Estimation of Student's Programming Expertise

Eduard Kuric
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 2, 842 16 Bratislava 4, Slovakia
eduard.kuric@stuba.sk

Mária Bieliková
Faculty of Informatics and Information Technologies
Slovak University of Technology
Ilkovičova 2, 842 16 Bratislava 4, Slovakia
maria.bielikova@stuba.sk

ABSTRACT

Context: Despite the fact, that the various automated expertise metrics were proposed, we do not know which metrics the most reliably capture/reflect expertise. ***Goal:*** To define metrics for estimation of developer's expertise based on programming tasks, to evaluate which of them most reliably capture expertise, and to propose and evaluate an automatic process to compare the metrics. ***Method:*** We define three expertise metrics with respects to such characteristics as spent time, performed activities and complexity of source code. We evaluate Spearman's correlation between our expertise metrics and students' score obtained after completion of a programming course with 251 students. ***Results:*** The best (very strong) correlation is between the metrics based on complexity of source code and the student's qualification points. ***Conclusions:*** Very strong but not perfect correlation is between our estimation of student's expertise and his/her score in the second third of the course. Approximately in the middle of the course we might be able to predict students' grades.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics

General Terms

Measurement, Experimentation

Keywords

expertise, software metrics, source code, interaction data, software development, programming tasks

1. INTRODUCTION

Software metrics are of limited use unless we can be sure about what they mean and what they imply [5]. The estimation of developer's expertise is a crucial factor for the development time required to complete a given task. In other words, it has an impact on how quickly and successfully development task is supposed to be solved. Even though that this statement is intuitive, surprisingly enough, research has not yet quantified the effect of developer's expertise on development time, i.e., it has not yet been thoroughly investigated empirically. Another problem is that there are many competing definitions of expertise. Without a clear baseline, it is problematic to determine which expertise metrics are the best characterization of expertise.

Estimation of expertise on a given piece of source code (source code artifact) has an impact on productivity, i.e., a developer that knows the functions/methods and classes to use for a target task does not need to consult the documentation nearly as much. Estimation of developer's expertise also allows, 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 (the most suitable developer can be found to help on a given task - someone who knows a software component or an application interface).

To the present, we have focused mainly on modeling developer's expertise in a software house environment [2]. It is based on investigation of software artifacts that the developer creates. We take into account the developer's source code contribution, its complexity and how the contribution were created to a software artifact (e.g. copy/paste actions from a web browser); the developer's know-how persistence about a software artifact; and technological know-how. All on daily basis of software development.

Despite the fact that various automated expertise metrics were proposed, we do not know which metric most reliably captures/reflects expertise. The main obstacle for comparison of expertise metrics is the lack of data with which to compare the metrics to each other. If we want to estimate developer's expertise the best possible way we have to consider comparable values only, to ensure the same conditions, and to ensure the same evaluation criteria.

In this paper, we present an approach for estimation of developer's expertise based on programming tasks. We define several metrics in order to find out which of them most reliably captures expertise. To compare and evaluate our expertise metrics properly we require the following: (1) developers solve tasks of equal size and complexity; (2) they have the same time for solving the tasks; (3) they solve the tasks in a uniform working environment; and (4) they use the same development environment. The study was carried in the academic environment with student participants. Program-

ESEM'14 September 18-19, 2014, Torino, Italy.

ming courses, aimed at practicing of student's algorithmic thinking, are often based on acquiring deeper knowledge of methods of designing and implementing efficient algorithms. Our approach is based on programming tasks which the students solve during seminars of a programming course. Each task is focused on training and acquiring skills of a concept (e.g. *priority queue, hash table, balanced 2-3-4 tree*).

We define three expertise metrics and we compare the correlation between each of the metrics and the points that the students gained in seminars and the final exam. We model student's expertise as a level of his/her expertise of a concept relatively to other investigated students. In other words, if students solve a task focused on acquiring skills of a particular concept, then by analyzing their resultant source code, interaction activity and by using appropriate software metrics, we are able to estimate levels of students' expertise of the concept. By using the particular students' expertise estimation of concepts we can estimate a level of student's expertise for the whole course and compare the values among the investigated students based on the same evaluation criteria.

2. RELATED WORK

In this area, several approaches have been proposed. They are usually based on interaction activities, changes, bugs or usage of technologies.

Fritz et al. [3] use interaction data. They proposed a Degree-of-Knowledge model (DoK) to capture source code familiarity. It consists of a component indicating a developer's longer-term knowledge of a source code element - represented by a Degree-of-Authorship (DoA) and a component indicating a developer's shorter-term knowledge - represented by a Degree-of-Interest (DoI). DoA is determined by 3 factors: (1) first authorship, (2) the number of deliveries and (3) the number of acceptances. DoI represents the amount of interaction actions (e.g. edits) a developer has had with a source code element. DoK is calculated as a linear combination of the factors contributing to DoA and DoI. The developer's expertise rises when he/she commits changes and diminishes when other developers make changes.

The Expertise Browser [8] and Expertise Recommender [6] use a simple heuristic (Line 10 Rule) that the developer who commits to a file has expertise in the file. The Expertise Recommender estimates developer's expertise as a binary function, i.e., at a certain time only one developer can have expertise in a file and it depends on who last changed it (last one wins). An argument is that the developer who last made a change has the source code "freshest" in mind. A refinement of the Expertise Browser is proposed in [4], where the authors take into account also size of the changes (i.e. the number of lines changed in a commit). The Emergent Expertise Locator [7] and Expertise Recommender improve the Expertise Browser so that they consider a relationship among changes in a file while estimating expertise.

Anvik and Murphy [1] use bug reports as the source of information for estimation of expertise. Expert Finder [9] allows to locate experts on Java APIs based on their usage.

Our approach differs from these works by contributing an automatic process to compare expertise metrics based on programming tasks, while other approaches rely on developer feedback which is expensive to gather, and potentially subjective to evaluate the expertise metrics for a given source code artifact.

3. METHODOLOGY

We argue that there are at least three key aspects in estimation of developer's expertise that we have to take into account, namely, *time*, *activity*, and *complexity*. In other words, well-performing expertise metric should exhibit at least one of the following characteristics:

- It is negatively correlated with the time taken to perform a task of equal size. Experts take comparatively less time than non-experts for a task of equal size.
- It is negatively correlated with the total number of activities (e.g. build/debug attempts, edits, selects, switches) needed for the delivery of well-working (acceptable) solution for a given task of equal size. For example, experts usually need/perform less build attempts than non-experts during creating well-working solution. Another example, experts usually do not perform considerably less edits than non-experts, but they arguably perform them faster (in less steps), that is, the density of navigation, edit actions is higher.
- It is also negatively correlated with the amount of effort to perform a given task of equal size. Effort can be estimated based on such characteristics as complexity and size. For example, source lines of code (SLOC) can be very effective in estimating effort as a software size metric. Programs with larger SLOC values take more time to develop. Experts usually produce less source lines of code than non-experts for a given task of equal size. On the one hand, SLOC is useful basic metric to measure size, on the other hand, it positively correlates with complexity. Based on the facts, we can "narrow" effort to complexity of source code expressed by a sizing metric such as SLOC.

As we mentioned our main objective is to define proper expertise metrics for estimation of developer's expertise based on programming tasks and to evaluate which of them most reliably captures expertise. We define three expertise metrics with respect to the characteristics highlighted above.

Time (T). Estimation of developer's expertise of a concept c, when taking into account the time needed to perform a given task $t_{c_i} \in T_c$, is calculated as follows:

$$Exp_c^T(d, T_c) = \sum_{t_{c_i} \in T_c} \frac{1}{log_2(1 + TD(d, t_{c_i}))}, \qquad (1)$$

where $TD(d, t_{c_i})$ returns time duration (TD) from the first to the last (acceptable) build of the solution for t_{c_i}. The less time a developer d needs to perform a task t_{c_i} the higher his/her expertise Exp_c^T is.

Activity (A). Estimation of developer's expertise of a concept c, when taking into account the total number of events (EA) corresponding with selected activities needed for the delivery of the acceptable solution for a given task $t_{c_i} \in T_c$, is calculated as follows:

$$Exp_c^A(d, T_c) = \sum_{t_{c_i} \in T_c} \frac{MED_{EA}(t_{c_i})}{EA(d, t_{c_i})}, \qquad (2)$$

$$MED_{EA}(t_{c_i}) = median(\cup_i EA(d_i, t_{c_i}))$$

For example, the higher the number of build attempts a developer performs in solving t_{c_i} the less his/her Exp_c^A is.

Table 1: Spearman correlation between students' score (points or grades) and metrics.

Score \ Metrics	EXP^T	#time	EXP^A	#NOBA	EXP^C	#LLOC
qualification points	0.80	0.71	0.83	0.74	**0.86**	0.84
exam points	0.67	0.59	0.70	0.61	**0.73**	0.71
total points	0.78	0.70	0.82	0.73	**0.85**	0.84
qualification grades	0.72	0.51	0.74	0.56	**0.80**	0.78
exam grades	0.56	0.51	0.56	0.40	**0.60**	0.55
final grades	0.69	0.50	0.72	0.54	**0.77**	0.74

Complexity (C). Estimation of developer's expertise of a concept c, when taking into account a product metric (PM) of a delivered solution for a given task $t_{c_i} \in T_c$ that reflects complexity at least to some extent (e.g. cyclomatic complexity), is calculated as follows:

$$Exp_c^C(d, T_c) = \sum_{t_{c_i} \in T_c} \frac{MED_{PM}(t_{c_i})}{PM(d, t_{c_i})}, \quad (3)$$

$$MED_{PM}(t_{c_i}) = median(\cup_i PM(d_i, t_{c_i}))$$

The more complex a developer's solution of t_{c_i} is (respective to other developers) the less his/her Exp_c^C is.

Based on our particular expertise metrics we estimate developer's expertise of the course as follows:

$$EXP^{(T|A|C)}(d) = \sum_c Exp_c^{(T|A|C)}(d, T_c), \quad (4)$$

We normalize all the calculated expertise values into the interval $[0, 1]$ by using the *tanh* normalization technique.

4. EVALUATION

Environment. We focused on estimation of students' expertise in a course *Data structures and algorithms* (bachelor study). The aim of this course is to provide students with a general overview of principles of specifying data types. Students should acquire deeper knowledge of approaches of designing and methods of implementing efficient algorithms and data structures for searching and sorting. We experimented with data gathered during the course. During seminars the students solve programming tasks. Each week is focused on training and acquiring skills of a concept such as *stack*, *priority queue*, *binary tree*, *oriented graph*, *hash table*, etc. The students solve the tasks in a learning system called *Peoplia*[1]. Students can select to solve a simpler or more complex task focused on acquiring skills of a concept. The number of tasks equals 20. Students get points for their successful solutions. In autumn semester 2013/2014, 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 submissions and the solutions are checked by a plagiarism detection system.

Concrete Metrics. In experiments, to estimate student's expertise based on his/her activity we chose a number of build attempts (NOBA) that he/she performed in solving a task t_{c_i}, i.e., $EA(s, t_{c_i}) = NOBA(s, t_{c_i})$ (see Eq. 2). To estimate complexity of a solution of a task t_{c_i} created by a student s we chose Logical Lines of Code (LLOC), i.e., $PM(s, t_{c_i}) = LLOC(s, t_{c_i})$ (see Eq. 3). The larger a program is (LLOCs) the more complex the program is. Although LLOC is widely accepted sizing metric, there is a lack of standard that enforces a consistency of what and how to count LLOC. For calculation of LLOC we adopted the definition from the CodeCount[2].

General Results. We first examine the correlation between our expertise metrics and student's score obtained after completion of the course. We report the non-parametric Spearman correlation. We use the asymptotic variant of the Spearman correlation as the exact variant is sensitive to ties. The student's score (*total points*) is divided into points earned in seminars (*qualification points*) and points of the final exam (*exam points*). We normalize *exam*, *qualification* and *total* points into the interval $[0, 100]$. Subsequently, we map the normalized points as follows: $[92, 100]$ to 1 (A), $[83, 91]$ to 2 (B), $[74, 82]$ to 3 (C), $[65, 73]$ to 4 (D), $[56, 64]$ to 5 (E), and $[0, 55]$ to 6 (FX), i.e., student's *qualification points* are mapped to *qualification grade*, his/her *exam points* are mapped to *exam grade* and *total points* to *final grade*.

Table 1 shows the correlation between our expertise metrics calculated based on Eq. 4 and students' course evaluation, i.e., $\{EXP^T, EXP^A, EXP^C\} \times \{$*qualification points*, *exam points*, *total points*, *qualification grade*, *exam grade*, *final grade*$\}$. Note, before calculating correlation between EXP^T and *qualification grades*, we normalize EXP^T values to grades (the same is applied in all comparisons with grades). In Table 1 we also provide calculated correlation between basic metrics and students' evaluation. *#time* is the total time the student spent in solving all tasks (accepted solutions), *#NOBA* is the total number of build attempts the student performed in solving all tasks (accepted solutions), and *#LLOC* is the total number of logical lines of code containing in his/her accepted solutions.

We have mentioned that expertise should negatively correlate with the time taken to perform a given task. As we can see our metric EXP^T reflects the negative correlation. In all cases EXP^T is higher in comparing with *#time*, i.e., when we "penalize" a student the correlation is higher. Similarly it is between EXP^A and *#NOBA*, and EXP^C and *#LLOC*. We can see that EXP^C and *qualification points* are very strongly but not perfectly correlated. We can also see that very strong correlation is between EXP^C and *total points*. In comparing, EXP^C provides the best results.

Student's score prediction. An example of practical usage of our approach is student's score prediction. We were interested in whether we are able to predict students' grades during the course. Our goal was to find out at what phase of the course the correlation between our metrics and students' evaluation is the highest. Therefore we divided the 20 tasks

[1]Peoplia: http://www.peoplia.org/fiit/

[2]CodeCount: http://sunset.usc.edu/research/CODECOUNT/

Table 2: Spearman correlation between students' score and calculated EXP^T from a subset of tasks.

Score \ Tasks	I_1^T	I_2^T	I_3^T	$I_1^T \cup I_2^T$	$I_2^T \cup I_3^T$
qualif. points	0.62	0.80	0.68	0.79	0.81
exam points	0.53	0.67	0.58	0.67	0.67
total points	0.61	0.78	0.67	0.79	0.79
qualif. grades	0.55	0.67	0.54	0.70	0.70
exam grades	0.43	0.54	0.38	0.58	0.51
final grades	0.52	0.66	0.54	0.69	0.67

Table 3: Spearman correlation between students' score and calculated EXP^A from a subset of tasks.

Score \ Tasks	I_1^T	I_2^T	I_3^T	$I_1^T \cup I_2^T$	$I_2^T \cup I_3^T$
qualif. points	0.65	0.82	0.69	0.81	0.83
exam points	0.55	0.69	0.58	0.69	0.69
total points	0.64	0.80	0.67	0.80	0.81
qualif. grades	0.58	0.69	0.56	0.73	0.72
exam grades	0.44	0.55	0.38	0.57	0.51
final grades	0.55	0.67	0.54	0.70	0.68

into three intervals I_1^T, I_2^T, and I_3^T. I_1^T contains 7 tasks assigned to students during the first third of the course, I_2^T contains 7 tasks (the second third), and I_3^T contains 6 tasks (the last third). In Table 2 we compare EXP^T divided into the 3 intervals with students' evaluation. Similarly, EXP^A in Table 3, EXP^C in Table 4. As we can see, the metric EXP^C again wins. The result is that the very strong but not perfect correlation is in the second third of the course. It means that approximately in the middle of the course we might be able to predict students' grades.

5. CONCLUSIONS

The estimation of developers' expertise is a valuable asset for a software company. We introduced an approach for estimation of student's expertise based on programming tasks in an academic environment. Although, we performed the evaluation of our expertise metrics in laboratory conditions we think that the presented hypothesis are also valid in a software house environment. We realize that there are much more aspects that affect developers in a software house environment. We should also take into account such aspects as forgetting, development styles, etc. We also suspect that the effect will be more strongly felt on larger tasks. Despite all the facts, we believe that the investigated characteristics are also valid in a software house due to their "intuitiveness", i.e., the relationship (individually) between *time*, *activity*, *complexity* and estimation of expertise is in negative correlation. However, the best way to measure developer's expertise on a source code artifact is still up for discussion.

We shown that the presented metrics are well applicable independently, however, an issue is how to combine them so that we can be sure about what they mean and what they imply. Therefore, we focus on different aspects of evaluation. Our work is a part of a research project called PerConIK[3] (Personalized Conveying of Information and Knowledge). We focus on support of applications development by viewing a software system as a web of information artifacts. Our aim is to devise the right metrics to evaluate software artifacts and to identify particular problems and recommending corrective actions. We developed several agents that collect and process documentations, source code repositories, developers' activities, etc. [2]. We create within the project a dataset of developers' implicit/explicit feedback based on monitoring behavior of developers (e.g. searching relevant information on the Web, writing/correcting source code in an IDE, peer review) and defining explicit feedback from the developers for the purpose of estimating expertise.

[3]PerConIK: http://perconik.fiit.stuba.sk/

Table 4: Spearman correlation between students' score and calculated EXP^C from a subset of tasks.

Score \ Tasks	I_1^T	I_2^T	I_3^T	$I_1^T \cup I_2^T$	$I_2^T \cup I_3^T$
qualif. points	0.63	**0.85**	0.73	0.84	**0.86**
exam points	0.55	**0.68**	0.64	0.70	**0.71**
total points	0.63	**0.82**	0.73	0.82	**0.84**
qualif. grades	0.51	**0.79**	0.63	0.79	**0.80**
exam grades	0.43	**0.54**	0.45	0.60	**0.58**
final grades	0.51	**0.73**	0.61	0.75	**0.76**

Acknowledgement. This work was partially supported by grant No. VG1/0675/11 and it is the partial result of the Research and Development Operational Programme project "University Science Park of STU Bratislava", ITMS 26240220084, co-funded by the European Regional Development Fund.

6. REFERENCES

[1] J. Anvik and G. C. Murphy. Determining implementation expertise from bug reports. In *Proc. of the 4th Int. Workshop on Mining Softw. Repositories*, pages 2–. IEEE CS, 2007.

[2] M. Bieliková et al. Platform independent software development monitoring: Design of an architecture. In *Proc. of the 40th Int. Conf. on Current Trends in Theory and Practice of Computer Science - Vol. 8327*, pages 126–137. Springer LNCS, 2014.

[3] T. Fritz et al. A degree-of-knowledge model to capture source code familiarity. In *Proc. of the 32nd Int. Conf. on Softw. Eng. - Vol. 1*, pages 385–394. ACM, 2010.

[4] T. Girba et al. How developers drive software evolution. In *Proc. of the 8th Int. Workshop on Principles of Softw. Evolution*, pages 113–122. IEEE CS, 2005.

[5] B. Kitchenham. Never mind the metrics what about the numbers. In *Proc. of the BCS-FACS Workshop on Formal Aspects of Measurement*, pages 28–37. Springer-Verlag, 1992.

[6] D. W. McDonald and M. S. Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *Proc. of the Conf. on Comp. Supported Cooperative Work*, pages 231–240. ACM, 2000.

[7] S. Minto and G. C. Murphy. Recommending emergent teams. In *Proc. of the 4th Int. Workshop on Mining Softw. Repositories*, pages 5–. IEEE CS, 2007.

[8] A. Mockus and J. D. Herbsleb. Expertise browser: a quantitative approach to identifying expertise. In *Proc. of the 24th Int. Conf. on Softw. Eng.*, pages 503–512. ACM, 2002.

[9] A. Vivacqua and H. Lieberman. Agents to assist in finding help. In *Proc of the SIGCHI Conf. on Human Factors in Comp. Systems*, pages 65–72. ACM, 2000.