International Journal of Engineering Research
Volume No.5, Issue No.7, pp : 630-634

ISSN:2319-6890)(online),2347-5013(print)
1 July 2016

Efficient Method for Edit Recommendation using Hybrid History Mining and Relevance Feedback

Miss. Shradha P. Patil[1] , Prof. B. Padmavathi[2]
G. H. Raisoni College of Engineering and Management, Wagholi, Pune.
shradhapatil7@gmail.com,b.padmavathi@raisoni.net

Abstract: Now a day's to speed up the software development process, use of recommendation systems plays important role while developing software by software developers. This paper presents EMI (Extended MI) technique which generates history-based recommendation systems following two approaches, one is by mining view histories and another by mining edit histories. Moreover it uses relevance feedback method to improve the accuracy of recommendation.

Keywords— Association rules, Context formation, Data mining, Mining programmer interaction histories, ROSE

I. INTRODUCTION

Programmer's significant amount of time is wasting in investigating files to edit. To increase the productivity and performance of software development the researchers have developed recommendation system. The use of recommendation system boosts up the productivity of developer by recommending files faster to edit. Association rules mining is done for such in software revision histories while, using only edit histories by mining coarse-grained rules produces recommendations with low accuracy, and generates recommendations only after the developer edits a file. The recommendation presented is falls into two groups. In first group T. Zimmermann et al.[5] proposed recommending based on software revision history. Another group the recommendation is based on the mining programmer interaction histories. But the existing system produces recommendation with less accuracy and takes more time for generating recommendations. These two have developed separately, leaving largely unanswered the question that which history is better to mine whether view history or edit history. By taking those problems in consideration recently MI technique is presented which produce recommendations by using both view and edit histories. This method generates better accuracy, flexibility and speed up recommendation but there is no end user satisfaction is considered with this method. So we have developed relevance feedback method to accomplish end user satisfaction and based on that the proposed system generates more accurate recommendation with less time. Using detailed edit, view histories and relevance feedback method to recommend files to edit produces the following leverages:

A. Accurate recommendations: The Use of relevance feedback algorithm and MI approach provides together generates more accurate recommendation.

B. Early recommendations: The programmer can get early recommendation even before editing a single file with EMI approach.

C. Flexible recommendations: When recommendations are based on viewed files the recommendations change in response to developer's analysis paths.

II. RELATED WORK

R. Robbes and M. Lanza, proposed to completely reconstruct development sessions here introduced an evolution monitoring prototype that records every semantic change applied on a system .In this work the fine-grained information is used to characterize and understand the development sessions as they were carried out on two object-oriented systems[1].

M. Kim and D. Notkin, suggested to implement existing differencing approaches, proposed Logical Structural Diff (LSdiff) to concisely denote systematic changes that deviate from these systematic changes . LSdiff abstracts a program as code elements and their structural dependencies. This abstraction is used to recognize systematic changes that are identified by consistent changes to code elements which share common structural characteristics like accessing the same field and implementing same interface. [2].

R. Agrawal, T. Imielinski and A. N. Swami, presented an efficient algorithm that presents all significant association rules between items in the database. A large database of the customer transactions is given. Each transaction consists of items acquired by a customer in a visit. The algorithm produces buffer management, pruning techniques and novel estimation. It also generate results by applying this algorithm to sales data from a large retailing company, which shows the effectiveness of the algorithm. [4].

T. Zimmermann, , S. Diehl ,P. Weissgerber and A. Zeller, says to guide programmers we apply data mining to version histories along with related changes. This uses the set of existing changes that suggest and predict the similar more changes, item linking that is invisible by program analysis and prohibit errors that occurs due to incomplete changes. After first transformation, this ROSE technique can correctly predict 26% of new files to be changed [5].

A. T. T. Ying, R. Ng and M. C. Chu-Carroll, G. C. Murphy-To help a developer we have developed new approach of mining revision history for recognizing appropriate source code for a change task. This applying approach to two open-source systems, Eclipse and Mozilla, which provides useful recommendations and then assessing the results based on the predictions and likely interest to a developer. Additionally to provide evidence for this hypothesis, we have developed a set of interesting criteria for evaluating the utility of recommendations that can be used in qualitative analysis of source code recommendations [6].

D. Kim, Y. Tao, S. Kim and A. Zeller proposed a two-phase prediction model to support developers in locating bugs and debugging that uses content of bug reports to suggest the files likely to be fixed. In the first phase, model checks whether the bug report contains sufficient information for prediction or not. If so, based on the content of the bug report, the model starts to predict files to be fixed. By analysis we observed that 70 percent of these predictions given the correct files[7].

III.PROPOSED WORK

A. EMI(Extended Mining Programmer interaction Histories)

MI is an approach which extends ROSE that mines software revision histories. To mine programmer interaction histories we have revised ROSE approach. The revised ROSE approach forms a context by using only edited files by mining the association rules from edited files of programmer interaction histories.

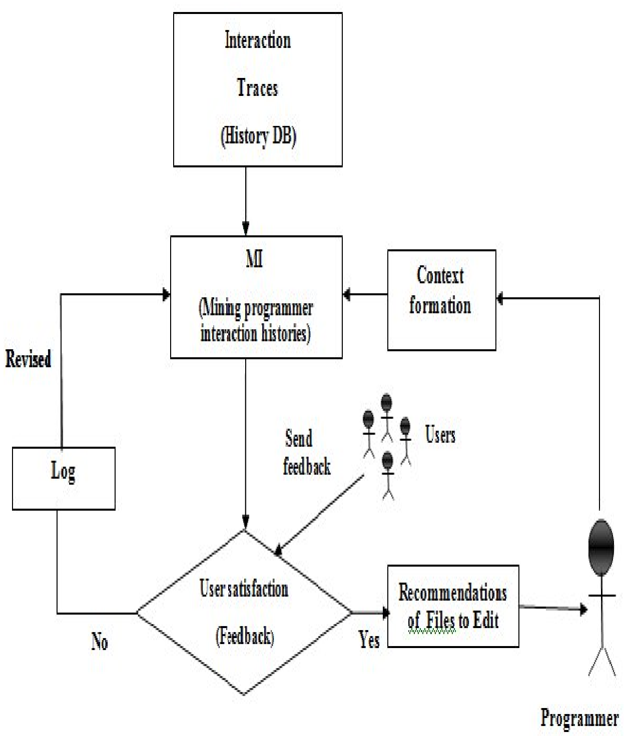


Fig. 1. Architecture of the proposed recommendation system EMI

To edit recommendation of files we have used this version of ROSE to encompass viewed files, recently MI approach is proposed, for mining association rules in programmers interaction histories (MI)[3].

MI forming a hybrid context of viewed files and edited files by mining the association rules from both viewed and edited files. This MI approach can use various methods to form a context from edited and viewed files.

The limitation of MI is that it does not considers end user satisfaction , so there is always scope for improvement in accuracy. So we have proposed extended MI technique(EMI) which considers end user satisfaction by using relevance feedback method. Relevance feedback approach maintains log of feedbacks generated by the end users. We are applying filtering technique on log record so that it can refine and regenerate more accurate recommendation next time for the same query in very less time.

Recommendation of files to edit by using the records of viewed files, MI mines programmer's interaction histories that are history database. As shown in Fig. 1, using the current context MI mines interaction traces and finds association rules. To gain the accuracy of recommendation we are using relevance feedback method, which maintains log of feedbacks based on end users satisfaction. Feedbacks may contain the irrelevant recommendations generated for the query. If the end users are satisfied with generated recommendation by MI , it is recommended to another programmer else, proposed system refine and regenerates more accurate recommendations next time for same query with less time.

The fundamental part of the recommendation system is the context. For recommendation system, context describes the information about the user, their environment (e.g., viewed files), and work that are present at the time of recommendation. To generate the context we are using context formation algorithm.

B. Interaction Traces of developers

An interaction trace is information consists of the records that define a programmer's actions (i.e. views and edits) and files on which the actions were taken.

An interaction trace *Tk* is transformed into a pair of sets *Tk = (Vk, Ek)*, where *Vk* is the set of viewed files in interaction trace *Tk*, *Vk = {v1, ...,vn}* and *Ek* is the set of edited files in interaction trace *Tk*, Where *Ek = {E1, ...,En}* .

The collected information of interaction traces can be expressed as *History DB = {Tk|1 ≤ k ≤ i-1}*.

C. Context Formation

Conceptually, a context defines information which can be used to describe the situation of a current user. In a recommendation system, a context generates a query, which forms a recommendation. In EMI, a context is formed from current behaviour of a programmer's. When the current programmer is handling a task m, a context is created on the basis of last files viewed and edited by the current programmer at each time-point in Tm. If the current programmer persists viewing and editing files, the context gets change. The context C can be defined as (Vc, Ec), where Vc is the set of the last viewed v files of the current programmer ,Vc = {v1, ..., vm}, and Ec is the set of the last edited e files by programmer Ec = {e1, ..., em} at each timepoint.

D. Algorithms

The proposed system uses two algorithms, one for context formation and other for relevance feedback.

The context formation algorithm is used for generating the context by using viewed and edited files of user interaction traces.

Relevance Feedback algorithm is used for taking the feedbacks from the users and to maintain the logs. To improve the accuracy and produce early recommendation this uses filtering technique on log records.

Algorithm 1 : Context formation:

Input : Dataset

Viewed files V= {v_1,v_2….v_i}

Edited files E= { e_1,e_2 …...e_i}

Conditional Operations:
Step 1 : If (both V_x and E_y exist) then
Step 2 : Start AND(V_x, E_y)
Step 3 : Else if (V_x or E_y exist) then,
Step 4 : Start OR(V_x , E_y)
Step 5 : Else if both Vx and Ey exist then
Step 6 : apply MI-VA // make recommendations after an edit
else
apply MI-VO // make recommendations before an edit
Step 7 : end else

Algorithm 2 : Relevance Feedback Algorithm
Input : Logs of feedback.
Step 1: Assume that all recommendations in datasets.
Step 2: Select the query as recommendation according to MI.
Step 3: Extract the features of recommendations.
Step 4: Onces we have extract features of recommendations after that generate log.
Step 5: Let L is log.
If { (size(L)==0) // If there is no log data
Goto Step 6 }
Else { While (L!=Null) do
{
Analysis of log
}
Returns gain of recommendations extracted.
}
Step 6: Extracted result is presented.
Step 7: Stop and close log files.

IV. MATHEMATICAL MODELLING

A. Interaction Traces:

T_k =An interaction trace;
k = software evolution task that a programmer performed.
V_k = the set of viewed files in T_k;
Ek is the set of edited files in T_k;
$V_k = \{v_1, ..., v_n\}$
$E_k = \{e_1, ..., e_m\}$.
The collections of interaction traces can be expressed as : History DB = $\{T_k | 1 \le k \le i-1\}$

B. Context:

C = context;
V_c= a set of the last v files that a current programmer has viewed,
E_c= a set of the last e files that the programmer has edited
$V_c = \{v1, ..., v_v\}$, and
$E_c = \{e1, ..., e_e\}$ at each timepoint.

C. Precision and recall:

To measure the accuracy of recommendation, we used several commonly used metrics precision (P) and recall(R).
A= Set of the files recommended to edit which are driven by context C in interaction trace T_i.
E= Set of the files that are actually edited in interaction trace Ti except the edited and viewed files of context C.
Where |E| varies with Ti, because a different Ti will have a different number of edited files.
Input : Dataset.
Output: Recommendation Accuracy.
P =Precision
R =Recall

$$P = \frac{|A \cap E|}{|A|}$$

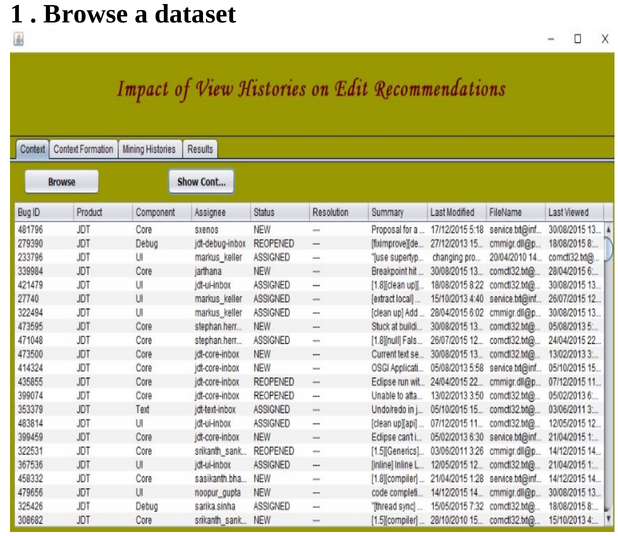
$$R = \frac{|A \cap E|}{|E|}$$

As there are concession between precision and recall it's difficult to compare the accuracy using just values of precision and recall. F-measure is the harmonic mean of precision and recall which allows to measure recommendation accuracy while capturing this concession. F-measure is calculated from the average values of precision and recall as below :

$$F - measure \;\; F = \frac{2 * P * R}{P + R}$$

V. Implementation details:

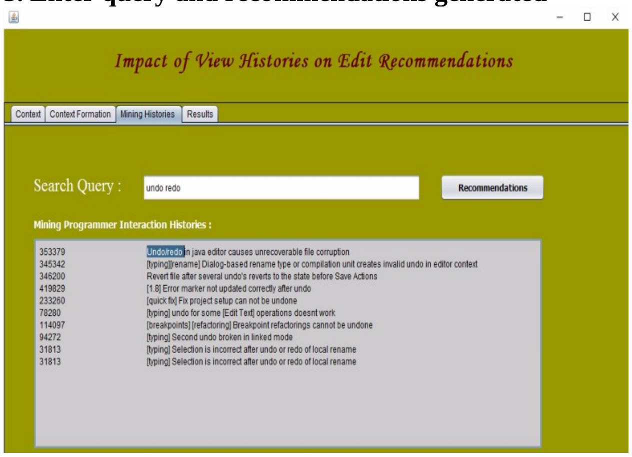
1 . Browse a dataset



ISSN:2319-6890)(online),2347-5013(print)
1 July 2016

2. Context formation

3. Enter query and recommendations generated



VI. EXPERIMENTAL EVALUATION

To evaluate performance of recommendation by Extended MI technique mining viewed and edit histories, it has observed that the performance changes using various *e* and *v* values by setting *v*=*e*=*n* and increasing *n* from 1 to 10.

Fig. 2 presents resulting graph of precision and recall. Curves “n-edits”,“n-views” and “n-views and n-edits” presents results of using edited files (e=n),viewed files(v=n) and both files(v=n and e=n) as a context.

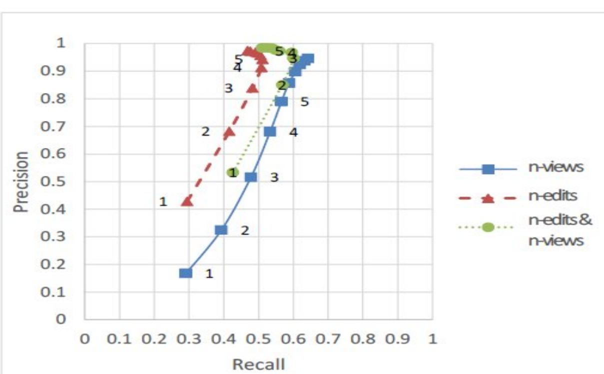


Fig. 2. Comparison of Recommendation Accuracy of MI and EMI

Fig. 2 shows that “n-views and n-edits” achieves higher precision than “n-edits” or “n-views” in y-axis, and also “n-views” shows persistently lower precision than “n-edits”.

It also represents that “n-views and n-edits” consistently gives higher recall than “n-edits” whereas, when n is greater than 5 ,”n-views” gives higher recall than others.

So , as “n-views and n-edits” produces higher precision and recall than “n-edits” we conclude that :

Using viewed and edited files (“n-views and n-edits”) produces consistently higher file-level recommendation accuracy than using only edited files (“n-edits”).

Fig. 3 shows the Comparison of Recommendation Accuracy of EMI and MI. This shows the recommendation accuracy of EMI is consistently higher than the MI across all over the project aspect. To evaluate this result the MI and EMI produces number of quires and recommendations and based on their results the comparison has been made.

Given these results it is clear that using Extended MI technique produces higher recommendation accuracy than using only MI technique for files to edit.

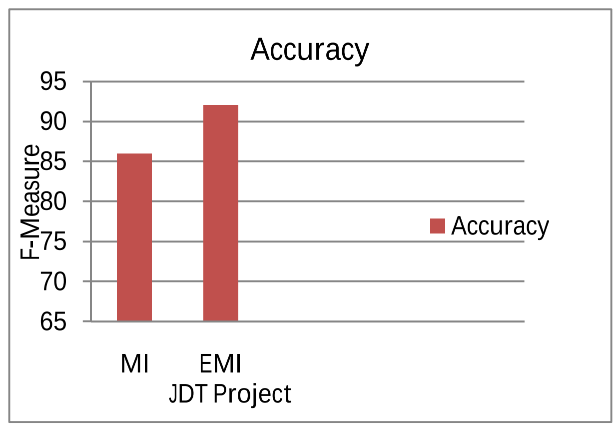


Fig. 3. Comparison of Recommendation Accuracy of EMI and MI

VII. SCOPE OF WORK

Main goal of this system is to present improved method for recommendation systems using explicit log based relevance feedback scheme.

- To develop a new powerful context formation approach and demonstrate its efficiency in mining programmer interaction histories (MI).
- To develop a comparative framework that helps understand which factors have much influence on the recommendation performance.
- To develop significant improvement in performance of recommendation system by use of relevance feedback.

ISSN:2319-6890)(online),2347-5013(print)
1 July 2016

VIII. CONCLUSION

This paper inspected how the use of view information collected from programmer interaction histories, can help to provide detailed context to programmer to get more accurate, flexible and earlier edit recommendations. This system proposed new approach EMI which extends previous approach MI. Relevance feedback algorithm helps to improve the accuracy.

ACKNOWLEDGEMENT

I would wish to thank all the people who showered their kind support required for the whole analysis. It was very knowledge gaining and helpful for the further research.

VIII. REFERENCES

i. R. Robbes and M. Lanza, *"Characterizing and understanding development Sessions," Proc. 15th IEEE International Conf. on Program Comprehension (ICPC '07), IEEE Computer Society, Washington, DC, USA, 2007, pp. 155-166.*

ii. M. Kim and D. Notkin, *"Discovering and representing systematic code changes," Proc. 31st International Conf. on Software Engineering (ICSE '09), IEEE Computer Society, Washington, DC, USA, 2009, pp. 309-319.*

iii. Seonah Lee, Sungwon Kang, Sunghun Kim, and Matt Staats *"The Impact of View Histories on Edit Recommendations", IEEE Transactions on Software Engineering,(Volume:41 , Issue: 3),March 1 2015*

iv. R. Agrawal, T. Imielinski and A. N. Swami, *"Mining association rules between sets of items in large databases," - Proc. ACM D.C., May 26-28, 1993, ACM Press, pp. 20–216.*

v. T. Zimmermann, P. Weissgerber, S. Diehl and A. Zeller, *"Mining version histories to guide software changes," IEEE Transactions on Software Engineering, 31(6), 2005, pp. 429–445.*

vi. A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, *"Predicting source code changes by mining change history," IEEE Transactions on Software Engineering, 30, 2004, pp. 574–586.*

vii. D. Kim, Y. Tao, S. Kim and A. Zeller" *Where Should We Fix This Bug?A Two-Phase Recommendation Model" IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 39, NO. 11, NOVEMBER 2013*

viii. N. Sawadsky, G. C. Murphy and R. Jiresal, *"Reverb: Recommending code related web pages," Proc. of the 35th ACM/IEEE Int'l Conf. on Software Engineering, 2013.*

ix. S. Lee, S. Kang and M. Staats, *"NavClus: A graphical recommender for assisting code exploration," Proc. 35th International Conference on Software Engineering (ICSE), May 2013, pp. 1315- 1318.*

x. D. Piorkowski, S. Fleming, C. Scaffidi, C. Bogart, M. Burnett, B. John, R. Bellamy and C. Swart, *"Reactive information foraging: an empirical investigation of theory based recommender systems for programmers," Proc. ACM annual conference on Human Factors in Computing Systems. ACM, 2012, pp. 1471–1480.*