Research Preview: Automatic Data Categorization in Issue Tracking Systems

Thorsten Merten[1] and Barbara Paech[2]

[1] Hochschule Bonn-Rhein-Sieg
Fachbereich Informatik
Grantham-Allee 20
53757 Sankt Augustin
thorsten.merten@h-brs.de

[2] Universität Heidelberg
Software Engineering Group
Im Neuenheimer Feld 326
69120 Heidelberg
paech@informatik.uni-heidelberg.de

Abstract: Issues in an issue tracking system contain different kinds of information like requirements, features, development tasks, bug reports, bug fixing tasks, refactoring tasks and so on. This information is generally accompanied by discussions or comments, which again are different kinds of information (e.g. social interaction, implementation ideas, stack traces or error messages).

We propose to improve automatic categorization of this information and use the categorized data to support software engineering tasks. We want to obtain improvements in two different ways. Firstly, we want to obtain algorithmic improvements (e.g. natural language processing techniques) to retrieve and use categorized auxiliary data. Secondly we want to utilize multiple task-based categorizations to support different software engineering tasks.

1 Problem definition

Most software companies use an issue tracking system[1] (ITS) [ST11] to support software engineering (SWE) work. ITS contain diverse categories of information like requirements, features, development tasks, bug reports, bug fixing tasks, refactoring tasks and so on.

However, for specific SWE tasks only subsets and parts of issues are relevant. For example a developer might be interested in stack traces and technical information in a bug report issue to fix the bug. But then a manager or team leader may be interested in different information like the affected features and components to plan resources. Thus, even parts of a single issue fit in different categories, depending on the [BDDL12] users information needs and the software engineering tasks the user wants to accomplish.

Although ITS provide the ability to manually categorize issues, not every role gets in touch with an issue before actually resolving the issue or executing other tasks, which require data from the ITS. Therefore, manual categorization is cumbersome and inefficient unless tasks requiring the same data from the ITS are recurring.

[1] Also: bug tracker, ticket system, helpdesk or project management software

2 Solution idea

We propose to solve this problem considering the methodology of design science [MS95], which basically consists of a three step process. The following list describes the process and shows how we will apply it to the problem of ITS categorization:

1. *Identify practical problems within a certain setting*:
   We identify and define different tasks in SWE which are supported by ITS. The above bug fixing example is such a task. Then we create a hypothesis how the task can be improved by categorization of ITS data. An example for bug fixing is to separate technical information (e.g. stack traces, log file excerpts or source code) from natural language texts, since a developer may need the problem description and the technical information only, to fix the task.

2. *Create artifacts which hypothetically improve the identified problems*.
   We create an artifact/solution that improves the problem identified above. In the bug fixing example this artifact is an algorithm that recognizes and differentiates (categorizes) the items natural language, code, stack trace and log file.

3. *Study the influence of the introduced artifacts in the setting*.
   We than evaluate how the situation has been influenced by the second step. In the bug fixing example we have to validate two things: Firstly, we want to know how well the separation works on different data, which could be measured e.g. in precision and recall. Secondly, we want to know how the separation influenced the SWE process, which could be measured e.g. by surveying developers.

3 Detailed Example and Related Work

The following example described in detail how we combine different natural language processing (NLP) algorithms. We use these combinations as preprocessing in order to improve categorization.

Sticking to bug fixing example described above, we know that the separation of technical items and natural language texts has been done for developer mailing list classification [BDDL12] e.g. to extract source code descriptions [PAD+12]. These specific techniques can separate technical items and natural language properly in ITS. For other techniques like naive Bayes classifiers the same technical information can be considered noise[2]. Preprocessing noisy data is a very important step in todays real life databases for any forms of data mining [HKP11, pp. 84]. Separating and removing technical information from ITS data reduces the noise in natural language texts, which itself improves the quality of semi supervised learning techniques.

[2] Actually, our goal is to separate technical information from natural language texts and treat both differently. For simplicity we consider the information noise in this example

4 Conclusion and Future Work

The categorization of ITS data is a complicated task. Text in issues suffers from inconsistencies like typos or ambiguity [KC11]. Furthermore, data in the same database field of an ITS varies from natural language texts to technical information.

Our idea is to combine multiple, very specific NLP approaches for data preprocessing. Examples are the approach described in Section 3 or rule-based algorithms (e.g. to extract requirements from ITS data [ZL11, VR12]). The preprocessed, less noisy data will then be fed in semi-supervised learning techniques to achieve better categorization results.

Currently, our work on this problem is twofold. Firstly, we explore which tasks and corresponding problems can benefit from text categorizations. Secondly, different NLP and supervised learning algorithms are tested for categorizations. In future work we will validate our combined approach in different case-studies.

References

[BDDL12] Alberto Bacchelli, Tommaso Dal Sasso, Marco D'Ambros, and Michele Lanza. Content classification of development emails. *2012 34th International Conference on Software Engineering (ICSE)*, pages 375–385, June 2012.

[HKP11] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, Third Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 3rd edition, 2011.

[KC11] Andrew J. Ko and Parmit K. Chilana. Design, discussion, and dissent in open bug reports. In *Proceedings of the 2011 iConference*, pages 106–113, New York, New York, USA, 2011. ACM Press.

[LDSV11] Ahmed Lamkanfi, Serge Demeyer, Quinten David Soetens, and Tim Verdonck. Comparing Mining Algorithms for Predicting the Severity of a Reported Bug. In *2011 15th European Conference on Software Maintenance and Reengineering*, pages 249–258. IEEE, March 2011.

[MS95] S T March and G F Smith. Design and natural science research on information technology. *Decision Support Systems*, 15(4):251–266, 1995.

[PAD+12] Sebastiano Panichella, Jairo Aponte, Massimiliano Di Penta, Andrian Marcus, and Gerardo Canfora. Mining source code descriptions from developer communications. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, pages 63–72, June 2012.

[ST11] Ian Skerrett and The Eclipse Foundation. The Eclipse Community Survey 2011, 2011.

[VR12] RE Vlas and WN Robinson. Two Rule-Based Natural Language Strategies for Requirements Discovery and Classification in Open-Source Software Development Projects. *Journal of Management Information Systems*, pages 1–39, 2012.

[ZL11] Tao Zhang and Byungjeong Lee. A Bug Rule Based Technique with Feedback for Classifying Bug Reports. In *2011 IEEE 11th International Conference on Computer and Information Technology*, pages 336–343. IEEE, August 2011.