Webification of Software Development: User Feedback for Developer's Modeling

Eduard Kuric and Mária Bieliková

Faculty of Informatics and Information Technologies,
Slovak University of Technology, Ilkovičova 2, 842 16 Bratislava 4, Slovakia
name.surname@stuba.sk

Abstract. In this paper we present an approach to leveraging experience from rapidly evolving field of information processing on the Web for software development. We consider a web of software artifacts (components) as an information space. Supporting any task in such environment of interconnected artifacts depends on our knowledge on user preferences and his characteristics. We envision the concept of collaborative software development to improve software quality and development efficiency by using both implicit and explicit user (software developer) feedback. It opens a space for using approaches originally devised for the Web. The core of our approach is based on our developed platform for independent code monitoring where we create a dataset of developers' implicit and explicit feedback based on monitoring developers' behavior. Employing this platform we acquire, generate and process descriptive metadata that indirectly refer source code artifacts, project documentations and developers activities via document models and user models. As an example of our concept we present an approach for estimation of student's expertise in a programming course.

Keywords: webification of software development, implicit/explicit feedback, interaction information, user modeling, monitoring user behavior.

1 Developer's Feedback in a Web of Software Artifacts

Developers often use a web of software artifacts as a giant repository of source code, which can be utilized for solving their software development tasks. Supporting any task in such environment of interconnected artifacts depends on our knowledge on user preferences and his characteristics. Relevance user feedback is typically used for user profiling during long/short-term modeling of user's interests and preferences on the Web. Relevance feedback techniques have been used to retrieve, filter and recommend a variety of items [4]. Our aim is to support software development by using both implicit and explicit user (software developer) feedback, which creates rich interconnections between software artifacts. This includes not only the shift towards the use of web-based resources in software processes, but also and more importantly it opens a space for using approaches originally devised for the Web (as a network of interconnected content) to support the software development process.

S. Casteleyn, G. Rossi, and M. Winckler (Eds.): ICWE 2014, LNCS 8541, pp. 550–553, 2014.

Our work is a part of a research project called PerConIK[1] (Personalized Conveying of Information and Knowledge). We cooperate with a medium size software company. We focus on support of applications development by viewing a software system as a web of information artifacts. Our aim is devising the right metrics to evaluate software artifacts and to identify particular problems and recommending corrective actions. The core of our approach is based on our developed platform for independent code monitoring [1]. We developed several agents that collect and process documentations, source code repositories, developers' activities, etc. We create within the project a dataset of developers' implicit and explicit feedback based on monitoring behavior of developers for the purpose of estimating developers' expertise. For example, we exploit implicit feedback such as searching relevant information on the Web and searching in source code during performing tasks, writing and correcting source code in development environment, and explicit feedback such as peer review (review feedback attached to source code).

To the present, we have focused mainly on modeling developer's expertise in software house environment. It is based on investigation of software artifacts which the developer creates and the way how the artifacts were created. In other words, we take into account the developer's source code contributions, their complexity and how the contributions were created to a software artifact (e.g. copy/paste actions from external resources, such as a web browser); the developer's know-how persistence about a software artifact; and technological know-how - the level of how the developer knows the used libraries, i.e., broadly/effectively. All on daily basis of software development.

In a software company estimation of developers' expertise allows, for example, on the one hand, managers and team leaders to look for specialists with desired abilities, form working teams or compare candidates for certain positions, on the other hand, developers can locate an expert in a particular library or a part of a software system (someone who knows a component or an application interface) [2,5]. It can be also used to support so-called "search-driven development". When a developer reuses a software artifact from an external source he has to trust the work of an external developer who is unknown to him. If a target developer would easily see that a developer with a good level of expertise has participated in writing the software artifact, then the target developer will be more likely to think about reusing.

On the contrary of a software company, where software is created by professionals, in academic environment, students learn how to design and develop software. Moreover, a student produces significantly less data (implicit/explicit feedback). Our goal is to provide a tool that allows a teacher to evaluate student's knowledge and skills (expertise) based on monitoring student's behavior during developing tasks and adaptation of instruments developed for the general approach. As an example of our approach we present an approach for estimation of student's expertise in a programming course. It allows, for example, teacher to adapt and modify his teaching practices.

[1] PerConIK: `http://perconik.fiit.stuba.sk/`

2 Estimation of Student's Programming Expertise

In our approach modeling student's expertise is based on estimation of a degree of student's expertise of a concept in comparing with other investigated students. In other words, if students solve a task focused on acquiring skills of particular concept, e.g., *priority queue*, then by analyzing their resultant source code, interaction data and by using appropriate software metrics, we can estimate levels of students' expertise of the concept. By using the particular students' expertise estimations of concepts we are able to estimate a degree of student's expertise for the whole course and compare the estimations among the investigated students based on the same evaluation criteria.

We experimented with data gathered during bachelor course on *Data structures and algorithms*. During seminars the students solve programming tasks. Each week is focused on training and acquiring skills of a concept such as *stack, binary tree, hash table, etc.* The students solve the tasks in a learning system *Peoplia*[2]. Students can select to solve a simpler or more complex task focused on acquiring skills of a concept. Students get points for their successful solutions. In autumn semester 2013/14, 251 students enrolled in the course.

When a student submits a solution of a task to *Peoplia*, its correctness and efficiency (time complexity) is evaluated. The solution is accepted if it is correct and efficiency tests are successful. The student has unlimited number of submission attempts and the solutions are checked by a plagiarism detection system. Estimation of a degree of student's expertise of a concept c based on a student's correct solution l for a programming task t is calculated as follows:

$$Exp_c(s,t,l) = CX(t) * EF(s,l) * \frac{1}{log_2(1+CT(s,t))}, \tag{1}$$

where $CX(t)$ is complexity of the task t estimated based on a combination of Logical Source Lines of Code (SLOC-L) and McCabe VG complexity metrics. For calculation of SLOC-L we have adopted the definition from the CodeCount[3]. $EF(s,l)$ returns 1.5 if the student's s submitted solution l is effective, otherwise 1. A solution is effective if its execution time is less than or is equal to a median value of all execution times of submitted correct solutions for t (it is based on preliminary experiments). $CT(s,t)$ is a number of submitted solutions by the student s for the task t (the last solution was accepted by Peoplia). The estimation of a degree of student's expertise of the course is calculated as $\sum_i Exp_{c_i}(s,t_i,l_i)$.

We estimated expertise for all students and compared our results to results achieved on exam. 78 out of 251 students were not allowed to take the final exam because they did not achieve the qualification criteria. Both values (estimated expertise and points of the final exam) were normalized into the interval $[0,100]$. To each student a pair (X,Y) is assigned, where $X \in [0,100]$ is a number of points of the final exam and $Y \in [0,100]$ is the estimated student's expertise. Subsequently, the values in each pair were mapped as follows: $A - [92,100]$, $B - [83,91]$, $C - [74,82]$, $D - [65,73]$, $E - [56,64]$, and $FX - [0,55]$.

[2] Peoplia: `http://www.peoplia.org/`

[3] USC CodeCount: `http://sunset.usc.edu/research/CODECOUNT/`

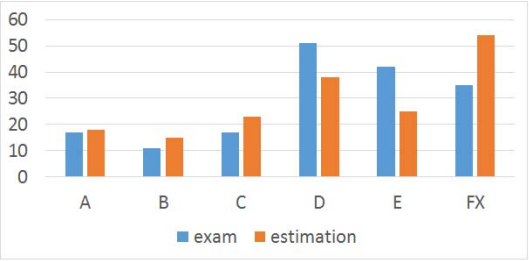

Fig. 1. Comparison of how automatic estimation of students' expertise correlates with exam results

The result of our experiment is illustrated in Figure 1. The number of concordant pairs equals 143 and the number of discordant pairs equals 30. The total number of pairs equals 173. We calculated precision as 0.83.

The idea of "webification" of software development which is based on viewing software repositories as webs is not new. Already Knuth in 1984 presented the idea that "a program is best thought of as a web"[3]. The novel aspect lies in considering not only software artifacts but also users (developers) together with their explicit and implicit feedback, which brings a new view on software and software process metrics. It helps developers be more efficient and can enrich them with the experience and knowledge of their colleagues while managers or senior developers can get advantage of improved planning and decision support via aggregation of statistical data for individual developers.

Acknowledgement. This work was partially supported by the Scientific Grant Agency of the Slovak Republic, grant No. VG1/0675/11 and it is the partial result of the Research & Development Operational Programme for the project PerConIK, ITMS 26240220039, co-funded by the ERDF.

References

1. Bieliková, M., Polášek, I., Barla, M., Kuric, E., Rástočný, K., Tvarožek, J., Lacko, P.: Platform independent software development monitoring: Design of an architecture. In: Geffert, V., Preneel, B., Rovan, B., Štuller, J., Tjoa, A.M. (eds.) SOFSEM 2014. LNCS, vol. 8327, pp. 126–137. Springer, Heidelberg (2014)
2. Fritz, T., Ou, J., Murphy, G.C., Murphy-Hill, E.: A degree-of-knowledge model to capture source code familiarity. In: Proc. of the 32nd Int. Conf. on Softw. Eng., vol. 1, pp. 385–394. ACM, USA (2010)
3. Knuth, D.E.: Literate programming. Comput. J. 27(2), 97–111 (1984)
4. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, USA (2008)
5. Minto, S., Murphy, G.C.: Recommending emergent teams. In: Proc. of the 4th Int. Workshop on Mining Softw. Repositories. IEEE Computer Society, USA (2007)