Predicting Change Request Severity Level Based on Random Forest Machine Learning Algorithm

|  |  |  |  |
| --- | --- | --- | --- |
| Luiz Alberto Ferreira Gomes Institute of Computing University of Campinas - UNICAMP Brazil gomes.luiz@ic.unicamp.br | | | Mario Lúcio Côrtes Institute of Computing University of Campinas - UNICAMP Brazil cortes@ic.unicamp.br |
|  | | |  |
|  |  |

ABSTRACT

No context de Change Request (CR) systems, the severity level of a change request is considered a critical variable in the planning of the software maintenance activities, indicating as soon a CR needs to be fixed. However, the severity level assignment remains a process essentially manual depending on the experience and expertise who has reported the CR. In this paper, we present the preliminary findings of research aim to predict the severity level of a CR by analyzing its textual description using text mining and Random Forest algorithms. The initial results evidence that these algorithms can predict the severity level with reasonable accuracy whereas a training data set with sufficient size is available.

CCS CONCEPTS

• **Software Maintenance** → **Change Request Systems**; *Issue Severity* • **Machine Learning** → Random Forest.

KEYWORDS

ACM proceedings, text tagging

ACM Reference format:

G. Gubbiotti, P. Malagò, S. Fin, S. Tacchi, L. Giovannini, D. Bisero, M. Madami, and G. Carlotti. 1997. SIG Proceedings Paper in word Format. In *Proceedings of ACM Woodstock conference, El Paso, Texas USA, July 1997 (WOODSTOCK’97)*, 4 pages.

DOI: 10.1145/123 4

1 INTRODUCTION

Change Request (CR) systems has been performed a key role in maintenance process in many software development settings, both in Close Source Software (CSS) and in Open Source Software (OSS) scenarios. Especially in the latter, it is characterized by the existence of many of users and developers with distinct expertise levels spread out around the world, which might register or deal with any amount of change requests[1].

Normally, a user interacts with a CR system often through a simple mechanism called CR form. This form enables him to request changes, to report bugs or to ask for support in a software product[2]. Initially, he or she should inform a short description, a long description, a type (e.g. bug, new feature, improvement and task) and a severity level (e.g. blocker, critical, major, minor and trivial) for his or her solicitation. Subsequently, a development team member will review this request and, if it is not refused for some reason (e.g. request duplication), he or she will complete the information in CR form, indicating, for example, its priority and the person responsible to accomplish it.

The severity level informationis recognized as a critical variable in the equation to estimate a prioritization of change request prioritization[3]. Consequently, it can be a decisive factor how soon it needs to be fixed [4]. However, the severity level assignment remains an essentially manual process which relies on experience and expertise of the person who has opened the CR [1,3,4,5]. So, it may allow a high degree of subjectivity and, consequently, it may be quite error-prone.

The number of CR made is frequently very high in large and medium software projects[5]. Severity level shifts throughout CR lifecycle (Figure 1) could cause important impacts on the maintenance activities planning and could drive the development team to solve the least important change requests before the most important ones. In this scenario, the support of a computational tool to help the user to assign and the development team to verify the CR severity level well desirable.

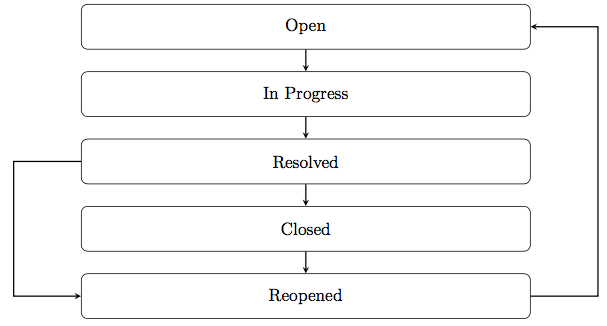


Figure 1: Change Request life cycle.

This paper presents the preliminary results of a research whose the objective is to implement an automatic mechanism, based on random forest machine algorithm, aim to aid users, developers and managers to assign the CR severity level more accurately. To achieve this, we investigated three research questions enumerated below:

1. **RQ1**: Is it possible to predict whether the change request severity degree will shift during its lifecycle? Based on fields filling by users in CR form, we have investigated if it is possible to predict whether the serevity level of request will be changed or not.
2. **RQ2**: Is it possible to predict whether the change request severity degree will increase or decrease during its lifecycle? Futhermore, we have investigated if it is possible to predict whether the serevity level of request will be increased or decreased.
3. **RQ3**: Is it possible to predict the change request severity degree at the end of its lifecycle? Finally, we have investigated if it is possible to predict the serevity level of request based on the same information used in the two previous questions.

The structure of this paper is as follows. In Section 2, we describe some related works. In Section 3, we provide the information background about CR systems, text mining and machine learning techniques necessary to understand our approach. Section 4 describe our work. Findings and discussions are presented in Section 5. Finally, we conclude and discuss future work in Section 6.

2 RELATED WORKS

Menzies[6] have been developed a method, named SEVERIS (SEVERity ISsue assessment), for evaluating the severity of changes requests. SEVERIS is based on common text mining techniques (e.g. tokenization, stop word removal, stemming, Tf\*Idf and InfoGain) and on the data mining techniques (e.g. RIPPER). The method was applied on five projects managed by the Project and Issue Tracking System (PITS). - an issue tracker system used by NASA.

Lamkanfi et al.[4] developed an aproach to predict severity level of a CR based on text mining algorithms (tokenization, stop word removal, stemming) and on the Naïve Bayes machine learning algorithm. They have been validated their approach over from three open source project Mozilla, Eclipse and GNOME and they accomplished that a training set with approximately 500 change requests per severity degree are enough to predict it with a reasonable accuracy. In another work, Lamkanfi et al. [5], these authors compared the accrual of four machine-learning algorithms (Naïve Bayes Multinomial, K-Nearest Neighbor, and Support Vector Machine). They have been concluded that Naïve Bayes Multinomial gave superior performance compared to the others proposed algorithms.

3 BACKGROUND

In this section, we describe de change request process, next we explain the common approach to pre-processing textual documents, and lastly we highlight the Random Forest algorithm to classify and predict the CR severity level.

3.1 Change Request Systems

Change Request systems[2] are softwares employed to keep the recording and tracking information of requests for modifications, bug fixes, and support that could occur during the software life cycle.

Although there is no a common sense regarding the terminology or the amount of information that users must fill in to complete his requisition between popular CR systems (eg. Bugzilla, Jira, and Redmine), typically, they shall fill in a form containing at least the following fields (Table 1)[3]:

Table 1: Common fields in CR form.

|  |  |
| --- | --- |
| Type | Type of request (e.g. bug, new feature, improvement, and new feature) |
| Title | Short description of request in one line. |
| Description | Long and detailed description of request in many lines. It could include source code snippets and stack tracing reports. |
| Severity | Level of severity of request (e.g. blocker, critical, major, minor and trivial). |

Once the request has been registered by the user, the development team will assess it and, if it not canceled for some reason (e.g. duplication), they will complement the information with, for example, the person responsible to handle this request. All these data are stored in a repository, keeping important historical information about a certain software.

3.2 Text mining

Text mining is the process to convert unstructured text into a structure suited to analysis[7]. It is composed of three basic activities: tokenization, stop word removal and stemming.

Tokenization is the action to parsing a character stream into a sequence of tokens by splitting the stream at delimiters. In this context, a token is defined as a block of text or a string of characters (without delimiters such as spaces and punctuation) that is recognized as useful portion of the unstructured data.

Stop words eliminates commonly used words that do not provide relevant information to a specific context, including prepositions, conjunctions, articles, common verbs, nouns, pronouns, adverbs, and adjectives (e.g. "is", "am", "are", "I" and etc.)

Stemming is the process stemming is the process of reducing or normalizing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form (e.g. “working” and worked into work).

3.3 Random Forest

The Random Forest algorithm[8] relies on two core principles: (i) in the the creation of hundreds of decision trees and the joining them into a single model; and (ii) in the closing decision based on the decision of the majority of the forming trees which are treated as equals.

A random forest model is considered a suited alternative for model construction for a many of reasons [9]:

* Requires little or no data preprocessing, no data normalization and is resilient to outliers.
* Requires no variable selections because the algorithm does its own.
* Models resultants from each tree in the forest tend not to overfit to the training dataset, because they are built using two levels of randomess (observations and variables).

4 EXPERIMENT

4.1 Dataset

4.2 Evaluation Measures

5 FINDINGS AND DISCUSSIONS

4 CONCLUSIONS

ACKNOWLEDGMENTS

REFERENCES

[1] Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. do C. Machado, T. F. Vale, E. S. de Almeida, and S. R. de L. Meira, “Challenges and opportunities for software change request repositories: a systematic mapping study,” *J. Softw. Evol. Process*, vol. 26, no. 7, pp. 620–653, Jul. 2014.

[2] I. Sommerville, *Software Engineering*. 2010.

[3] Y. Tian, D. Lo, and C. Sun, “Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction,” in *2012 19th Working Conference on Reverse Engineering*, 2012, pp. 215–224.

[4] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, “Predicting the severity of a reported bug,” *Proc. - Int. Conf. Softw. Eng.*, pp. 1–10, 2010.

[5] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonckz, “Comparing mining algorithms for predicting the severity of a reported bug,” *Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR*, pp. 249–258, 2011.

[6] T. Menzies and A. Marcus, “Automated severity assessment of software defect reports,” *IEEE Int. Conf. Softw. Maintenance, 2008. ICSM 2008*, pp. 346–355, 2008.

[7] R. Feldman and J. Sanger, *The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data*. Cambridge University Press, 2007.

[8] L. Breiman, “Random Forests,” *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[9] G. Williams, *Data Mining with Rattle and R: The Art of Excavating Data for Knowledge Discovery*. 2011.