Recommending Relevant Code Artifacts for Change Requests Using Multiple Predictors

Oliver Denninger
FZI Research Center for Information Technology
Karlsruhe, Germany

***Abstract*—Finding code artifacts affected by a given change request is a time-consuming process in large software systems. Various approaches have been proposed to automate this activity, e.g., based on information retrieval. The performance of a particular prediction approach often highly depends on attributes like coding style or writing style of change request. Thus, we propose to use multiple prediction approaches in combination with machine learning. First experiments show that machine learning is well suitable to weight different prediction approaches for individual software projects and hence improve prediction performance.**

***Keywords*-recommendation systems; software maintenance**

I. Introduction

It's widely recognized that maintenance dominates costs of software projects. Especially understanding the parts of a large software system relevant for a given change request is a time-consuming activity [1]. A recommendation system capable of recommending code artifacts relevant to a particular change request would improve developers productivity significantly. Most beneficially, when starting to work on a change request, the developer would be provided with a list of relevant code artifacts.

Various approaches have been proposed to tackle this problem – using different data sources and prediction techniques (see section IV). Summing up results in literature strongly suggest that the recommendation of relevant code artifacts for change requests is in principle possible. But none of the approaches proposed outperforms the others. Especially none performs constantly well on a broad variety of software projects. Thus, combining multiple prediction approaches seems obvious. However, using multiple predictors increases the number of prediction results and overwhelms developers. To crop the result set to a reasonable number, the results of different predictors need to be weighted. This problem should be well suited for machine learning.

The following sections describe the proposed recommendation approach using multiple predictors in combination with machine learning in detail, show first results along with a possible evaluation strategy, and list related work. At the end an outline on future work is given.

II. Recommendation Approach

In this work the term code artifact is restricted to methods and classes. Technically it could include files, packages, or even whole software components. Furthermore, to emphasize the use case of recommending code artifacts, the term predictor is used for approaches which are actually information retrieval (IR) approaches.

A. Preprocessing Data

For the proposed recommendation system, three different sources for textual attributes are available – source code, version control systems, and issue tracking systems (usually used to manage change requests). Source code contains identifiers and comments. Version control systems additionally contain revision descriptions (commonly called commit messages). Issue trackers record for each change request a title, a description and comments. How to preprocess repository data and establish links between code revisions and change requests has been shown in the literature [2].

B. Recommendation Process with Multiple Predictors

To bridge the gap between change requests (CRs), described in natural language, and code artifacts (CAs), multiple information retrieval based prediction approaches are used. Figure 1 shows the proposed recommendation process. Using the given data sources, three basic types of predictors are feasible. First, predicting former revisions based on text similarity between the CR and commit messages. Second, using text similarity of CRs to predict former CRs. For both types an additional step is needed to look up the corresponding CAs. Third, predicting CAs directly based on text similarity between the CR and both code identifiers and code comments.

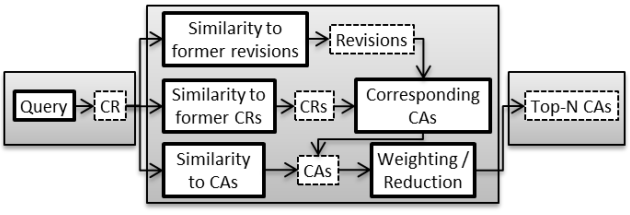

Figure 1. Recommendation process – activities marked with continuous boxes and entities marked with dashed boxes

The set of prediction results has to be reduced to a comparable small number of CAs, before presented to the developer. A reasonable result set should not contain more

than five to ten CAs. As all predictors are IR based, retrieved CAs are scored. But these scores are only comparable for a single predictor, not across different predictors. Thus, these scores cannot be used directly for a top-n reduction.

To make theses scores comparable, an individual weight needs to be assigned to each predictor. This is done using machine learning, as weights are expected to vary for different software projects. For each CA in the result set a feature vector is computed with the predictors as attributes. If a predictor has retrieved this particular CA then the according score is used as attribute value, otherwise the attribute value is set to zero. These, in general sparse, feature vectors are used along with the expected output – CA is relevant (value 1.0) or not relevant (0.0) – to train a neural network. This means, the neural network learns how to weight the scores produced by different predictors to make them comparable. After this weighting is performed, a simple top-n strategy is used to reduce the result set.

III. Evaluation and First Results

For each of the three prediction types presented in fig. 1, several predictors have been implemented using Apache Lucene as IR framework. The predictors are designed to handle Java code artifacts and vary in the way they preprocess texts, e.g., split camel-case identifiers, use stemming, or use latent semantic indexing (LSI). Experiments have been conducted so far using data from Eclipse JDT UI, JBoss AS, Tomcat 6 & 7, and ArgoUML.

The first experiment compared the results of the different predictors (without performing weighting and reduction) by counting unique relevant CAs. They are relevant to a query and retrieved only by one of the predictors. The results showed that the predictors, retrieving a large number of unique relevant CAs vary between projects, e.g., a predictor using part-of-speech (POS) analysis to index nouns only, performed well for Eclipse JDT UI, while a predictor only indexing camel-case terms performed well for ArgoUML.

The second experiment used the encog framework to train a neural network for weighting the results of different predictors. This approach was compared against a simple ranking approach based on scores – ignoring that these scores are not comparable across different predictors – by calculating the presicion@5 values. The learning based approach improved the precision@5 performance by up to 20%.

IV. Related Work

The Jimpa approach [3] indexes texts from both change requests and revisions and uses a probabilistic IR model for recommendations based on text similarity. In contrast, the Suade approach [4] performs static analysis on source code to recommend additional CAs related to a given set of CAs. The SNIAFL approach [5] establishes links between feature descriptions and CAs (represented by method names) based on text similarity. The ROSE approach [6] uses a technique called association rule mining (ARM) to derive rules from version history like: if CA "A" has been changed, then CA "B" has been changed too. Gethers et al. [7] use LSI to index identifiers and comments from source code for recommendations. To improve prediction performance they combine it with ARM and dynamic analysis if available. Ahsan et al. [8] use a multi label classification approach for recommendations, labeling CRs with the CAs changed. The DebugAdvisor approach [9] uses text similarity to recommend related bug reports or debugger logs. The design supports multiple preprocessors for indexing, based on the data source used, but not multiple predictors.

V. Conclusion and Future Work

The recommendation approach presented using multiple predictors in combination with machine learning is promising. First experiments show that multiple predictors are capable of outperforming a single predictor. Furthermore, using machine learning to weight results from different predictors allows to reduce the results set to a reasonable number. The question of whether it is sufficient to learn weights once per project or they have to be adapted while the project evolves, has not been studied so far. An additional improvement could be the use of context creating techniques. For example association rule mining or static analysis could predict additional CAs related to the ones predicted so far.

References

[1] A. J. Ko, B. A. Myers, M. J. Coblenz and H. H. Aung, *An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks*, in IEEE Transactions on Software Engineering, vol. 32, 2006.

[2] M. Fischer, M. Pinzger and H. Gall, *Populating a release history database from version control and bug tracking systems*, Proc. Int. Conference on Software Maintenance, 2003.

[3] G. Canfora and L. Cerulo, *Fine grained indexing of software repositories to support impact analysis*, Proc. Int. Workshop on Mining Software Repositories, 2006.

[4] F. Weigand-Warr and M. P. Robillard, *Suade: topology-based searches for software investigation*, Proc. Int. Conference on Software Engineering, 2007.

[5] W. Zhao, L. Zhang, Y. Liu, J. Sun and F. Yang, *SNIAFL: towards a static non-interactive approach to feature location*, Proc. Int. Conference on Software Engineering, 2004.

[6] T. Zimmermann, P. Weigerber, S. Diehl and A. Zeller, *Mining version histories to guide software changes*, Proc. Int. Conference on Software Engineering, 2004.

[7] M. Gethers, H. Kagdi, B. Dit and D. Poshyvanyk, *An adaptive approach to impact analysis from change requests to source code*, Proc. Conf. on Automated Software Engineering, 2011

[8] S. N. Ahsan and F. Wotawa, *Impact analysis of SCRs using single and multi-label machine learning classification*, Proc. Int. Symposium on Empirical Software Engineering and Measurement, 2010.

[9] B. Ashok, J. Joy, H. Liang, S. Rajamani, G. Srinivasa and V. Vangala, *DebugAdvisor: a recommender system for debugging*, Proc. ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2009.