time of *0 to 19 days* with a significance of 3 percent and a certainty of 100 percent. Second association rule means that a bug with severity *Major*, priority *P3*, and summary containing terms *text* and *box* can have a fix time of *0 to 19 days* with a significance of 3 percent and a certainty of 100 percent. Third rule shows that a bug with severity *Major*, priority *P3* and summary containing terms *window and* assignee *Andreas Nilssson* can have a fix time of *0 to 19 days* with a significance of 3 percent and a certainty of 100 percent. Rule four reveals that 3 percent of the bugs in the bug data set have summary containing terms *tool, toolbar,* assignee *Blake Winton* and bug-fix time of *0 to 19 days.* 100 percent of the bugs in the bug data set that have summary containing terms *tool, toolbar* and assignee *Blake Winton* also have bug-fix time of *0-19 days.* The fifth rule shows that the bug with summary containing terms *config, auto* and assignee *Blake Winton* can have bug-fix time of *0 to 19 days* with a significance of 3 percent and a certainty of 100 percent. Similarly we have interpreted association rules of other bug-fix time ranges.

We have shown top five association rules to predict bug-fix time for Bugzilla product in table IV.

TABLE IV. TOP FIVE ASSOCIATION RULES FOR BUGZILLA

Association Rules (minimum support=7%, minimum confidence=20%)	
Bug-fix time 0-19 days	
1.	Severity {Major} ∧ Priority {P1} ∧ Term {check} ∧ Term {set} ∧ Term { setup } ∧ Term { checksetup} ⇒ Bug-fix time {0-19 days} @ (11%, 100%)
2.	Priority {P1} ∧ Term {ing} ∧ Term {check} ∧ Term {set} ∧ Term {setup} ∧ Term {checksetup} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
3.	Assignee {Daniel Buchner} ∧ Term{bug} ∧ Term{hang} ∧ Term{chang} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
4.	Priority{P3} ∧ Term{bug} ∧ Term{ing} ∧ Term{bugzilla} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
5.	Priority{P3} ∧ Assignee {Daniel Buchner} ∧ Term{hang} ∧ Term {chang} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
Bug-fix time 20-64 days	
1.	Priority {P3} ∧ Term {cgi} ∧ Term {edit} ⇒ Bug-fix time {20-64 days} (8%, 100%)
2.	Priority {P3} ∧ Term {edit} ⇒ Bug-fix time {20-64 days} @ (10%, 67%)
3.	Severity {Major} ∧ Term {temp} ∧ Term {templat} ⇒ Bug-fix time {20-64 days} @ (8%, 62%)
4.	Priority {P3} ∧ Term {user} ⇒ Bug-fix time {20-64 days} @ (8%, 57%)
5.	Severity {Major} ∧ Term {temp} ⇒ Bug-fix time {20-64 days} @ (8%, 57%)
Bug-fix time 65-99 days	
1.	Assignee {Gervase Markham} ∧ Term{temp} ∧ Term{templat} ⇒ Bug-fix time {65-99 days} @ (7%, 39%)
2.	Assignee {Gervase Markham} ∧ Term{cgi} ⇒ Bug-fix time {65-99 days} @ (7%, 39%)
3.	Assignee {Matthew Barnson} ⇒ Bug-fix time {65-99 days} @ (10%, 38%)
4.	Assignee {Max Kanat-Alexander} ∧ Term{ing} ⇒ Bug-fix time {65-99 days} @ (9%, 31%)
5.	Assignee {Dawn Endico} ⇒ Bug-fix time {65-99 days} @ (7%, 30%)

The first association rule is a six antecedent rule, which reveals that a bug with severity *Major*, priority *P1*and summary containing terms *check, set, setup* and *checksetup* can have a fix time of *0 to 19 days* with a significance of 11 percent and a certainty of 100 percent. Second association rule means that a bug with priority *P1*, and summary containing terms *check, set, setup* and *checksetup* can have a fix time of *0 to 19 days* with a significance of 7 percent and a certainty of 100 percent. Third rule shows that a bug with assignee *Daniel Buchner* and summary containing terms *bug, hang* and *chang* can have a fix time of *0 to 19 days* with a significance of 7 percent and a certainty of 100 percent. Rule four reveals that 7 percent of the bugs in the bug data set have priority *P3*, summary containing terms *bug, ing, bugzilla* and bug-fix time of *0 to 19 days.* 100 percent of the bugs in the bug data set that have priority *P3* and summary containing terms *bug, ing and Bugzilla* also have bug-fix time of *0-19 days.* The fifth rule shows that a bug with priority *P3*, assignee *Daniel Buchner* and summary containing terms *hang* and *chang* can have bug-fix time of *0 to 19 days* with a significance of 7 percent and a certainty of 100 percent. Similarly we have interpreted association rules of other bug-fix time ranges.

In order to analyze the rule length (number of antecedents) of association rules, we draw the distribution of association rules across all the datasets (Fig. 1 to 3).

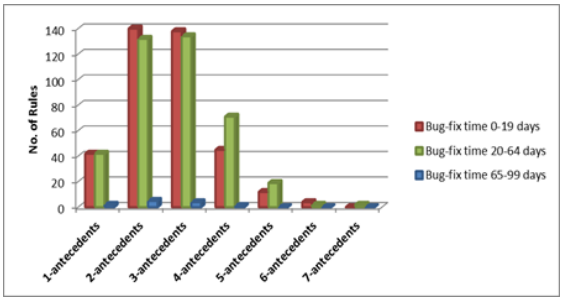

Fig. 1. AddOnSdk association rules (min.supp=7% and min.conf=20%)with different rule length

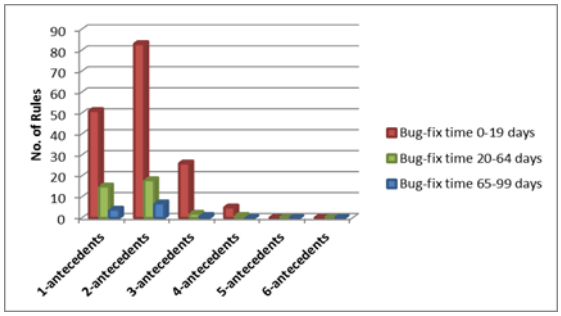

Fig. 2. Thunderbird association rules (min.supp=3% and min.conf=20%)with different rule length

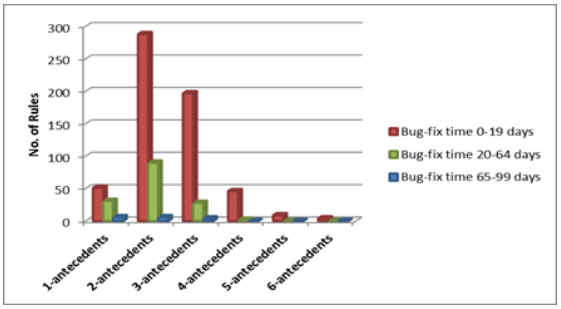

Fig. 3. Bugzilla association rules (min.supp=7% and min.conf=20%)with different rule length

Figure 1 to 3 show that we have maximum association rules with two antecedents (length 2) across all the datasets.

We observe that in all products, we have some rules with same antecedents and consequent except assignee. These rules reveal that for different assignee we have same bug-fix time for same values of other attributes. In this case we will prefer an assignee with higher confidence value to whom we can assign the bug as he is more potential and experienced in fixing such type of bugs. In this way the proposed approach will help in bug triaging which will help in software quality improvement.

We have observed following rules from AddOnSDK product.

1. Severity {Major} ∧ Term {test} ∧ Assignee {Alexandre Poirot}
   ⇒ Bug-fix time {0-19 days} @ (16%, 89%)
2. Severity {Major} ∧ Term {test} ∧ Assignee {Dave Townsend}
   ⇒ Bug-fix time {0-19 days} @ (12%, 71%)
3. Severity {Major} ∧ Term {test} ∧ Assignee {Erik Vold}
   ⇒ Bug-fix time {0-19 days} @ (8%, 50%)
4. Severity {Major} ∧ Priority {P1} ∧ Term {con} ∧ Assignee {Will Bamberg}
   ⇒ Bug-fix time {20-64 days} @ (11%, 65%)
5. Severity {Major} ∧ Priority {P1} ∧ Term {con} ∧ Assignee {Alexandre Poirot}
   ⇒ Bug-fix time {20-64 days} @ (9%, 35%)

First three rules reveals that bugs with severity *Major* and summary containing term *test* have three choices of assignee i.e. *Alexandre Poirot* or *Dave Townsend* or *Erik Vold* to get fixed in 0 to 19 days with certainty of 89, 71 and 50 percent respectively. We observe that the bug should be assigned to *Alexandre Poirot* as the rule with this assignee gives highest certainty. Similarly we can infer from last two rules that we should assign the bug to *Will Bamberg* as the rule with this assignee gives higher certainty. Similar inference we can draw for other two datasets also.

We observe that in all products we have some rules with same antecedents except assignee. These rules reveal that different assignee will fix same bugs with same attributes with different bug-fix time. In this case, we will prefer an assignee with lower fix time in fixing such type of bugs. In this way the proposed approach will help in choosing assignee which can fix the bug in shortest time.

We have observed following rules from Bugzilla product.

1. Severity {Major} ∧ Assignee {Terry Weissman}
   ⇒ Bug-fix time {0-19 days} @ (67%, 80%)
2. Severity {Major} ∧ Assignee {Bradley Baetz}
   ⇒ Bug-fix time {20-64 days} @ (7%, 44%)
3. Severity {Major} ∧ Assignee {Max Kanat-Alexander}
   ⇒ Bug-fix time {65-99 days} @ (8%, 22%)
4. Priority{P1} ∧ Assignee {Dave Miller}
   ⇒ Bug-fix time {0-19 days} @ (7%, 78%)
5. Priority{P1} ∧ Assignee {Max Kanat-Alexander}
   ⇒ Bug-fix time {20-64 days} @ (11%, 42%)

First three rules reveals that bugs with severity *Major* can be assigned to three different assignee: *Terry Weissman*, *Bradley Baetz* and *Max Kanat-Alexander*. All the three assignee will fix the same bug with severity Major with different fix time ranges. We will preferably assign the bug to an assignee who will fix it in minimum time and i.e. *Terry Weissman*. Similarly we can infer from last two rules that we should assign the bug to *Dave Miller* as he will solve the bug earliest. Similar inference we can draw for other two datasets also.

In second approach, we have presented clustering based association rule mining for bug-fix time prediction. We have partitioned the AddOnSDK dataset into 5 clusters using k-means clustering method. In cluster 1, there is only one data. Cluster 2 contains 93 data, cluster 3 contains 379 data, cluster 4 contains 115 data and cluster 5 contains 28 data. After portioning, we have applied Apriori algorithm on each cluster with minimum confidence 20% and minimum support 2%.

Table V presents top five association rules from five clusters formed by k-means clustering for AddOnSDK product.

TABLE V. TOP FIVE ASSOCIATION RULES FOR ADDONSDK

Association Rules (minimum support=2%, minimum confidence=20%)
Bug-fix time 0-19 days
Cluster 2
1. Term {con} ∧ Term {test} ∧ Term{fail} ⇒ Bug-fix time {0-19 days} @ (5%, 100%)
2. Priority {P1} ∧ Term {con} ∧ Term {test} ⇒ Bug-fix time {0-19 days} @ (5%, 100%)

3.	Assignee {Alexandre Poirot} ∧ Term {test} ∧ Term{fail} ⇒ Bug-fix time {0-19 days} @ (5%, 100%)
4.	Priority{P1} ∧ Assignee { Alexandre Poirot } ∧ Term {test} ⇒ Bug-fix time {0-19 days} @ (5%, 100%)
5.	Priority {P1} ∧ Term{con} ∧ Term {test} ∧ Term {fail} ⇒ Bug-fix time {0-19 days} @ (5%, 100%)
Cluster 3	
1.	Priority {P1} ∧ Term {fire} ∧ Term {test} ∧ Term{firefox} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
2.	Priority {P1} ∧ Assignee {Alexandre Poirot} ∧ Term {fail} ∧ Term {test} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
3.	Severity {Major} ∧ Priority{P1} ∧ Term {test} ∧ Term{firefox} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
4.	Severity {Major} ∧ Priority{P1} ∧ Term {test} ∧ Term {fire} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
5.	Severity {Major} ∧ Priority{P1} ∧ Term {test} ∧ Term {fire} ∧ Term{firefox} ⇒ Bug-fix time {0-19 days} @ (7%, 100%)
Cluster 4	
1.	Severity {Major} ∧ Priority{P2} ∧ Term {cfx} ⇒ Bug-fix time {0-19 days} @ (2%, 100%)
2.	Severity {Major} ∧ Priority{P1} ∧ Term {get} ⇒ Bug-fix time {0-19 days} @ (2%, 100%)
3.	Severity {Major} ∧ Priority{P2} ∧ Term {get} ⇒ Bug-fix time {0-19 days} @ (2%, 100%)
4.	Severity {Major} ∧ Priority{P2} ∧ Assignee {Alexandre Poirot} ∧ Term {get} ⇒ Bug-fix time {0-19 days} @ (2%, 100%)
5.	Severity {Major} ∧ Priority{P3} ∧ Term {fail} ⇒ Bug-fix time {0-19 days} @ (2%, 100%)
Cluster 5	
1.	Severity {Major} ∧ Assignee {Alexandre Poirot} ∧ Term {con} ∧ Term {content} ⇒ Bug-fix time {0-19 days} @ (5%, 83%)
2.	Severity {Major} ∧ Term {con} ∧ Term {content} ⇒ Bug-fix time {0-19 days} @ (5%, 71%)
3.	Severity {Major} ∧ Priority{P1} ∧ Term {fail} ⇒ Bug-fix time {0-19 days} @ (6%, 67%)
4.	Priority{P1} ∧ Term {fail} ∧ Term {win} ∧ Term {window} ⇒ Bug-fix time {0-19 days} @ (5%, 63%)
5.	Severity {Major} ∧ Priority{P1} ∧ Term {fail} ∧ Term {test} ⇒ Bug-fix time {0-19 days} @ (5%, 63%)
Bug-fix time 20-64 days	
Cluster 2	
1.	Severity {Major} ∧ Priority {P4} ∧ Assignee {Will Bamberg} ∧ Term {con} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
2.	Severity {Major} ∧ Priority {P3} ∧ Assignee {Will Bamberg} ∧ Term {updat} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
3.	Severity {Major} ∧ Priority {P1} ∧ Assignee {Will Bamberg} ∧ Term {document} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (6%, 100%)
4.	Severity {Major} ∧ Assignee {Will Bamberg} ∧ Term {con} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
5.	Priority {P3} ∧ Assignee {Will Bamberg} ∧ Term {con} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
Cluster 3	
1.	Severity {Major} ∧ Priority {P1} ∧ Assignee {Will Bamberg} ∧ Term {doc} ∧ Term {document} ⇒ Bug-fix time {20-64 days} @ (8%, 62%)
2.	Severity {Major} ∧ Priority{P1} ∧ Term {page} ⇒ Bug-fix time {20-64 days} @ (9%, 60%)
3.	Severity {Major} ∧ Priority{P1} ∧ Term {tab} ⇒ Bug-fix time {20-64 days} @ (10%, 59%)
4.	Severity {Major} ∧ Priority {P2} Term {mod} ⇒ Bug-fix time {20-64 days} @ (7%, 54%)
5.	Assignee {Will Bamberg} ∧ Term {document} ⇒ Bug-fix time {20-64 days} @ (16%, 53%)
Cluster 4	
1.	Severity {Major} ∧ Priority{P1} ∧ Assignee {Will Bamberg} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (2%, 100%)
2.	Severity {Major} ∧ Assignee {Will Bamberg} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (3%, 100%)
3.	Severity {Major} ∧ Priority {P1} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (3%, 100%)
4.	Priority {P1} ∧ Assignee {Will Bamberg} ∧ Term {doc} ⇒ Bug-fix time {20-64 days} @ (2%, 100%)
5.	Severity {Major} ∧ Assignee {Will Bamberg} ∧ Term {updat} ⇒ Bug-fix time {20-64 days} @ (2%, 100%)
Cluster 5	
1.	Severity {Major} ∧ Term {win} ∧ Term {window} ∧ Term {updat} ∧ Term {private} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
2.	Severity {Major} ∧ Priority{P1} ∧ Term {window} ∧ Term {updat} ∧ Term {private} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
3.	Severity {Major} ∧ Priority{P1} ∧ Term {win} ∧ Term {updat} ∧ Term {private} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
4.	Severity {Major} ∧ Priority{P1} ∧ Term {mod} ∧ Term {modul} ∧ Term {private} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
5.	Severity {Major} ∧ Priority{P1} ∧ Term {win} ∧ Term {window} ∧ Term {updat} ∧ Term {private} ⇒ Bug-fix time {20-64 days} @ (5%, 100%)
Bug-fix time 65-99 days	
Cluster 2	
1.	Severity {Major} ∧ Term {tab } ⇒ Bug-fix time {65-99 days} @ (6%, 35%)
2.	Term {tab} ⇒ Bug-fix time {65-99 days} @ (6%, 33%)
3.	Severity {Major} ∧ Term {window} ⇒ Bug-fix time {65-99 days} @ (5%, 25%)
4.	Severity {Major} ∧ Term {win} ∧ Term {window} ⇒ Bug-fix time {65-99 days} @ (5%, 25%)
5.	Term {window} ⇒ Bug-fix time {65-99 days} @ (5%, 24%)
Cluster 3	
1.	Priority{P1} ∧ Term {modul} ⇒ Bug-fix time {65-99 days} @ (7%, 25%)
2.	Severity {Major} ∧ Priority{P1} ∧ Term {modul} ⇒ Bug-fix time {65-99 days} @ (7%, 27%)
3.	Priority{P1} ∧ Term {mod} ∧ Term {modul} ⇒ Bug-fix time {65-99 days} @ (7%, 25%)
4.	Severity {Major} ∧ Priority{P1} ∧ Term {mod} ⇒ Bug-fix time {65-99 days} @ (7%, 21%)
5.	Severity {Major} ∧ Priority{P1} ∧ Term {mod} ∧ Term {modul} ⇒ Bug-fix time {65-99 days} @ (7%, 27%)
Cluster 4	
1.	Severity {Enhancement} ∧ Priority{P3} ⇒ Bug-fix time {65-99 days} @ (2%, 67%)
2.	Severity {Major} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (2%, 67%)
3.	Severity {Major} ∧ Term {sdk} ⇒ Bug-fix time {65-99 days} @ (2%, 40%)
4.	Severity {Major} ∧ Priority{P1} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (2%, 67%)
5.	Priority{P1} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (2%, 67%)
Cluster 5	
1.	Severity {Major} ∧ Priority{P1} ∧ Assignee { Dave Townsend } ∧ Term {con} ∧ Term {add} ∧ Term {text}

	⇒ Bug-fix time {65-99 days} @ (2%, 100%)
2.	Severity {Major} ∧ Priority{P1} ∧ Assignee { Dave Townsend } ∧ Term {con} ∧ Term {test} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (2%, 100%)
3.	Severity {Major} ∧ Priority{P1} ∧ Assignee { Dave Townsend } ∧ Term {con} ∧ Term {test} ∧ Term {add} ⇒ Bug-fix time {65-99 days} @ (2%, 100%)
4.	Priority{P1} ∧ Term {test} ∧ Term {add} ∧ Term {fail} ∧ Term {error} ∧ Term {addon} ⇒ Bug-fix time {65-99 days} @ (2%, 67%)
5.	Severity {Major} ∧ Priority{P1} ∧ Assignee { Dave Townsend } ∧ Term {con} ∧ Term {test} ∧ Term {add} ∧ Term {text} ⇒ Bug-fix time {65-99 days} @ (2%, 100%)

We observe that, if we apply association mining after clustering, we get different association rules. As we are partitioning the datasets into clusters, we get association rules with decreased support count i.e. 2%. Results also show that, the confidence count lies in the range of 21 to 100%.

We get the similar results for other datasets.

V. Related work

In last few years, a number of valuable studies have been conducted to address the problem of bug-fix time prediction. A study on 72,482 bug reports from nine versions of Linux software named Ubuntu has been conducted by [3]. Results show that people participating in groups of size ranging from 1 to 8 users fixed 95% bug reports. The study results in 92% linear relationship between the number of people participating in fixing a bug report and bug-fix time. The applied linear regression model resulted in R^2 up to 0.98. At attempt has been made on 512,474 bug reports of five open source projects –Eclipse, Chrome and three products of Mozilla project – Thunderbird, Firefox and Seamonkey to test the prediction performance of existing models by using multivariate and univariate regression [4]. As a result it was found that existing models have predictive power between 30% and 49% and more independent attributes can be included. No correlation was found between bug-fix likelihood, bug-opener's reputation and the time it takes to fix a bug. A model has been proposed for six projects: Eclipse JDT, Eclipse Platform, Mozilla Core, Mozilla Firefox, Gnome GStreamer and Gnome Evolution to predict that how promptly a new bug report will receive attention [5]. Results show an improvement in bug-fix time prediction accuracy if number of developers and number of comments are included. An attempt has been made to show the bug-fix time trends in Mozilla and Apache projects [22]. It was found that on average resolution time for bugs of priority levels 4 and 5 exceeds 100 days, bugs of the priority level 2 are resolved in 80 days or less and bugs of the priority level 1 or 3 are resolved in 30 days or less. An attempt has been made to focus on the delays incurred by developers during bug fixing [25]. A study has been conducted to filter out the data sets by identifying the potential outliers in the distribution of the fix-time attribute. Results showed that filtering these outliers can improve the accuracy of the prediction models [26].

An attempt has been made to present an application of association rule mining to predict software defect associations and defect correction effort with SEL defect data [23]. The results show that for the defect association prediction, the minimum accuracy is 95.38 percent, and the false negative rate is just 2.84 percent; and for the defect correction effort prediction, the accuracy is 93.80 percent for defect isolation effort prediction and 94.69 percent for defect correction effort prediction. Recently, a study discussed the application of association mining in bug triaging. Authors have used Apriori algorithm to predict the right developer to work on the bug by taking bug's severity, priority and summary terms as the antecedents [24]

To best of our knowledge, no approach has been proposed till now to mine association rules among different bug attributes to predict bug-fix time. Managers can use association rules to improve development process by doing a bug-fix time prediction for a given set of bug attributes. Several performance studies have resulted in better accuracy for associative classification than state-of-the-art classification methods [9-18]. Our work has been motivated by the successful application of association rule mining in various fields.

VI. Threats to Validity

Factors that can affect the validity of our study are as follow:

Construct Validity: We have not empirically validated the independent attributes taken in our study.

Internal Validity: Except the four attributes namely severity, priority, summary terms and assignee taken in our study, developer's reputation can also be considered as it is an important attribute which can contribute in bug-fix time prediction.

External Validity: We have considered only open source Mozilla products. The study can be extended for other open source and closed source software.

Reliability: RapidMiner, SPSS and MATLAB software have been used in this paper for model building and testing. The increasing use of these software confirms the reliability of the experiments. Errors in performance measures such as accuracy of these tools has not been considered and handled.

VII. Conclusion

The time to fix a bug after the bug was introduced is called bug-fix time. It is an important factor for bug related analysis, such as measuring software quality or coordinating development effort during bug triaging. Prior work has proposed many bug-fix time prediction models based on various bug attributes (number of developers who participated in fixing the bug, bug severity, bug-opener's reputation, number of patches) for predicting the fix time of a newly reported bug. Several studies have been conducted by using classification and regression models. We have proposed an approach for bug-fix time prediction based on other bug attributes namely summary terms, priority, severity and assignee by using Apriori algorithm and *k*-means clustering followed by Apriori algorithm. We have also used k-means clustering method to get groups of correlated variables

followed by association rules mining inside each cluster. We have validated our results on 1,695 bug reports of AddOnSDK, Thunderbird and Bugzilla products of Mozilla open source project. We have presented top five association rules for 20% minimum confidence and 3% and 7% minimum support. We observe that, if we apply association mining after clustering, we get different association rules. As we are partitioning the datasets into clusters, we get association rules with decreased support count i.e. 2%. Results show that, the confidence count lies in the range of 21 to 100%.

By using these rules we can predict the bug-fix time for a newly coming bug. We also observe that our approach for bug-fix time prediction will be helpful in bug triaging by assigning a bug to the most potential and experienced assignee that will solve the bug in minimum time period. Prediction of bug-fix time will help the managers in measuring software quality and in software development process. From results, we can observe the number of association rules having high confidence and support with higher severity and priority as antecedents and short bug-fix time as consequent. A large number for such rules show that more important bugs are fixed with out any delay. This information is useful in determining software quality during software evolution process. Further, for bugs with long predicted fix time we need to pay more attention to the related source files to make sure that the files remain stable during fixing process. This will again help in determining software quality. We will extend our work with other association mining algorithms to empirically validate the results.

References

[1] S. Kim and J. E. Whitehead, "How long did it take to fix bugs?," Int. Workshop Mining Software Repositories. New York, NY, USA, ACM, pp. 173–174, 2006

[2] P. Hooimeijer and W. Weimer, "Modeling bug report quality," ASE 2007.

[3] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," Int. Conf. Software Management (Edmonton, AB). IEEE, pp. 523-526, September 20-26, 2009, DOI= http://ieeexplore.ieee.org/10.1109/ICSM.2009.5306337.

[4] P. Bhattacharya and I. Neamtiu, "Bug-fix Time Prediction Models: Can We Do Better?," 8th Working Conf. Mining Software Repositories (New York, NY, USA). ACM, pp. 207-210, 2012, DOI= http://dl.acm.org/10.1145/1985441.1985472.

[5] E. Giger, M. Pinzger and H. Gall, "Predicting the fix time of bugs," Int. Workshop Recommendation Systems on Software Enginnering (New York, NY, USA), ACM, pp. 52-56, 2010.

[6] M. Sharma, M. Kumari and V.B. Singh, "Understanding the Meaning of Bug Attributes and Prediction Models," 5th IBM Collaborative Academia Research Exchange Workshop, I-CARE, Article No. 15, ACM, 2013.

[7] R. Agrawal, T. Imielinski and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," SIGMOD Conf. Management of Data, ACM, May 1993.

[8] Q. Song, M. Shepperd, M. Cartwright and C. Mair, "Software defect association mining and defect correction effort prediction," IEEE Transactions on Software Engineering, Vol. 32(2) pp. 69 – 82, 2006.

[9] K. Ali, S. Manganaris and R. Srikant, "Partial Classification Using Association Rules," Int. Conf. Knowledge Discovery and Data Mining., pp. 115-118, 1997

[10] G. Dong, X. Zhang, L. Wong, and J. Li, "CAEP: Classification by Aggregating Emerging Patterns," Int. Conf. Discovery Science, pp. 30-42, 1999.

[11] B. Liu, W. Hsu, and Y. Ma, "Integrating Classification and Association Rule Mining," Int. Conf. Knowledge Discovery and Data Mining, pp. 80-86, 1998.

[12] R. She, F. Chen, K. Wang, M. Ester, J.L. Gardy and F.L. Brinkman, "Frequent-Subsequence-Based Prediction of Outer Membrane Proteins," ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2003.

[13] K. Wang, S.Q. Zhou and S.C. Liew, "Building Hierarchical Classifiers Using Class Proximity," Int. Conf. Very Large Data Bases, pp. 363-374, 1999.

[14] K. Wang, S. Zhou and Y. He, "Growing Decision Tree on Support-Less Association Rules." Int. Conf. Knowledge Discovery and Data Mining, 2000.

[15] Q. Yang, H.H. Zhang and T. Li, "Mining Web Logs for Prediction Models in WWW Caching and Prefetching," ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining, 2001.

[16] X. Yin and J. Han, "CPAR: Classification Based on Predictive Association Rules," SIAM Int. Conf. Data Mining, 2003.

[17] A.T.T. Ying, C.G. Murphy, R. Ng and M.C. Chu-Carroll,. "Predicting Source Code Changes by Mining Revision History," Int. Workshop Mining Software Repositories, 2004.

[18] T. Zimmermann, P. Weigerber, S. Diehl and A. Zeller, "Mining Version Histories to Guide Software Changes," Int. Conf. Software Engineering, 2004.

[19] I. Mierswa, M. Wurst, R. Klinkenberg, M. Scholz and T. Euler, "YALE: Rapid Prototyping for Complex Data Mining Tasks," ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD-06), 2006 (http://www.rapid-i.com).

[20] M. Porter, "An algorithm for suffix stripping," Program.Vol. 14 (3), pp. 130–137, 2008.

[21] "http://in.mathworks.com/.../3016-armada-data-mining-tool-version-1-4", 2015, URL: http://in.mathworks.com/[accessed:2015-07-24].

[22] A. Mockus, R. T. Fielding and J. D. Herbsleb, "Two case studies of open source software development: Apache and Mozilla," ACM Trans. on Software Eng. Vol. (11)3, 2002.

[23] M. Plassea, N. Nianga, G. Saportaa, A. Villeminotb and L. Leblondb, "Combined use of association rules mining and clustering methods to find relevant links between binary rare attributes in a large data set," Computational Statistics & Data Analysis, ELSEVIER, 2007.

[24] M. Sharma, M. Kumari and V.B. Singh, "Bug Assignee Prediction Using Association Rule Mining," ICCSA 2015, Part IV, LNCS 9158, pp.444–457, 2015.

[25] F. Zhang , F. Khomh , Y. Zou and A. E. Hassan, "An Empirical Study on Factors Impacting Bug Fixing Time," 19th Working Conference on Reverse Engineering (WCRE), pp. 225-234, 15-18 Oct 2012.

[26] W. AbdelMoez, M. Kholief and F. M. Elsalmy, "Improving bug fix-time prediction model by filtering out outliers," International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE), 2013 , pp.359-364, 9-11 May 2013.