Towards a Collaborative Code Review Plugin

Vera Barstad, Vladimir Shulgin, Morten Goodwin, Terje Gjøsæter

Faculty of Engineering and Science, University of Agder Serviceboks 509, NO-4898 Grimstad, Norway {verab12, volods12}@student.uia.no {morten.goodwin, terje.gjosater}@uia.no}

Abstract

This paper investigates the feasibility of a plugin facilitating identification of code quality. The proposed solution includes a distributed plugin that performs static code analysis as well as machine learning based classification, and is able to identify well written and badly written code.

Introduction

Many collaborative tools for peer and machine code analysis exists. Most of them are complicated to use and do not support both automated and manual code analysis. This paper presents a plugin for Eclipse with the intention of simplifying source code rating. This is done based on a state-of-the-art analysis of benefits and disadvantages with existing plugins. The proposed plugin, developed in collaboration between academia and the industry, aims at a general purpose code quality tool for software development companies and institutions.

Since code review came into focus there have been published several papers about peer and automated code review [7]. Automated code analysis/code review and efficiency of this technology still remains a topic of intense research after more than thirty years [3, 9, 6]. Some work has already been done in the direction of rating code and code review, there exist several plugins aimed on providing a way of rating the code [10, 9, 4, 1]. Most of these plugins are useful, but these plugins are complicated and mostly focused either on peer rating or automated rating. Similarly papers have been published about collaborative tools [5, 4, 2], and research of peer code review have been done in different areas [8, 11].

We are investigating to what extent a tool that facilitates identification of well written and badly written source code improves the code quality. The initial part of this project, and the core part of this paper, is to implement a proof of concept for this investigation. The tool should use both peer review/human rating, static code analysis and automated machine learning code review. It should be possible to compare the classification from different methods to examine which have the most impact, if any.

Design

To investigate the possibility of code analysis, we have proposed a solution for a plugin that facilitates identification of well written and badly written code. We focus on a simple implementation of automated code analysis, the plugin should use both automated and peer code review.

The users of the plugin will open a source file that is submitted for review, select the code that s/he wants to review, and push a button or select a menu to review. We suggest a framework that is extendable with new methods, both static and classification methods. Requirements for the plugin was created in cooperation with the industry.

This paper was presented at the NIK 2013 conference. For more information see http://www.nik.no/

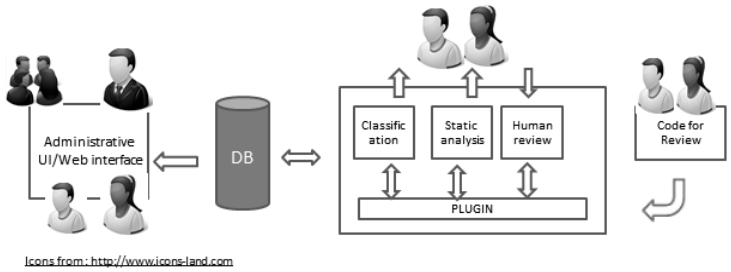

Figure 1: Architectural overview

Developers use the plugin to send code for review, rate code, and get feedback from the static analysis. It is possible for administrators, groups of developers or the individual developer to use an administrative or web interface to get more detailed information and reports from the system. A reviewer on the other hand, selects the code s/he wants to review, and pushes a button or selects a menu item to give it a review. S/he should be able to select the type of review and add a comment. When a file is opened for review, all previous reviews are shown. It is possible to add a new review to a part of the file that already has previous reviews.

The code reviews from the human rating is statically analyzed. Several features/parameters are extracted and used as training data for the Naïve Bayes classification using parameters and maximum likelihood. This facilitates classification of new source code. The results from the static analysis is stored, and the user gets a visible summary of the analysis result, and a notification whether his/hers code does not follow the defined rules. The information will help the users improve the quality of the source code. Naïve Bayes classification will give a possibility to find out which of the features influence mostly on the reported "likes" or "dislikes".

The plugin has the possibility to compare different methods in the editor. “The Leave one Out” method can be used to compare the quality of one classification method against another classification method, or compare review done by human rating against automated result from the classification. This facilitates identification of which methods that are most accurate if several results are compared.

Discussion

In this project we developed a plugin for Eclipse and Java, that supports code review. We have shown that it is possible to implement a plugin, that includes peer code review, static analyzer and automatic classification of code.

Research about source code analysis (SCA) has shown that SCA, bot related to automated and peer review, has a positively influence on the quality of code. Using collaborative tools, stimulates developers to think more about their code and its quality. Our tool, which provides all the basic features of peer code review, makes developers pay more attention to their code. The way of evaluation of code quality for peer review is based on their “likes” and comments. While “likes” directly show that code is considered to be well or badly written, comments provide additional information. This approach is used by many collaborative tools and many developers web tools, including online resource for providing help with software development [1]. Such systems have shown in practice that rating of comments is useful and might help to distinguish between well written and badly written code. In our project we have implemented a

[1] http://stackoverflow.com

system for ranking source code. Moreover, our main goal was to implement a plugin, which will provide peer review with a possibility to give likes and leave comments.

When we speak about identification of well written and badly written code done by static code analysis, we also focus on static tools. We have shown a simple implementation of static analysis, as, for example, examining if the variables are Camel Cased and start with a lowercase. According to official documentation from Oracle[2], using these simple rules for naming improves code quality and make it more user-friendly and reusable. Our motivation was to show that a static tool can be used for improving the code quality and that it could be used in a pair with peer review. Sufficient data will be obtained during the following project. Code analysis is evolving and there comes up new efficient ways of evaluation of the source code, one of them is the application of algorithms and techniques from areas such as Machine Learning and Data Mining. In this project, we are interested in whether an approach like Naïve Bayes classification can be used for code analysis. We chose this approach, because it is relatively easy to use in comparison with other classification methods, but still gives a reliable result. Our results were based on synthetic data, which was created for this purpose. As attributes we used features from the static analysis. The result of applying this classification we predict if the user will “like” or “dislike” the code. If one of the attributes is more important for that user, it will have more influence on the tuple of Naïve Bayes classification, and thus on the overall estimation.

We have implemented a plugin with possibility for peer review. Moreover, we followed some of the 11 best practices for code review [9], sending smaller portions of the code for review should improve the effectiveness of code review, and for this reason, our program is limited to reviewing one method at a time. Another recommendation was that: "Managers must foster a good code review culture in which finding defects is viewed positively". Another best practice is that our plugin was done in a lightweight-style. Lightweight-style code review is efficient, practical, and effective at finding bugs. We have proposed a possible solution for static code analysis. And base the static code analyzer on different patterns that can be found in the source code. Providing a model for static code analysis is a good starting point for further research. We have provided the theoretical background for implementation of both static code analysis and classification of the code. To implement classification in our project, we will use patterns from static analysis and “likes” from peer review as a training data. The solution has been successfully tested using synthetic data.

Conclusion

We have proposed a framework for identification of well written and badly written source code performed by a combination of human reviews, static analysis and classification methods. We have implemented an open source Eclipse plug-in that facilitates identification of well written source code done by peer review and also proposed models for the implementation of static code analysis and Naive Bayes classification. We have shown that Naïve Bayes classification can be used in the purposes of predicting "good" and "bad" quality of code. Our initial analysis has shown that source code analysis and collaborative tools are topical and efficient instruments for software developers. Better understanding of SCA tools might help to improve user’s code and increase his/her knowledge, which will lead to better project outcome. Contributions from this work include: A plug-in that facilitates identification of well written code. Design for a framework that use static analysis and classification methods.

[2] http://www.oracle.com/technetwork/java/codeconv-138413.htm

Algorithms and proposed solution for using static analysis together with Naïve Bayes to automatically identify well written and badly written source-code.

Future Work

This project is part of an ongoing research project. In the next states we plan to extend the Proof of Concept to a fully functional plugin. Open issues include answering questions such as: - Is the quality of the source code in projects improved if this tool is used in a social setting? - Is it possible to use the tool to identify code quality difference in different projects. - Does the plugin have a positive impact on the code quality, in line with [8, 9, 5]. We plan to investigate additional machine learning approaches for code quality. Further, we will do empirical analysis in collaboration with the industry through social networks.

References

[1] Jupiter eclipse plugin. [Online]. Available: https://code.google.com/p/jupiter-eclipse-plugin/ Accessed : 2013-09-30

[2] G. Booch and Brown A. Collaborative development environments. Advances in Computers, 59, 2003.

[3] D. Binkley. Source code analysis: A road map. Future of Software Engineering FOSE, 7:104–119, 2007.

[4] L-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up eclipse with ollaborative tools. In Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, pages 45–49, 2003.

[5] M. Coccoli, L. Stanganelli, and P. Maresca. Computer supported collaborative learning in software engineering. In Global Engineering Education Conference (EDUCON), 2011 IEEE, pages 990–995, 2011.

[6] Forrester Consulting. The value and importance of code reviews: It's time to exploit modern technology for improved code reviews. 2010.

[7] Sarah Heckman and Laurie Williams. A systematic literature review of actionable alert identification techniques for automated static code analysis. Information and Software Technology, 53(4):363 – 387, 2011.

[8] Ken Reily, Pam Ludford Finnerty, and Loren Terveen. Two peers are better than one: aggregating peer reviews for computing assignments is surprisingly accurate. In Proceedings of the ACM 2009 international conference on Supporting group work, pages 115-124, 2009.

[9] SmartBear Software. 11 best practices for peer code review. 2012. [Online] Available: http://www2.smartbear.com/Best-Practices-Peer-Code-Review.html Accessed : 2013-09-30

[10] Mason Tang. Caesar: A social code review tool for programming education. 2010. [Online] Available: http://groups.csail.mit.edu/uid/other-pubs/masont-thesis.pdf Accessed : 2013-09-30

[11] D. A. Trytten. A design for team peer code review. In Proceedings of the 36th SIGCSE technical symposium on Computer science education, volume 37, pages 455–459, 2005.