Recommendations for Developers to Prevent Errors in Source Code

Bálint CSÓKA*

Slovak University of Technology in Bratislava
Faculty of Informatics and Information Technologies
Ilkovičova 2, 842 16 Bratislava, Slovakia
xcsoka@stuba.sk

These days, the number of developers working on projects from open code repositories are growing. With continuous tracking of the created source code is possible to create a model of methods and classes, from which we can determine, which fragments of code were used in the current build, which are the most used methods etc., that can be helpful for the developer to achieve his goal. When developing a large-scale software system, maintaining existing code and fixing errors may take more time than the development of new functions. It is possible to recommend activities based on static source code analysis, helping developers write less error-susceptible code by using the already existing (and tested) methods for the same problem domain. Static source code analysis is a type of analysis, where source code is inspected without executing the program.

In the corresponding thesis, a partially source-code based recommendation system (SCoReS) is designed and developed to endorse the developer in writing error-free code and give advice on the recommended methods to use [1]. Since the paper is not a pioneer in this area, the solution is finally compared to the already existing solutions in this problem domain.

Our goal is to analyse the existing methods of processing and collecting data from open repositories, explore ways of tracking the developer to effectively anticipate the error-susceptible parts of code based on static source code analysis. The aim was to create a plugin that integrates into the development environment and warns/recommends based on the current activities of the user. Another important aspect to implement was to be able to recommend existing (tested) solutions to the developer, such as similar methods, based on the footprint of the method to avoid rewriting the same functionality by different developers in large software projects. We focus the development of this module in the Eclipse[1] development environment, because it is open-source, available at no cost, and supports multiple platforms. These

* Supervisor: Eduard Kuric, Institute of Informatics and Software Engineering
[1] http://eclipse.org

Spring 2015 PeWe Workshop, April 11, 2015, pp. 11-12.

characteristics imply a larger user base for the module. The method recommender module will be able to work with Java language due to its popularity.

The discussed behaviour can be achieved by generating a dataset of available methods from the source code and using on-the-fly analysis of the developer's latest text input. The operators, variables and entities in the source code are converted to tokens to construct the meta-layer and enable abstraction from variable naming, loop construction and position of fragments. The solution evaluates and updates the model of the currently edited code artefact while the developer is typing and compares the artefact in question to the existing ones in the dataset to find a similar, which can be inserted immediately without re-typing the desired functionality. The functionality and effectiveness of the module will be compared to the existing solutions. As an alternative and comparison base, we selected modules FrUiT and CodeBroker because of their ability to recommend similar functionality in terms of methods and classes and RASCAL for the activity analysis of the developer.

The most important part of the recommender is the software clone detection package. It allows the program to compare two signatures of a code segment to detect similarity [3]. In our case, similarity (or partial similarity) means that the currently edited method implements the same or analogous functionality as an already existing one. This behaviour is needed to determine which functions need to be recommended to the developer. The developer can then decide whether to use the recommended method based on the original intent. Since this kind of recommendation would be useless, if the developer already finished the construction of the method, we need to identify similarity at the earliest point. To achieve this, the program maintains a dataset of existing and already processed methods in its memory, and creates the current method's model at the time of insertion dynamically. Then, the footprint of the method is compared to the existing footprints in the dataset by comparing sub-sequences of entity tokens. After a threshold is met by this comparison, the method is recommended to the developer in the form of a visual popup. Thus, the successful recommendation of similar methods helps the developer achieve a more error-susceptible code as a result, by preventing redefinition and encouraging the use of tested methods.

Acknowledgement. This contribution is the partial result of the Research & Development Operational Programme for the project Research of methods for acquisition, analysis and personalized conveying of information and knowledge, ITMS 26240220039, co-founded by the ERDF.

References

[1] Robillard, M., Walker, R., Zimmermann, T. Recommendation Systems for Software Engineering. 2010. ISBN 9783642451348

[2] Jannach, D., Zanker, M., Felfernig, A., Friedrich, G. Recommender Systems: An Introduction. Cambridge: Cambridge University Press, 2010. ISBN 9780521493369

[3] Roy, CH. K, Cordy, J. R. A Survey on Software Clone Detection Research. Technical Report 541, Queen's School of Computing. 2007.