Identifying Hidden Source Code Dependencies from Developer's Activity

Martin KONÔPKA*

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

Evaluation of software projects is important part of the process when reaching desired goals in planned time schedule is of high interest. Delivering quality software products in time requires to monitor them, evaluate their attributes in order to manage and support the development process. Traditionally, we evaluate software from the source code perspective as a result of its development process. With numerous metrics we are able to quantify the source code, evaluate its quality, complexity or maintainability [2].

Software source code is the result of development process, i.e. developer's activity. Developer performs various activities during the development which we can monitor [1]. Main focus is on developer's activities directly associated with development, i.e., programming, modelling or checking-in work results. Secondly, the work results, source code contents and time-based information about developer's work depends on activities associated with development indirectly, i.e., searching for solutions, studying existing source code base and documentation. In the end, these activities are interweaved with activities which are not associated with development but still has high impact, i.e., emotional state of developer, environmental context, etc.

Traditional source code metrics do not rely on the mentioned types of developer's activities. However, we can search for connections of activities with resulting attributes of software to identify hidden patterns and relationships in the development process as well as to gather more information about it. One such application is to compare the metrics and techniques based solely on syntactic analysis of source code with approaches based on evaluation of monitored activities.

In our work we focus on identification of hidden source code dependencies from developer's activity to enhance existing source code dependency graph. Source code dependencies represents connections of software components, e.g., call references, inheritance or containment. Developer uses dependency graph to find out impact of possible change in one software component on others. Based on the source for identification of dependencies, we distinguish between:

* Supervisor: Mária Bieliková, Institute of Informatics and Software Engineering

Spring 2014 PeWe Workshop, March 21, 2014, pp. 67–68.

- Explicit dependencies – identified with syntactic analysis of source code,
- Implicit dependencies – identified from developer's activity.

Implicit dependencies describe which components are related to each other, even when no explicit dependency exists between them at all. Implicit dependencies are identified from developer's activity of interactions with the source code during the development, e.g., when developer was studying existing solution in the source code, copied code fragment from one component into another. Interactions with source code also describe which components relate to each other in context of the particular task which developer was working on. We identify implicit dependencies from these activities with source code files:

- Time-based activities – open, close and switch to source code file.
- Copy-paste code fragment between two source code files.

Implicit dependencies overlap explicit ones but also introduce new connections to the existing dependency graph. Developer searching for dependent components may then find the components which are loosely coupled in the source code, but highly coupled during the runtime or development.

Our work is part of research project PerConIK – Personalized Conveying of Information and Knowledge (http://perconik.fiit.stuba.sk) [1]. We use services and logs provided by this project for evaluation of our method. We also see its possible application in code review and identification of problematic places when code reviewer searches for places developer worked on. If reviewer identifies problem in one place developer worked on, there may be problems in other places she touched as well. Surely, this is possible to extract from the list of checked-in files, however it is important to prioritise this list.

Extended version was published in Proc. of the 10th Student Research Conference in Informatics and Information Technologies (IIT.SRC 2014), STU Bratislava, 474-479.

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-funded by the ERDF.

References

[1] Bieliková, M., Návrat, P., Chudá, D., et al.: Webification of Software Development: General Outline and the Case of Enterprise Application Development. In: *AWERProcedia Information Technology & Computer Science: 3rd World Conf. on Information Technology*, Barcelona, (2012), pp. 1157-1162.

[2] Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. 2nd Edition, PWS Pub. Co., Boston, MA, USA, (1998).