Developing Prediction Models to Assist Software Developers and Support Managers

Meera Sharma[1(✉)] and Abhishek Tondon[2]

[1] Department of Computer Science, University of Delhi, Delhi, India
meerakaushik@gmail.com
[2] SSCBS, University of Delhi, Delhi, India
abhishek.tondon@rocketmail.com

Abstract. A huge amount of historical information about the evolution of a software project is available in software repositories, namely bug repositories, source control repositories, archived communications, deployment logs, and code repositories. By mining the evolutionary history of a software, we have designed prediction models to assist software developers by predicting bug attributes like priority, severity, assignee and fix time. We have evaluated the uncertainty in the software in terms of entropy arises due to source code changes done in files of the software to fix different issues. To support software managers, we have designed prediction models to predict potential values of entropy and different issues, namely bugs, improvements in existing features (IMPs) and new features (NFs) over a long run. In this research work, we have developed mathematical models to assist software managers and developers in bug triaging, bug fixing and different software maintenance related tasks. Our work has been validated on issue and code change data of several open source projects, namely Eclipse, Open office, Mozilla and Apache.

1 Problem Statement and Its Importance

Advancement in internet and communication technologies has given impetus to the development and usage of Open Source Software (OSS). Various open source software repositories: source control repositories, bug repositories, archived communications, deployment logs, and code repositories are available online and easily accessible. These repositories help bug triager/developers in the bug resolving process and also in the evolution of the software products. Software repositories are helpful in understanding the development of a software. Different predictions can be made about the development of a software by using the information available in different software repositories. It will be again helpful for planning different aspects of the software evolution. The bug report history is stored in bug reporting and tracking systems (BugZilla, Jira, Mantis, Trac, Gnats, Fossil and Bugtracker.net) separately. Reporting a bug requires several parameters/attributes to be filled at the time of bug submission. Some attributes such as summary, reporter and submission date are entered during the initial submission and remain constant. Other attributes such as severity, priority, resolution, status, assignee and milestone are entered later on in the bug fixing process and can be updated.

DOI: 10.1007/978-3-319-62404-4_41

Bug priority describes about the urgency and importance of fixing a bug. It can be assigned into 5 levels from 1 to 5. Priority level 1 is the highest and priority level 5 is the lowest priority. Another important bug parameter is severity. Severity is one of the ordinal attribute which is used to measure the effect of the bug on the software functionality. Open source projects have used seven severity levels from 1 (Blocker) to 7 (Trivial). During bug fixing process, correct bug severity assignment and prioritization help in bug assignment/bug fix scheduling and resource allocation. In case of failure of this, resolution of important bugs will be delayed. A bug priority and severity prediction system will be helpful for software developers to fix reopened or newly reported bugs. In the available literature, several studies have been conducted for the proposal of severity and priority prediction [1–10]. Prediction of bug priority and severity needs historical data on which we can train the classifiers. But, unavailability of bug history for a software, especially for a new one, is a problem. In this situation, building prediction models based on cross project is the solution. No author has attempted to work on cross project validation in case of bug severity and priority prediction. In this paper, we have conducted a study to predict bug priority and severity by using bug summary (a brief description about the bug) in the context of cross project [27, 29, 33].

The absence of Association Rule Mining based study for bug fix time and assignee prediction motivated us to propose bug fix time and assignee prediction models [28, 30, 31]. To assist bug triaging process, Apriori algorithm has been used

- to predict the bug fix time by using the bug severity, priority, summary terms and assignee and
- to predict the assignee of a newly reported bug by using the bug severity, priority and summary terms.

In Open Source Software (OSS), users located at different geographical locations participate in the evolution of the software by reporting bugs and requesting for the new features (NFs) and improvements in the existing features (IMPs). Software developers do source code changes in files of the software to fix bugs or to meet the demands of the users. These source code changes in a given time period can be used to measure the potential complexity of code changes, i.e. possible code changes in the software over a long run. OSS evolves as a result of these modifications. An empirical understanding of different issues fixing and changes done in source code files can help software mangers to plan different evolutionary aspects of a software. We proposed mathematical models for prediction of potential bugs to be detected or fixed [32]. We have also predicted new features and improvements in existing features that can be incorporated in the software over a long run. The code change complexity has been quantified by using entropy based measures. Then, the potential entropy has been predicted by using Cobb-Douglas function [15]. We also extended Cobb-Douglas based diffusion models to incorporate the impact of different issues on the complexity of code changes. The rest of the paper is organized as follows: Sect. 2 gives the related literature. Section 3 presents the research questions deigned for this study. Proposed solution for the research questions and its novelty has been discussed in Sect. 4. Present status and future research plan have been presented in Sect. 5. Finally, the paper has been concluded in Sect. 6.

2 Related Work

In this section, we have presented an overview of the related studies. In text based bug severity assessment, studies have been conducted for bug severity prediction by using textual description, i.e. summary of bug [1–3]. In a study [4], NB, MNB, SVM, k-NN, J48, and RIPPER classifiers have been used for bug severity prediction and results showed that for closed source projects NB worked with significant accuracy performance and For open source projects, SVM worked well. A method has been proposed for the bug triage and the bug severity prediction [5]. The authors used historical bug reports in the bug repository for extracting topic(s) and then map the bug reports related to each topic. The authors identified corresponding reports having similar attributes: component, product, priority and severity, with the newly submitted bug report. After that severity for the bug and the most appropriate developer are recommended. Reference [6] suggested a concept profile-based severity prediction technique which first analyzed historical bug reports in the bug repositories and then build the concept profiles from them. Recently, a new approach has been proposed in [7] which assess the accuracy of bug severity prediction by taking into consideration the unreliable nature of the data. Results of this approach show that current prediction approaches perform well 77–86% agreement with human-assigned severity levels. Reference [22] predicted the priority of a bug during the software testing process using Artificial Neural Network (ANN) and Naïve Bayes classifier. References [9, 10] used a classification based approach to compare and evaluate the Support Vector Machine (SVM) and Naïve Bayes classifiers to automate the prioritization of new bug report by using the categorical and textual attributes of a bug report to train the model. They have shown that SVM performance was better than the Naïve Bayes with textual attributes and Naïve Bayes performance was better than SVM for categorical attributes. But this analysis has been carried out on a limited data and techniques. Cross project bug priority and severity prediction is a new and challenging task. No attempt has been made in available literature to discover association rules for different attributes of bug.

In OSS, source code needs to be changed frequently in order to fix different issues reported by different users. These changes done in the files of the software system makes the source code complex and buggy. Such source code changes can be measured by using various complexity measures proposed in the literature. In a study [23], complexity is defined on the basis of expanded system resources when the system performs a given task by interacting with a part of software. In case of a computer system, the complexity is defined by using the storage needed and execution time required to do the computation. If a programmer is considered as an interacting system then the difficulty faced by him in performing different tasks is used to measure the complexity. The task may be of coding, testing, debugging and modification. According to thermodynamics second law the disorder of a closed system remains constant or it increases, but it never reduces and entropy is defined as a measure of this disorder. The Entropy of an evolving system always increases [24]. As a software is modified or evolved, the source code changes and these changes lead to the uncertainty or randomness in the software which results in complex and buggy code. Entropy of a software is defined as a measure of this uncertainty or randomness or complexity of the source code. The first attempt to predict the bugs by using past defects has been made by [12]. The authors numerically evaluated the process of code change by using entropy based measures. Reference [13] proposed a model which predicts the code change complexity of a software over a long run by using entropy based measures. The authors also determined the rate of code change complexity diffusion. The historical code change data of various components of open source Mozilla wed browser have been used to validate the proposed model. In literature, [25, 26] have used 2-dimensional Cobb-Douglas function for software reliability modeling to consider the impact of efforts used in testing, testing time and testing coverage. The effect of different types of issues: bugs, NFs and IMPs, have not been considered so far for code change complexity prediction.

3 Research Questions

We have designed following research questions for our study

Research Question 1: Does training data from other projects provide acceptable bug priority and severity prediction results?

Research Question 2: Does the combination of training data sets provide better performance than single training data set?

Research Question 3: Can we discover association rules between different bug attributes that will assist the developers in bug assignment and prediction of bug fix time?

Research Question 4: Can we predict the potential entropy that can be diffused in the software over a period of time?

Research Question 5: Can we predict the potential bugs to be detected/fixed and NFs, IMPs that can be incorporated in the software over a long run?

4 Proposed Solution and Its Novelty

Active users located at different geographical locations, participate in the software evolution by reporting bugs and requesting for the new features (NFs) and improvements in the existing features (IMPs) as shown in Fig. 1.

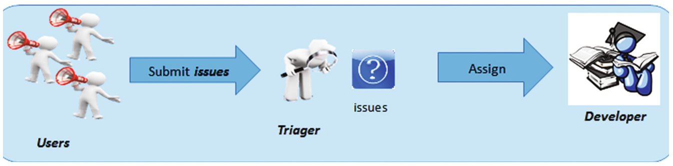
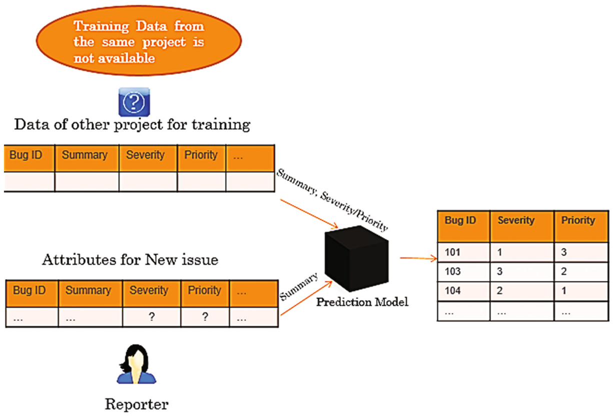


Fig. 1. Issues (bugs/NFs/IMPs) submission in open source software

Prediction models for bug priority and severity assist software developers in bug triaging. The prediction models need historical data on which we can train the classifiers. But, unavailability of bug history for a software, especially for a new one, is a problem. In this situation, building prediction models based on cross project is the solution [17–21]. In answer to Research Question 1, we proposed cross project bug priority and severity prediction models as shown in Fig. 2. We trained the classifiers with summary attribute of fixed bug reports (with known labels of priority and severity) of projects other than the testing projects.


Training Data from the same project is not available
Data of other project for training
Summary, Severity/Priority
Attributes for New issue
Summary
Prediction Model
Reporter

Bug ID	Summary	Severity	Priority	...

Bug ID	Summary	Severity	Priority	...
...	...	?	?	...

Bug ID	Severity	Priority
101	1	3
103	3	2
104	2	1
...	...	...

Fig. 2. Cross project bug priority and severity prediction

We have used text mining [1] to extract terms/features of bug summary attributes based on information gain criteria to train the classifiers.

In answer to Research Question 2, we used all the combinations consisting of maximum 3 datasets from projects other than the testing project. During making the combination of training datasets, we have excluded project which is going to be tested. The number of combinations consisting of less than or equal to N datasets for M available datasets/projects can be computed as shown below in Eq. (1).

$$\sum_{j=1}^{N} \frac{M!}{j!(M-j)!} \tag{1}$$

Example of generating training datasets for DB dataset has been shown in Fig. 3 [29].

Based on the accuracy and f-measure performance, the best training candidate for every testing dataset has been identified.

The absence of association rule based bug fix time and assignee prediction models motivated us to design Research Question 3. To assist bug triaging process, we used Apriori algorithm

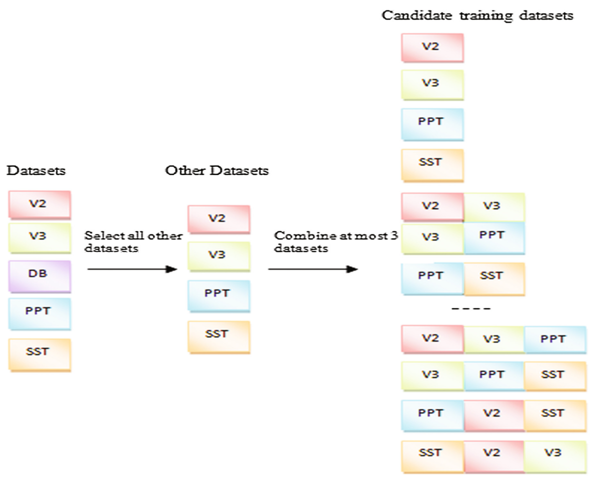


Fig. 3. Procedure of generating training datasets for DB dataset

- to predict the bug fix time by using bug severity, priority summary terms and assignee (Fig. 4) [28] and
- to predict the assignee of a newly reported bug by using bug severity, priority and summary terms (Fig. 5) [30, 31].

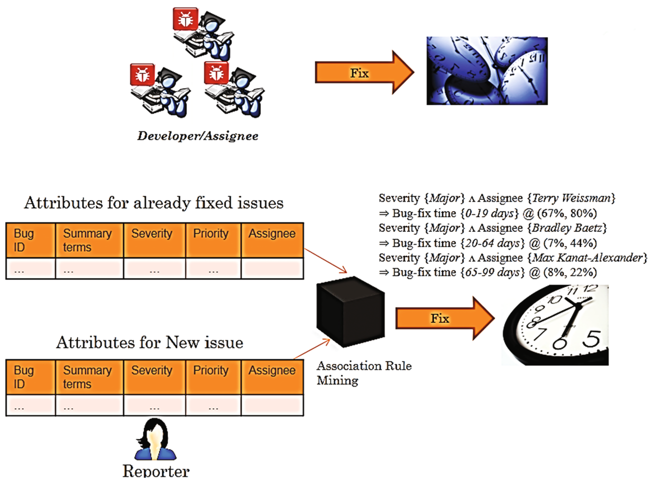

Fix
Developer/Assignee
Attributes for already fixed issues
Bug ID | Summary terms | Severity | Priority | Assignee
Severity {Major} ∧ Assignee {Terry Weissman} ⇒ Bug-fix time {0-19 days} @ (67%, 80%)
Severity {Major} ∧ Assignee {Bradley Baetz} ⇒ Bug-fix time {20-64 days} @ (7%, 44%)
Severity {Major} ∧ Assignee {Max Kanat-Alexander} ⇒ Bug-fix time {65-99 days} @ (8%, 22%)
Fix
Association Rule Mining
Attributes for New issue
Bug ID | Summary terms | Severity | Priority | Assignee
Reporter

Fig. 4. Association rule Mining based Bug fix time Prediction

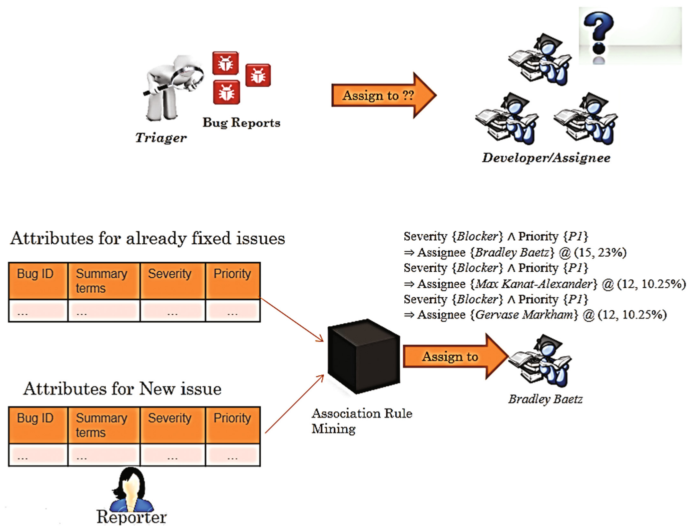

Triager
Bug Reports
Assign to ??
Developer/Assignee
Attributes for already fixed issues
Bug ID | Summary terms | Severity | Priority
Attributes for New issue
Bug ID | Summary terms | Severity | Priority
Reporter
Association Rule Mining
Severity {Blocker} ∧ Priority {P1} ⇒ Assignee {Bradley Baetz} @ (15, 23%)
Severity {Blocker} ∧ Priority {P1} ⇒ Assignee {Max Kanat-Alexander} @ (12, 10.25%)
Severity {Blocker} ∧ Priority {P1} ⇒ Assignee {Gervase Markham} @ (12, 10.25%)
Assign to
Bradley Baetz

Fig. 5. Association rule Mining based Bug Assignee Prediction

Frequent code changes are required to fix different issues reported by users, which results in complex code as shown in Fig. 6.

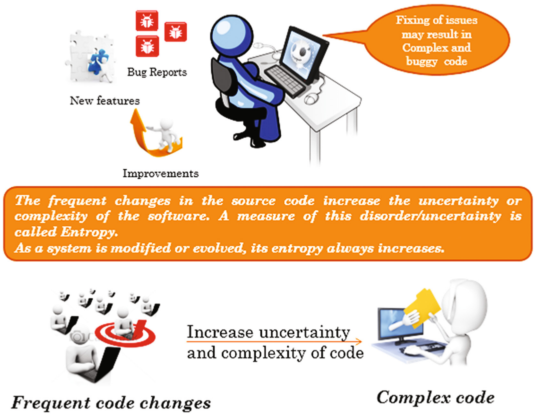

Bug Reports
New features
Improvements
Fixing of issues may result in Complex and buggy code
The frequent changes in the source code increase the uncertainty or complexity of the software. A measure of this disorder/uncertainty is called Entropy.
As a system is modified or evolved, its entropy always increases.
Increase uncertainty and complexity of code
Frequent code changes
Complex code

Fig. 6. Frequent changes make source code complex

The source code changes done in the software in order to fix different issues are quantified using entropy based measures and termed as complexity of code changes. The quantification of code changes is based upon the Shannon entropy [11] as given in Eq. (2):

$$E_n(P) = -\sum_{j=1}^{n} (P_j * \log_2 P_j), \quad P_j \geq 0 \quad \sum_{j=1}^{n} P_j = 1 \tag{2}$$

The probability of occurrence of a change is denoted by P_j, where P_j is defined as follows:

$$P_j = \frac{no.\ of\ times\ j^{th}\ file\ changed\ in\ a\ period}{total\ no.\ of\ changes\ done\ in\ all\ the\ files\ in\ that\ period}$$

The concept of entropy to measure the complexity of code changes has been used by [12] as given in Eq. (3).

$$\begin{aligned} E(P) &= \frac{1}{Max\ Entropy\ for\ Distribution} * E_n(P) \\ &= \frac{1}{\log_2 n} * E_n(P) = \frac{1}{\log_2 n} * -\sum_{j=1}^{n} \left(p_j * \log_2 p_j\right) \\ &= -\sum_{j=1}^{n} \left(p_j * \log_n p_j\right) \\ \text{where}\ & p_j \geq 0, \forall j \in 1, 2, \ldots, n \quad and \sum_{j=1}^{n} p_j = 1. \end{aligned} \tag{3}$$

As the entropy (E) depends on that how many files the system have (n), to prevent the reduction in entropy due to the rarely modified files, to get the effective entropy, the number of recently modified files has been used to divide [12, 13]. Fifty source code changes done in a single file are easy to recall instead of fifty single changes done in fifty different files.

There is a challenge before us to determine the future requests that can come from different users and to determine the potential source code change that can be done in the software over a long run. We need to link all the issue implementations with the source code changes of the software.

In answer of the Research Question 4, we proposed models along the line of [13, 14] for potential entropy prediction. We defined r as the rate at which entropy is diffused in the source code due to NFs or IMPs and s is defined as the rate at which entropy is diffused due to bug fixes.

We have made the following assumptions for the proposed models:

- Potential entropy remains same.
- The effect of NFs/IMPs on entropy is independent and may generate bugs.
- At t = 0, entropy is zero.

The entropy diffusion per unit time can be written as follows:

$$\frac{d(E(t))}{dt} = r\left(\overline{E} - E(t)\right) + s\frac{E(t)}{\overline{E}}\left(\overline{E} - E(t)\right) \tag{4}$$

where $\overline{E}$ is the potential entropy to be diffused in software and $E(t)$ is the amount of entropy at any given time t. We solved Eq. (4) with $t = 0$ and $E(0) = 0$, we get

$$E(t) = \overline{E}\left[\frac{1 - e^{-(r+s)t}}{1 + \frac{s}{r}e^{-(r+s)t}}\right]$$

or

$$E(t) = \overline{E}\left[\frac{1 - e^{-\alpha t}}{1 + \gamma e^{-\alpha t}}\right] \tag{5}$$

Here $\alpha = r + s$ is the rate at which entropy is diffused due to source code changes and $\gamma = \frac{s}{r}$ is a constant.

We have extended the Cobb-Douglas type function [15] to consider the effect of NFs and IMPs on the entropy. In Eq. (6), v denotes IMPs and w denotes NFs, and μ is the degree of the impact on the entropy.

$$t \equiv v^{\mu}w^{1-\mu} \quad (0 \le \mu \le 1) \tag{6}$$

We have proposed a model to measure the entropy depending on two factors: NFs and IMPs, and for that we extended t of Eq. (5) by using Eq. (6) as

$$E(v,w) = \overline{E}\left[\frac{1 - e^{-\alpha(v^{\mu} * w^{1-\mu})}}{1 + \gamma e^{-\alpha(v^{\mu} * w^{1-\mu})}}\right] \tag{7}$$

We have extended the Cobb-Douglas type function [15] to incorporate the effect of NFs, IMPs and bugs on the entropy of the software. In Eq. (8), v, w and x denote the IMPs, NFs and bugs. α and γ are the effect of NFs and IMPs on the entropy.

$$t \equiv v^{\mu}w^{\lambda}x^{1-(\mu+\lambda)} \quad (0 \le \mu \,\text{and}\, \lambda \le 1) \tag{8}$$

We model to measure the effect of NFs, IMPs and bugs on the entropy diffusion simultaneously by extending t of Eq. (5) by using Eq. (8) as

$$E(v,w,x) = \overline{E}\left[\frac{1 - e^{-\alpha(v^{\mu} * w^{\lambda} * x^{1-(\mu+\lambda)})}}{1 + \gamma e^{-\alpha(v^{\mu} * w^{\lambda} * x^{1-(\mu+\lambda)})}}\right] \tag{9}$$

In answer to Research Question 5, we considered 'a' as the potential issues: bug, NFs, IMPs and NFs + IMPs, that are reported in the software over a period of time and this follows nonhomogeneous poison process. In line of [14, 16], the following equation has been written. We considered the logistic rate, i.e. c, for diffusion of NFs/IMPs and bug detection in the software.

$$\frac{d}{dt}m(t) = \frac{c}{1+\gamma \exp\left(-ct\right)}\left(a - m(t)\right)$$

where $m(t)$ is the cumulative value of bug/NFs/IMPs/(NFs + IMPs) at any time t.

In above equation, we take $m(t) = 0$ at t = 0, which results in Eq. (10)

$$m(t) = a\left[\frac{1-\exp(-ct)}{1+\gamma \exp(-ct)}\right] \tag{10}$$

Above model in Eq. (10) is used for prediction of potential bugs/NFs/IMPs/(NFs + IMPs) at any given time.

Validation of the proposed prediction models has been done by using historical data for several open source projects, namely Eclipse, Open office, Mozilla and Apache. Results show that bug priority prediction accuracy in cross project context is better than within project. Cross project priority prediction gives accuracy above 72% for SVM, k-NN and NNET. For combined training datasets, we get accuracy which is above 73% for all cases. Combined cross-project datasets result in improved f-measure performance in 12 cases and improved accuracy performance in 11 cases out of 20 total cases across all the 4 classifiers. SVM and NNET are better than k-NN and k-NN is better than Naïve Bayes in terms of accuracy performance.

The results of cross project bug severity prediction show that Combined cross-project datasets result in improved f-measure performance in 12 cases and improved accuracy performance in 10 cases out of 21 total cases across all the 3 classifiers.

Non-parametric Mann-Whitney U test has shown a statistical significant difference in f-measure performance of different training candidates for a particular testing dataset. The two ensemble approaches: ***Vote*** and ***Bagging***, have improved the f-measure performance up to 5% and 10% respectively for the severity levels having less number of bug reports in comparison of major severity level.

Association rule mining based prediction models for bug fix time and assignee predictions reveal rules that will assist developers in bug triaging and fixing process. Results show that proposed models for prediction of entropy and different issues submitted by users give a high goodness of fit for various performance measures, namely R^2, Bias, Variation, MSE and RMSPE.

5 Current Status and Future Plans

All the proposed prediction models have been described in detail in published papers [27–33]. In future, we will propose mathematical models to incorporate the unique paradigms of OSS development, such as the multiple releases property and rate of issues fixed by different types of contributors (innovators and imitators). In OSS, a number of contributors are innovative in nature in the sense that they work without their own implicit requirements, but for the evolution of the software product. We consider them as innovators (core developers) and others may be imitators. Issues are fixed by the innovators and a proportion of issues fixed, also causing to fix more issues by imitators.

A proportion of issues which are fixed by the innovators gave an impetus to fix more issues (imitators). We will also develop a mathematical model to embody the open source software development. The model will be based on the rate at which new features, feature improvements are added and the rate at which bugs are generated from these additions. We will extend the proposed model by considering the complexity of code changes (entropy) and discuss the release time planning of the software using the proposed model by maximizing the user's satisfaction level subject to fixing of issues up to a desired level. The proposed decision model will assist management to appropriately measure the software reliability, i.e. the issues left in the software and to determine the optimal release-update time for open source software.

6 Conclusion

In the paper, prediction models have been proposed to assist software developers and support managers in software maintenance and evolution. Bug reports of open source projects have been analyzed to predict bug severity and priority in cross projects context using text mining and machine learning techniques. We concluded that historical data of other projects developed in open source environment is better priority predictor and priority prediction in cross project context is working well. Results indicate that cross-project bug severity prediction is a serious challenge, i.e. simply using training data from projects in the same domain does not lead to accurate predictions. More research is needed to find out how to best describe the selection of a training project automatically. The association rule mining can be used to mine the association rules for bug fix time and assignee prediction. The proposed models to predict the potential of different issues and the entropy fit the observed data.

References

1. Menzies, T., Marcus, A.: Automated severity assessment of software defect reports. In: International Conference on Software Maintenance, pp. 346–355. IEEE (2008)
2. Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B.: Predicting the severity of a reported bug. In: Mining Software Repositories, pp. 1–10. MSR (2010)
3. Lamkanfi, A., Demeyer, Soetens, Q.D., Verdonck, T.: Comparing mining algorithms for predicting the severity of a reported bug. In: 15th European Conference on Software Maintenance and Reengineering, pp. 249–258. IEEE (2011)
4. Chaturvedi, K.K., Singh, V.B.: An empirical comparison of machine learning techniques in predicting the bug severity of open and close source projects. Int. J. Open Source Softw. Process. **4**(2), 32–59 (2013)
5. Yang, G., Zhang, T., Lee, B.: Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: Computer Software and Applications Conference (COMPSAC), pp. 97–106. IEEE (2014)
6. Zhang, T., Yang, G., Lee, B., Chan, A.T.: Predicting severity of bug report by mining bug repository with concept profile. In: 30th Annual ACM Symposium on Applied Computing, pp. 1553–1558, April 2015

7. Tian, Y., Ali, N., Lo, D., Hassan, A.E.: On the unreliability of bug severity data. Empirical Softw. Eng. **21**(6), 2298–2323 (2015)
8. Zhang, T., Chen, J., Yang, G., Lee, B., Luo, X.: Towards more accurate severity prediction and fixer recommendation of software bugs. J. Syst. Softw. **117**, 166–184 (2016)
9. Kanwal, J., Maqbool, O.: Managing open bug repositories through bug report prioritization using SVMs. In: Proceedings of the International Conference on Open-Source Systems and Technologies, Lahore, Pakistan (2010)
10. Kanwal, J., Maqbool, O.: Bug prioritization to facilitate bug report triage. J. Comput. Sci. Technol. **27**(2), 397–412 (2012)
11. Shannon, C.E.: A mathematical theory of communication. Bell Syst. Tech. J. **27**(379–423), 623–656 (1948)
12. Hassan, A.E.: Predicting faults based on complexity of code change. In: International Conference on Software Engineering, pp. 78–88 (2009)
13. Chaturvedi, K.K., Kapur, P.K., Anand, S., Singh, V.B.: Predicting the complexity of code changes using entropy based measures. Int. J. Syst. Assur. Eng. Manage. **5**, 155–164 (2014)
14. Bass, F.: A new product growth model for consumer durables. Manage. Sci. **15**, 215–227 (1969)
15. Varian, H.R.: Intermediate Microeconomics — A Modern Approach. W.W. Norton & Company, New York (1991)
16. Bittanti, S., Bolzern, P., Pedrotti, E., Pozzi, M., Scattolini, R.: A flexible modelling approach for software reliability growth. In: Bittanti, S. (ed.) Software Reliability Modelling and Identification. LNCS, vol. 341, pp. 101–140. Springer, Heidelberg (1988). doi:10.1007/BFb0034288
17. Zimmermann, T., Nagappan, N., Gall, H., Giger, E., Murphy, B.: Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp. 91–100 (2009)
18. Turhan, B., Menzies, T., Bener, A.: On the relative value of cross-company and within-company data for defect prediction. Empirical Softw. Eng. **14**(5), 540–578 (2009)
19. Ma, Y., Luo, G., Zeng, X., Chen, A.: Transfer learning for cross-company software defect prediction. Inf. Softw. Technol. **54**(3), 248–256 (2011). Science Direct, Elsevier
20. He, Z., Fengdi, S., Ye, Y., Mingshu, L., Qing, W.: An investigation on the feasibility of cross-project defect prediction. Autom. Softw. Eng. **19**, 167–199 (2012). Springer
21. Peters, F., Menzies, T., Marcus, A.: Better cross company defect prediction. In: 10th IEEE Working Conference on Mining Software Repositories (MSR), pp. 409–418. IEEE (2013)
22. Yu, L., Tsai, W., Zhao, W., Wu, F.: Predicting defect priority based on neural networks. In: 6th International Proceedings on Advanced Data Mining and Applications, pp. 356–367, Wuhan, China (2010)
23. Basili, V.R.: Qualitative software complexity models: a summary. In: Tutorial on Models and Methods for Software Management and Engineering. IEEE Computer Society Press, Los Alamitos, California (1980)
24. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object Oriented Software Engineering: A Use Case Driven Approach. ACM Press, Addison Wesley, pp. 69–70 (1992)
25. Inoue, S., Yamada, S.: Two-dimensional software reliability measurement technologies. In: IEEE IEEM, pp. 223–227 (2009)
26. Kapur, P.K., Pham, H., Gurjeet, A.G.: Two dimensional multi-release software reliability modeling and optimal release planning. IEEE Trans. Reliab. **61**(3), 758–768 (2012)
27. Singh, V.B., Misra, S., Sharma, M.: Bug severity in cross project context and identifying training candidates. J. Inf. Knowl. Manage. **16**(1), 30 (2017). World Scientific Publishing

28. Sharma, M., Kumari, M., Singh, V.B.: The way ahead for bug-fix time prediction. In: 3rd International Workshop on Quantitative Approaches to Software Quality (QuASoQ), co-located with 22nd Asia-Pacific Software Engineering Conference (APSEC 2015), New Delhi, India, pp. 31–38, 1–4 December 2015
29. Sharma, M., Bedi, P., Singh, V.B.: An empirical evaluation of cross project priority prediction. Int. J. Syst. Assur. Eng. Manage. **5**(4), 651–663 (2014). Springer
30. Sharma, M., Singh, V.B.: Clustering-based association rule mining for bug assignee prediction. Int. J. Bus. Intell. Data Min. **11**(2), 130–150 (2016)
31. Sharma, M., Kumari, M., Singh, V.B.: Bug assignee prediction using association rule mining. In: Gervasi, O., Murgante, B., Misra, S., Gavrilova, M.L., Rocha, A.M.A.C., Torre, C., Taniar, D., Apduhan, B.O. (eds.) ICCSA 2015. LNCS, vol. 9158, pp. 444–457. Springer, Cham (2015). doi:10.1007/978-3-319-21410-8_35
32. Singh, V.B., Sharma, M.: Prediction of the complexity of code changes based on number of open bugs, new feature and feature improvement. In: 25th IEEE International Symposium on Software Reliability Engineering (ISSRE), WOSD, Neples, Italy, pp. 478–483 (2014)
33. Sharma, M., Bedi, P., Chaturvedi, K.K., Singh, V.B.: Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: International Conference on Intelligent Systems Design and Applications (ISDA), pp. 539–545. IEEE (2012)