Semantic tools for improving software development in open source communities

Gregor Leban

Jožef Stefan Institute, Ljubljana, Slovenia
gregor.leban@ijs.si

Abstract. Software development communities use different communication channels such as mailing lists, forums and bug tracking systems. These channels are not integrated which makes finding information difficult and inefficient. As a result of the ALERT project we developed a system that is able to collect and annotate information from various communication channels and store it in a single knowledge base. Using the stored knowledge the system can provide users valuable functionalities such as semantic search, finding potential bug duplicates, custom notifications and issue recommendations.

Keywords: information extraction, semantic search, software development, open source, data integration

1 Introduction

Large open source communities frequently use several communication channels for information exchange. Detected bugs are reported on bug tracking systems such as Bugzilla and Mantis. Forums and mailing lists are used for exchanging general discussions and questions about various aspects of the developed software. Source code is shared and updated using source code management systems such as git and svn. Project documentation is often written using a platform such as wiki.

Due to use of various information channels these communities face different challenges. A common problem is finding information. In order to find an answer to a question, the user has to use different search interfaces of these channels to find information of interest. Additionally, since computer science terminology contains many synonyms and abbreviations the search has to be repeated several times using different keywords. Because of the lack of integration, duplicates of the same questions are frequently asked on different information channels. Although some user generated data contains structured information it is not available to the user. For example, a developer is not able to see who are the developers who in the past modified a particular method or class. Such functionality can be crucial in identifying who could be responsible for and therefore fix a particular bug. For large communities such as KDE or Eclipse, the amount of generated content can also be a problem. When hundreds of posts and bugs are reported each day it is time consuming for a user to sweep though the content and find the relevant information.

In order to alleviate such problems we developed a system called ALERT that will be described in the rest of the paper. We will start by describing the system architecture, followed by main features of the system and finish with a conclusion.

2 System architecture

In order to integrate information from different information sources we developed a set of sensors that are able to detect whenever new information is available in the source. Our sensors can import data from five different types of sources: bug tracking systems (BTS), mailing lists, forums, source code management systems (SCM) and wikis. For each data source we collect all available information. In case of a bug report, for example, we obtain the title and description of a bug as well as the author name, time, the assigned product and component information, status and resolution of the bug, etc. In case of a code commit we collect the comment of the commit, time and author as well as extract all the methods modified in the commit.

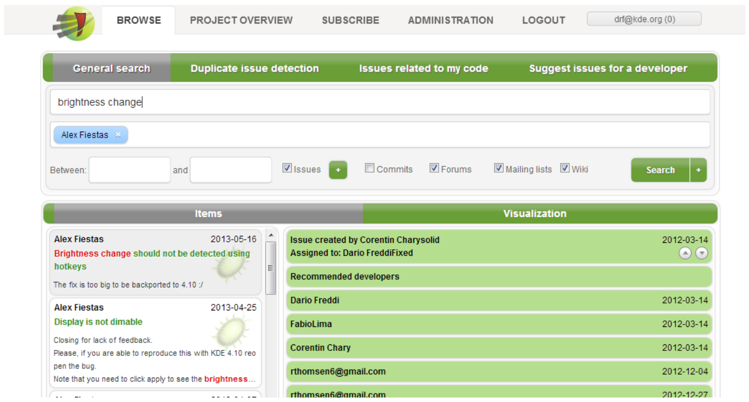
Since all information sources provide some user generated text we first send it to a text enrichment component that annotates the text using a custom built Annotation ontology. The ontology has 6,196 distinct concepts that contain 10,233 labels related to computer science. The details of ontology construction are described in [1]. After annotating the text we store the information from the sensors together with the annotations in a knowledge base.

After new data is stored, two components are notified about the change: the Recommendation service and the Event detector. The Recommendation service is responsible creating and maintaining an expertise profile for each user. The profile consists of code expertise (based on the methods created or modified by the user) and of topics expertise (based on the content the user is writing about). The Event detector is responsible for detecting if the new data matches any of the patterns that users provided as their interests. If it matches, then corresponding users are notified that new information relevant for them has been posted. Additional component of the ALERT system is also the search/visualization service which provides an interface for the user to query the knowledge base.

3 Core features of the ALERT system

Advanced search and visualization functionality

ALERT system provides an intuitive user interface (see Figure 1) where the user can find information by specifying a rich set of search conditions. The most commonly used is the keyword search. Using the Annotation ontology, the system is able to expand the search terms with their synonyms. In this way, searching for term "dialog" also returns posts containing terms "window" or "form". In the additional input box the user can specify constraints such as the author



brightness change

Alex Fiestas

[✓ Issues] Commits [✓ Forums] [✓ Mailing lists] [✓ Wiki]

Alex Fiestas 2013-05-16
Brightness change should not be detected using hotkeys
The fix is too big to be backported to 4.10 :/

Alex Fiestas 2013-04-25
Display is not dimable
Closing for lack of feedback.
Please, if you are able to reproduce this with KDE 4.10 reopen the bug.
Note that you need to click apply to see the brightness...

Issue created by Corentin Charysolid 2012-03-14
Assigned to: Dario FreddiFixed

Recommended developers

Dario Freddi	2012-03-14
FabioLima	2012-03-14
Corentin Chary	2012-03-14
rthomsen6@gmail.com	2012-12-04
rthomsen6@gmail.com	2012-12-27

Fig. 1. Screenshot of the search interface of ALERT system.

of the post, the product/component of the bug report, id of the bug report or the file/class/method name. The user can also uncheck individual sources if he wishes to ignore results from them and also set a specific time period of interest.

Search results are displayed in a list where each item contains the author, date and a short snippet of text. Selecting an item displays its full details on the right. Clicking an issue, for example, displays its available meta data, the description and all following comments. Clicking a commit, on the other hand, displays the tree of files, classes and methods modified in the commit. Results are also visualized and summarized using three visualizations based on the research done in [2]. The social graph shows a network of people involved in the results. There is an edge between two persons in the graph if one is responding to the others post (e.g., one person sends an email to the other). The timeline visualization shows the distribution of results over time. This allows one to quickly identify interesting patterns, such as very high/low activity. The last visualization is a tag cloud that provides a keyword summary of results.

Recommending open issues to developers

When bug reports are created, a stack trace with a detailed exception information is frequently provided. The trace contains names of the methods in the call path where the exception occurred. Since we are monitoring the source code developed in the project (using SCM sensors) we are able to automatically identify references of the known methods. By taking into account also the information about who changed individual methods, we can then suggest developers who can most likely fix a particular issue.

A developer can be recommended even in cases when no stack trace is provided. In such cases we can recommend developers based on the topics discussed in the bug report. For each developer we store a profile based on the content he wrote about. When we need a recommendation for a report we can suggest developers whose profile matches best with the content of the report.

Bug duplicate detection

A common problem on issue tracking systems is that users unknowingly create duplicates of the same issue. A bug triager needs to determine if a new report is describing a new issue or is a duplicate, before he can assign a developer to fix it. Since this is an error prone and time consuming task we wanted to make it easier. Given a bug report, the ALERT system can provide the user with a ranked list of other most content-wise similar bug reports. The similarity is computed using cosine similarity on bug reports represented as TF-IDF normalized vectors. Our experiments indicate that 60% of bug duplicates can be identified already by checking only the first 5 most similar reports.

Custom notifications

ALERT system allows the users to define custom patterns of interest and when they occur, the users are notified about it by an email or in a notification window inside the system. The user can, for example, choose to be notified when a post with X, Y or Z topic is published in some information channel, when more than X new bugs are reported in Y days, when the person X creates a new post, when a new bug is more than X% similar to an existing bug, etc. Using such notifications the user does not have to manually check for new relevant information – instead the content is automatically "pushed" to him.

4 Conclusion

In this paper we presented a system called ALERT whose goal is to help software developing communities. The system collects data from various information channels used by the community, extracts available information from it and stores it in a knowledge base. Using the stored data, the system can provide the users with advanced functionalities, such as semantic search and visualization over all information sources, bug recommendation, bug duplicate detection and custom notifications. The developed system was tested in three open source communities (KDE, Linagora OW2 and Morfeo project) that provided very positive feedback. The demo of the system containing KDE data is currently running at http://aidemo.ijs.si/alert/.

5 Acknowledgements

This work was supported by the Slovenian Research Agency and ALERT (ICT-249119-STREP).

References

1. G. Leban, L. Dali, and I. Novalija. Enabling semantic search in open source communities. In *Proceedings of ESWC 2012, Heraklion, Crete, Greece*, 2012.
2. L. Stopar and G. Leban. Searching for information in software development projects using ALERT system. In *Proceedings of the 15th International Multiconference on Information Society IS-2012*, 2012.