Combining Activity Metrics and Contribution Topics for Software Recommendations

Konstantinos Christidis, Fotis Paraskevopoulos, Dimitris Panagiotou, and Gregoris Mentzas
National Technical University of Athens
Athens, Greece
{kchrist, fotisp, dpana, gmentzas}@mail.ntua.gr

Abstract—In this paper we outline work in progress for the development of a recommender system for open source software development communities that takes into account information from multiple sources. Specifically our approach combines latent semantics of contributed information artifacts with quantitative metrics that indicate developer activity.

Keywords—Software Development; Recommender System; Semantic Analysis; Topic Models

I. INTRODUCTION

Open source developer communities today face a number of challenges. Developers and other contributors need to deal with a huge amount of constantly changing information, residing in multiple tools, and use it in order to make important decisions. The developers need to answer questions, such as “Who is the most competent developer to solve this issue?”, or, “Which issue can I solve?”. To support members of developer communities in answering these and similar questions, recommender systems for software development have emerged [1].

Existing approaches either quantitatively evaluate the developer activity in order to identify their expertise (e.g. [2]), or, provide some kind of content-based recommendation, based on analysis of the contributed content (e.g., [3]). Measuring the developer activity can provide an estimation of their overall expertise, however it cannot predict a developer’s ability to address a specific issue based on his/her area of expertise. Conversely, content similarity measures can only provide an estimation of how capable a developer is to address a specific issue, given his previous interests. Our approach aims to support developer activities considering both the area of expertise of the developer and the metrics that reflect the amount of involvement in the community.

In this paper we define a recommendation system for software developers that can merge different types of information from multiple sources. This recommender system combines the semantics of the information encountered in development artifacts with quantitative metrics linked to developer activity. Work presented here is currently in progress and will be extended and thoroughly evaluated in cooperation with three open source software development communities.

In the following sections, we explain the application of semantic analysis as well as the gathering of activity metrics, describe the model we have developed, outline its usage and discuss our future next steps.

II. BUILDING DEVELOPER PROFILES

In order to support developers and other contributors in dealing with information overload in open source development environments, a personal profile needs to be built. This profile can subsequently be used for personalised recommendation functionality. To create this profile, the recommender system can follow two distinct strategies: first, to semantically analyse the textual elements of the development process, and second, to extract quantitative measures of the developer activity.

A. Semantic Analysis

To perform semantic analysis, a number of unsupervised semantic analysis techniques have emerged that move beyond simple vector space models such as TF-IDF. Latent Semantic Analysis has been developed for correlating semantically related terms in a collection of text documents, while Probabilistic Latent Semantic Analysis (pLSA) incorporates a probabilistic foundation [4].

Latent Dirichlet Allocation (LDA) [5] has been proposed as an extension of pLSA that includes prior distributions on the generation of topics and words. LDA is based on the assumption that a generative model can sufficiently describe the document corpus and this model can be discovered: documents are produced by sampling topics from a distribution of topics over documents and sampling words from a distribution of words over each topic. The variables that represent the word-topic and topic-document distributions have a Dirichlet prior and need to be identified in order to fully describe the model. To this end, LDA is trained by analysing a number of text documents and distributions converge before the model can be used in applications. A number of additional techniques based on LDA have been proposed in order to better address time evolution, correlation between topics and hierarchies.

In this paper we take into consideration the fact that in modern software development a multitude of information exchange tools are being used. Mailing lists, source code management software, issue tracking systems and other collaborative platforms are being used in order to exchange information artifacts. Therefore, we propose applying semantic analysis in all different artifacts that contain textual information, such as revisions between different
versions of the source code, email messages, issue descriptions and issue comments and documentation.

B. Activity Metrics

Another approach for identifying developer expertise is to build profiles using quantitative information extracted from the developer activities during the software development process. These activities can be traced and measured from tools that are used for collaborative development. Each tool is considered a separate information source from which activity can be extracted in the form of quantitative metrics. Metrics are dependent on the information source which they are being extracted from. For example, in source code management systems, the expertise of the developer can be estimated from source code contributed [6] as well as the time of the commits [7]. In the case of communication tools such as mailing lists, expertise is determined from the communication history [8] [9]. Other indicative examples of using metrics for expertise identification can be found in [2], while for an extensive list of metrics see [10].

In this paper we propose the measurement of developer activities in the available collaboration tools of a community. Metrics that can be extracted include, but are not limited to, number of lines of code added, APIs introduced, documentation page changes, issues resolved, replies to a thread or mailing list and time since last activity. These metrics provide a quantitative estimation of the overall developer expertise.

III. RECOMMENDATION FRAMEWORK

We propose a multiple step recommender system that takes into account quantitative and qualitative information in order to build developer profiles (Fig. 1).

These developer profiles can in turn be used for recommendation activities. Two main recommendation activities that can take advantage of the proposed framework are the recommendation of bugs to assign to a given developer and the recommendation of developers to whom to assign a given bug. However, the proposed framework’s generality and the combination of multiple sources can allow for recommendations of other types of assignments to an open source community. These can include the recommendation of contributors for documentation requests or testing requests.

Step 1: Semantic analysis of information artifacts. In this step, all available information artifacts in the project are analysed using an unsupervised latent semantic analysis method. We propose the use of LDA and its variants in this step for the extraction of a stable common topic model. In the case of LDA, this step results in the approximation of two distributions: the distribution of topics over artifacts and the distribution of words over each topic.

Step 2: Calculation of per-artifact activity metrics for each developer. The activity of each developer is extracted from various available information sources. In this process, the proposed system keeps track of the specific artifacts that the developer acted upon: for example, the method to which lines of code were added, the wiki page that was updated, the comment that was submitted, etc. For every developer activity, the related metric is also saved. If we approach this calculation from a different standpoint, for each pair of developer and metric, all relevant information artifacts and a value describing the related activity is stored. This can be seen in (1) where the relationship between developer dev and metric m_i is described and m refers to the i_th metric that is calculated for all different artifacts a.

\[
\text{dev}(m_i) = (m_i(u, a_1), m_i(u, a_2), ..., m_i(u, a_N))
\]

For each developer, a matrix is calculated, where the rows correspond to the different metrics and the columns to the information artifacts (Fig 1.).

Step 3: Calculation of topic-based developer profile. In this step, we combine the results from step 1 and step 2 in order to build a topic-based developer profile, where the different metrics are aggregated over a common topic. In order to do so, we use the similarity between resources and topics that was calculated in step 1 together with an aggregation function. Results of steps 1 and 2 are consolidated in the form of a metrics vector that is a representation of the metrics that are calculated for a user and relate to a specific topic t. The aggregation function is not defined in this work, and is highly dependent on the impact that each metric has on the developer expertise. In (2) a developer profile is calculated by estimating developer expertise for each topic, where dev^D is the topic-based developer profile vector, f_ag the aggregation function, m the metrics vector and t the topics.

\[
\text{dev}^D = (f_{ag}(m(t_1)), f_{ag}(m(t_2)), ..., f_{ag}(m(t_K)))
\]

The resulting developer profiles provide a description of the developer expertise in the form of a K-dimensional vector.

![Figure 1. Recommendation Framework](image-url)
Step 4: Artifact and developer recommendation. The developer profiles generated in step 3 are used for generating recommendations. When a new issue arises, its semantic blueprint is extracted by topic inference using the existing topic model. Then, the most competent developer in the specific topic distribution can be found. In a different setting, when a developer looks for an issue to address, the recommendation system provides the issues that are more similar semantically to what he has been working on.

IV. RELATED WORK

A number of recommendation systems for software engineering have been presented that take into account the developer activity or the semantics of the contributions. In [11] the authors propose the usage of the terms found in the source code as a term vector for suggesting developers who have the proper expertise for handling a bug report. In [12] the writers propose a context-aware recommender system that assists engineers in switching artifacts based on the type of the development task and the interaction history. In [13], the authors propose a system that recommends a ranked list of developers to assist in performing software changes, using Latent Semantic Indexing. In [14], an LDA-based method is employed in order to analyse the Eclipse source code and provide insight on developer similarity, while in [15] the writers propose a variation of LDA that can model the evolution of topics in source code repositories.

A number of works indicate that topics can help to extract information from multiple sources in a project, as well as from multiple projects. The authors of [16] apply a topic modeling approach in order to establish a relation between source code and high level artifacts, such as requirements. In [17] the authors utilize topic models to demonstrate the relationship between the developers’ blogging behavior and their commits, while in [18] the proposed approach applies LDA across projects and together with a taxonomy can provide topic naming suggestions.

V. CONCLUSIONS AND FUTURE WORK

In this paper we have proposed a recommendation framework for open source development communities.

We are currently working on evaluating our model with different aggregation functions and topic extraction techniques. We also plan to evaluate the performance of the system in multiple recommendation activities, specifically a bi-directional recommendation setting between developers and issues, as well as for documentation and testing requests. To this end, we are collaborating with the KDE community and two smaller communities and have planned a validation experiment.

As the framework that we propose is generic and combines information from multiple sources, it remains to be seen how it compares to existing approaches that are focused on specific tasks, such as bug triaging. Additionally, we expect to examine the results of the topic modeling procedure for different types of artifacts and fine-tune the model as required.

REFERENCES


