Analyzing Programming Behavior to Support Self-Reflection for Improving Source Code Quality

Christian R. Prause, Maren Scheffel, Katja Niemann, Martin Wolpers
Fraunhofer Institute for Applied Information Technology FIT, Germany
{christian.prause, maren.scheffel, katja.niemann, martin.wolpers}@fit.fraunhofer.de

Abstract. Improving the quality of source code is an important aspect of reducing software development cost in industry. This position paper postulates that developers in a software team conduct different activities during software development – like closing the source code editor for compilation as opposed to leaving it open – that either improve or degrade the quality of resulting source code. We propose to combine mining of contextualized attention metadata with developers' quality reputations to identify activities that seem to be inhibitors of quality. By analyzing the behavior of an entire team, adverse activities can be detected automatically, and can be reported back to developers so that they can learn to avoid adverse behavior.

Introduction

Internal quality describes those characteristics of a software product that are only perceivable to the developing organization (hence internal) and not the customer (external). Improving the internal quality of software is an important way of reducing development costs (ISO-9126, 2001). A notable artifact in software projects that is only visible to the developing organization is source code. This is due to software development being highly collaborative work that involves much communication and coordination, and source code has become a medium of

communication between humans (Dubochet, 2009). Consequently, there is much interest in the software industry to improve the quality of software source code.

Researchers and practitioners have attempted different ways of how the quality of source code could be improved including reviews, pair-programming, static analysis, or reputation based approaches (e.g. Wray, 2009; Moha et al., 2010; Prause & Apelt, 2008).

With this paper we want to open up a new view on how source code quality could be improved. We propose to identify activities in a developer's programming behavior – i.e. what and how a developer uses his different software tools like compiler, editor, or browser while programming – that seem to lead to either better or worse code quality as evidenced by the developer's average code quality. Once such activities are identified, they can be made available to the rest of the team to allow collaborative learning from the other developers' behavior.

Technological Background

This section explains the technological foundations of our concept. We want to mine into developer activities trying to identify successful behaviors. What is needed for this is a framework that provides activity data and mining capabilities, and a method for identifying what *successful* means.

Contextualized attention metadata

The Contextualized Attention Metadata (CAM) schema models a user's interaction with digital contents across system boundaries (Schmitz et al., 2011). CAM records of a user can describe his entire computer usage behavior and not merely the foci of attention. Due to its event centeredness CAM is suited for evaluating and analyzing user observations. All user actions are described by an event – defined by id, name and timestamp – that is connected to information about the session it takes place in and the entities it involves (e.g. the user, documents, application, etc.). CAM analyses thus provide an overview about when and where user actions take place and among others enable the discovery of popularity, usage bursts and trends of tools.

Computing developer reputations

According to the Oxford Dictionary[1], reputation is what is generally said or believed about the abilities or qualities of somebody or something. A reputation system is software that determines a means of a user's reputation from his

[1] http://www.oed.com/

observable actions or work results. Reputation systems are often used in Web 2.0 societies to promote well-behaving and trust (Jøsang et al., 2005).

The overall idea of reputation is this: A developer who contributes a lot of high-quality source code to the team's revision control system (e.g. Subversion[2]) gets a higher reputation than a developer who contributes less quality code.

A viable definition of source code quality is through code smells, or more precisely, the absence of code smells. Fowler (2001) coined the term *code smells* to describe features in source code that make it more difficult to maintain. Such code can therefore be considered as being of lesser quality. To a certain degree, it is possible to identify code smells using static analysis. Static analysis toolkits examine source code in the absence of input data and without running the code (Ayewah et al., 2008). One such freely available toolkit for Java is Checkstyle[3]. By counting the rule violations within all files of source code, we derive a notion of quality for each file.

Next, we obtain authorship information for each file by looking at its evolution history. Subversion, for example, provides the "svn blame" command to identify the author of each line of source code. We compute a developer's reputation by averaging the quality of all lines of source code that were contributed by him.

Learning from mining CAM

We have previously analyzed user activities and detected usage patterns in the context of a programming course at a university (Scheffel et al., 2011). CAM was used to represent the multiple events occurring in a learning environment. Once the events had been captured, techniques were applied to first extract key actions from the collected data, and then to identify usage patterns, specifically after receiving error messages. The procedure was validated by applying it to an undergraduate engineering course on introductory programming in C. The teaching staff identified some errors as most problematic that should be discussed in more detail in the course.

Taking this idea further, analysis of collected CAM can be used to support developers during their software production process. Results from a reputation system can be juxtaposed to those from CAM analyses. For example, the time of day might play a role in the quality of the code (and thus a developer's reputation): all source code written at night could be good while most code produced around noon or directly after the lunch break is not (or vice versa). A relation between other tool usage and the code quality can also be examined, i.e. did a developer really focus on his work or was he distracted by other tasks (whether they are work related or not). As source code production is often a

[2] http://subversion.apache.org/

[3] http://checkstyle.sourceforge.net/

collaborative work, CAM from several developers can be taken into account: Mary and Peter always produce high quality code and looking at their CAM shows that they often chat with one another while coding. Peter and Paul however usually produce quite low quality code when collaborating. Looking at their CAM shows that they never communicate during collaborative coding. A correlation seems likely. It is also possible to identify general action/working patterns of those developers with high reputation and offer this information to those with low reputation so they can learn from them.

Conclusion

In this paper we proposed to combine results of contextualized attention metadata analyses with those of quality reputation systems for developers to identify activities that seem to be inhibitors of source code quality. We are currently planning to put this idea into action. The monitoring tools to collect CAM events have already been implemented and work is now focused on a self-reflection tool. First tests will be run with a small group of researchers to evaluate if the quality of the code and the reputation can be improved.

Acknowledgments

This research was supported by the BRIDGE (project number 261817) EU project.

References

Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J. & Pugh, W. (2008): Using Static Analysis to Find Bugs. *IEEE Software, IEEE Computer Society, 2008, 25*, pp. 22-29.

Dubochet, G. (2009): Computer Code as a Medium for Human Communication: Are Programming Languages Improving?, *21st Annual Workshop of the Psychology of Programming Interest Group*.

Fowler, M. (2001): *Refactoring: Improving the Design of Existing Code,* Addison-Wesley.

ISO/IEC 9126-1 (2001): Software engineering - Product quality: Part 1: Quality model, *ISO (International Organization for Standardization)*.

Jøsang, A., Ismail, R. & Boyd, C. (2005): 'A survey of trust and reputation systems for online service provision'. *Decision Support Systems*, vol. 43, no. 2, pp. 618–644.

Moha, N., Guéhéneuc, Y.-G., Duchien, L. & Meur, A.-F. L. (2010): DECOR: A Method for the Specification and Detection of Code and Design Smells. I*EEE Transactions on Software Engineering,* 36, pp. 20-36.

Prause, C. R. & Apelt, S. (2008): An Approach for Continuous Inspection of Source Code. *Sixth International Workshop on Software quality, ACM.*

Scheffel, M., Niemann, K., Pardo, A., Leony, D., Friedrich, M., Schmidt, K., Wolpers, M. & Delgado Kloos, C. (2011): 'Usage Pattern Recognition in Student Activities'. In: Delgado Kloos et al. (eds.): *ECTEL-2011*. LNCS, To be published, Springer Heidelberg.

Schmitz, H.-C., Wolpers, M., Kirschenmann, U. & Niemann, K. (2011): 'Contextualized Attention Metadata'. In: Roda, C. (ed.): *Human Attention in Digital Environments*. Cambridge University Press, pp. 186-209.

Wray, S. (2009): How Pair Programming Really Works. *IEEE Software, IEEE Computer Society,* 27, pp. 50-55.