Using Developer Interaction Data to Compare Expertise Metrics

Romain Robbes
PLEIAD Laboratory
Computer Science Department
University of Chile
rrobbes@dcc.uchile.cl

David Röthlisberger
Computer Science Department
Universidad Técnica Federico
Santa María, Chile
roethlis@inf.utsm.cl

***Abstract*—The expertise of a software developer is said to be a crucial factor for the development time required to complete a task. Even if this hypothesis is intuitive, research has not yet quantified the effect of developer expertise on development time. A related problem is that the design space for expertise metrics is large; out of the various automated expertise metrics proposed, we do not know which metric most reliably captures expertise.**

What prevents a proper evaluation of expertise metrics and their relation with development time is the lack of data on development tasks, such as their precise duration. Fortunately, this data is starting to become available in the form of growing developer interaction repositories. We show that applying MSR techniques to these developer interaction repositories gives us the necessary tools to perform such an evaluation.

I. INTRODUCTION

Developer expertise is both a major factor in software projects and has, surprisingly enough, not yet been thoroughly investigated empirically. Anecdotal evidence is sufficient to realize that expertise on a given piece of code has a large impact on productivity: a developer that knows the methods and classes to use for a given task does not need to consult the documentation nearly as much. As such, most of the research on expertise has focused on expert recommendation, where an expertise metric is computed for each developer, so that the most suitable developer can be found to help on a given task.

However, there are many possible and competing definitions of expertise, with few comparisons between them. Without a clear baseline, it is hard to determine which automated expertise metric is the best characterization of expertise. The main obstacle to a comparison of expertise metrics is the lack of data with which to compare the metrics.

We claim that repositories of fine-grained developer interactions—recorded with tools such as Mylyn [1], Syde [2], or Spyware [3]—contain the data needed for a comparison of expertise metrics. These repositories of developer interactions contain several relevant data points for expertise metrics: (1) duration, or the time a developer took to perform a development task; (2) edits, *i.e.* the sequence of code changes that were necessary to perform said task; and (3) selections, or the code elements that were consulted, during the task implementation. More widespread sources of data, such as version control systems, store only the final outcome of a development session in a commit, and not the actual programmer activity [4], making them unsuitable for our needs.

We argue that a well-performing expertise metric, should exhibit the following characteristics: (1) it is inversely correlated with the time taken to perform a given task (experts take comparatively less time than non-experts for a task of equal size); and (2) it is also inversely, but less strongly, correlated with the amount of activity (editions, selections) to perform a task (experts usually do not perform considerably less edits than non-experts, but they arguably perform them faster, that is, the density of navigation, edit actions is higher). The best estimate of an expertise metric will have a stronger inverse correlation to the duration, and a weaker correlation with editions and selection. We also suspect that the effect will be more strongly felt on larger tasks.

In this paper, we propose a novel approach to automatically evaluate expertise metrics based on the assumptions and premises mentioned above, using developer activity information from a large dataset of more than 1,700 development sessions[1], authored by 31 developers, and belonging to two development projects. We define two expertise metrics, find that both metrics exhibit the characteristics highlighted above, and compare them.

II. RELATED WORK

Several works have defined expertise metrics.

Changes. McDonald [5] presents a recommendation system that uses Change History and Tech Support heuristic to perform its recommendations. Change History information is based on the "Line 10 rule", *i.e.* , the developer whose name appears on line 10 of the change log—the last person to change the file—is considered an expert as he or she is the person with the "freshest" memory of the file. The Expertise Browser of Mockus and Herbsleb [6] uses the same rule, but also counts the number of times a given developer changed the file as a measure for his expertise. A later refinement of this is the work by Girba *et al.* [7], where the size of the changes (measured by the number of lines changed in a commit) is also taken into account to determine expertise.

Usage. Vivacgua and Lieberman [8] present Expert Finder, a system that locates experts on java APIs based on their usage

[1]As a development session we consider a consecutive collection of developer interactions; a session ends with a commit or if more than two hours time has elapsed between two interactions.

of the API. Ma *et al.* [9] also present an expert recommender based on usage expertise.

Defects. Anvik and Murphy [10] use bug reports as the primary source of information for their expertise metric.

Interaction data. Fritz *et al.* [11] investigated whether interaction data from Mylyn's Degree of Interest indicates knowledge of the source code viewed, by having subjects fill out a questionnaire, and found that recent interactions where associated with a higher knowledge of the source code. Later, Fritz *et al.* [12] expanded the Degree of Interest in a Degree of Knowledge, and compared it to Mockus and Herbsleb's expertise recommender, finding an improvement according to developer feedback.

Fine-grained changes. Hattori *et al.* [13] proposed a finer-grained definition of ownership, based on recorded developer interactions, instead of ones recovered from the versioning system. They also propose to use the notion of forgetting to obtain a more realistic notion of expertise decaying with time.

Our approach differs from these related works by contributing an automatic process to compare expertise metrics, based on fine-grained developer activity information, while other approaches use more coarse-grained information (*e.g.* based on source code history or bug reports), or rely on developer feedback—expensive to gather, and potentially subjective—to evaluate the expertise metrics for a given source artifact.

III. Methodology

A. Data gathering

The Mylyn [1] tool offers integration with defect tracking systems: a developer submitting a bug fix via Mylyn will upload his Mylyn interaction history as an attachment to the bug fix. As a consequence, there are now several thousands of bug fixes in the Eclipse bug tracker with developer interaction histories attached, principally in two projects, Mylyn itself, and the Plugin Development Environment (PDE).

To gather the data necessary for this experiment, we downloaded all the 9,280 bugs reports with a Mylyn attachment in Eclipse's bugzilla repository (http://bugs.eclipse.org) that were online and publicly visible on July 6th, 2012. We then applied filters on the data, namely: (1) we kept all the data originating from the Mylyn and PDE projects, and (2) we filtered out interaction histories which did not contain any edits.

We also downloaded the git repositories of Mylyn and PDE, in order to compute expertise metrics based on version control histories. However, not all the entities present in a development session were successfully associated to files present in the repository. We thus filtered out development sessions for which more than half of the events could not have any expertise value computed. For the remaining sessions, we removed the events with missing information while computing the expertise value associated with each session.

B. Expertise metrics

In this study, we compare two expertise metrics to illustrate our approach for empirically evaluating expertise metrics using developer activity information. A "naïve" expertise metric, that equates expertise on a given entity to the amount of commits a given developer performs on it, and a "decay" expertise metric that, similar to Hattori *et al.* [13], takes into account the recency of the changes.

Computing expertise metrics. Our expertise metrics are defined at one point in time, and for a given developer, respective to other developers. At time t, to obtain the expertise of developer d on the source code entity e, we: (1) compute the raw expertise metric (according to one of the variants below), giving us a positive value, for each possible developer; and (2), normalize the expertise value as the ratio of the expertise of d on the sum of the expertise of each developers, yielding an expertise value between 0 and 1.

$$Exp(d,e,t) = \frac{RawExp(d,e,t)}{\sum_{d' \in D} RawExp(d',e,t)} \quad (1)$$

Expertise variants. The raw naïve expertise $ExpNaive$ is simply the sum of commits that d performed on the file that contains e, until time t.

$$ExpNaive(d,e,t) = \sum\nolimits_{c \in C} WN(c,d,e,t) \quad (2)$$

$$WN(c,d,e,t) = \begin{cases} 1 & Valid(c,d,e,t) \\ 0 & otherwise \end{cases} \quad (3)$$

$$Valid(c,d,e,t) = Author(c,d) \wedge Changes(c,e) \wedge Date(c) < t \quad (4)$$

The decay expertise metric $ExpDecay$ also counts the number of changes perform by d on the file that contains e, until t, but also applies a liner decaying factor: the weight of each commit c is 1, divided by the number of days between t and the date of the commit. As such, changes performed earlier in the past have a smaller weight than recent changes, showing the propensity of people to forget what they learned over time.

$$ExpDecay(d,e,t) = \sum\nolimits_{c \in C} WD(c,d,e,t) \quad (5)$$

$$WD(c,d,e,t) = \begin{cases} \frac{1}{t - Date(c)} & Valid(c,d,e,t) \\ 0 & otherwise \end{cases} \quad (6)$$

Aggregating expertise at the level of sessions. So far, we described the computation of the expertise of a single code entity. To calculate the expertise of a developer with respect to a task, we compute the expertise for each entity in the session, for the developer that sent the mylyn interaction history, at a time t, which is the start of the session. This yields a list of expertise metrics between 0 and 1, for each entity in the session (excluding entities with missing information). We then perform a weighted average of each entity, where the weight is the number of occurrences of the entity in the session, and obtain a single expertise value for d at time t.

$$Expertise(d,s) = \frac{\sum_{e \in Entities} ExpEntity(e,d,s)}{NumberOfEvents(s)} \quad (7)$$

$$ExpEntity(e,d,s) = Exp(d,e,Start(s)) \times Freq(e,s) \quad (8)$$

C. Validation

As described in the introduction, we are interested in an expertise metric that is: (1) negatively correlated with time, and (2) weakly correlated with the actual activity in the session.

We compute several additional metrics for each sessions. They are: (1) $Time$, the length of time of the session, which is the difference between the timestamps of the last and the first events; (2) $Files$, the number of individual files that were edited or viewed during the session; (3) $Edits$, the number of edition events during the session; and (4) $Selections$, the number of selection events during the session.

To evaluate the accuracy of the metrics (that is, how well an expertise metric explains decreasing amount of time spent or of navigation, edit activity), we compute the overall correlation between $ExpNaive$, $ExpDecay$, and $Time$, $Edits$, and $Selections$. We also want to investigate whether the expertise is a more important factor for larger tasks. As such, we also compute similar correlations, but on the first, second, third and fourth quartiles of the data as determined by the $Files$ metric.

To be less sensible to outliers, we report the non-parametric Spearman correlation instead of the Pearson correlation. We use the asymptotic variant of the Spearman correlation as the exact variant is sensitive to ties. In the tables showing correlations with expertise, correlations which are not significant at $\alpha = 0.05$ are displayed in parenthesis; correlations above a threshold of 0.15 are highlighted in bold.

IV. Results

A. General results

We first examine the correlations between the basic metrics, without considering expertise yet. Table I shows the correlations between time, edits, and selections, across all sessions, and also across development session quartiles. From the table, we observe the following:

- $Time$ and $Edits$, and $Time$ and $Selections$, are correlated. The correlations are however not perfect: there are other factors at play beyond the raw size of the task that influence its duration. Most likely, expertise is such a factor, and its effect is hopefully measurable.
- $Selections$ and $Edits$ are very strongly but not perfectly correlated, which means that some sessions are more exploratory in nature than others.
- If we look at the behavior of these correlations among quartiles of the data, we do not see explicit trends, with the exception that larger sessions feature a slightly higher correlation between $Selections$ and $Edits$.

Metrics	All	Q1	Q2	Q3	Q4
Time & Edits	0.66	0.61	0.38	0.35	0.52
Time & Selections	0.73	0.59	0.39	0.34	0.61
Selections & Edits	0.90	0.78	0.89	0.90	0.93

TABLE I
SPEARMAN CORRELATIONS BETWEEN BASIC METRICS

Metric	All	Q1	Q2	Q3	Q4
Time	**-0.15**	(-0.04)	-0.12	-0.13	**-0.32**
Edits	0.05	0.10	**0.18**	0.14	(0.03)
Selections	(0.01)	0.10	**0.20**	0.12	(-0.09)

TABLE II
SPEARMAN CORRELATIONS OF NAIVE EXPERTISE

Metric	All	Q1	Q2	Q3	Q4
Time	**-0.22**	(-0.02)	**-0.16**	**-0.21**	**-0.35**
Edits	(-0.04)	(0.07)	(0.09)	0.12	(-0.09)
Selections	-0.08	0.09	0.11	(0.1)	**-0.16**

TABLE III
SPEARMAN CORRELATIONS OF DECAY EXPERTISE

B. Naive expertise metric

We then examine the correlations of the basic metrics with our naïve expertise metric, in Table II. Overall, we see that:

- There is a weak, but significant, negative correlation between the $ExpNaive$ and $Time$. If we analyze the correlations by quartiles, we see that the correlation for the first quartile is very weak and non-significant. The correlation grows across quartiles, as the size of tasks grows. For the larger quartile, the correlation is as high as -0.32, which, although not a strong correlation, is still remarkable, especially if compared to the correlation between $Time$ and $Edits$ (0.52).
- The relationship between $ExpNaive$ and $Edits$ is hard to describe: we observe some correlation, but they are certainly weaker than the correlation with time. Most of them are positive correlations, which comforts us in the belief that $ExpNaive$ is describing a different aspect than $Time$ and $Edits$, which are positively correlated.
- We can make similar observations with respect to the relationship between $ExpNaive$ and $Selections$. The correlations are positive, but overall weaker than the negative correlation of $ExpNaive$ and $Time$.

Our first attempt at defining an expertise metric is encouraging: We defined a metric that is negatively correlated with $Time$, but not strongly correlated with $Edits$ or $Selections$.

C. Does decay lead to a better expertise metric?

Table III presents the correlations we observed between $ExpDecay$, our expertise metric that takes into account decay, and $Time$, $Edits$, and $Selections$. Our first observation is that its overall behavior is similar to the $ExpNaive$. Both metrics have negative correlations with $Time$, which increase with larger tasks. Both metrics have relatively weak correlations with $Edits$ and $Selections$ by themselves. However, several factors lead us to believe that the $ExpDecay$ is a better indicator of expertise than $ExpNaive$:

- The negative correlation with $Time$ is stronger: the overall correlation for all sessions goes from -0.15 to -0.22, a 50% increase. We see increases in all quartiles that previously featured a significant correlation, with most pronounced effects in the second and the third quartile.
- The correlation with $Edits$ and $Selections$ is weaker: the values across the board are overall lower, and additionally, 5 of the measurements are not statistically significant, up from 3 for $ExpNaive$.

For all of these reasons, we think that $ExpDecay$ better isolates the actual expertise of a developer, and is hence a better expertise metric overall. However, $ExpDecay$ is still a fairly naive metric and more sophisticated metrics (*e.g.* metrics taking into account the extent of the change to a source artifact per commit) are likely to (negatively) correlate better with time spent and number of selections, edits. With the proposed approach, we can easily automatically test whether any newly defined expertise metrics shows a higher correlation.

V. Threats to validity

Construction validity. Missing information may be the source of errors in the measurement. In particular, we were not able to find the change history of all the files that were changed in the development sessions. We took measures to limit this effect, by discarding sessions with too much missing data, but we cannot guarantee to not have introduced a bias in the data in this way. The fact that developers where Mylyn users means that Mylyn itself may have an impact on their productivity, introducing systematic errors in the measurements.

Statistical conclusion validity. In this study, we present only correlations between our variables of interest. It is well-known that correlation does not imply causation, as such we cannot be certain other variables are at play. In addition, we did not attempt to predict the time taken in a development session based on the expertise metrics and/or other metrics.

External Validity. Due to the scarcity of publicly available interaction data repositories, our findings concern two projects only (Mylyn and PDE). We cannot be sure the findings extend to other projects, or if they do, whether the magnitude of the effects are similar.

Internal validity. Our validation of expertise measurement takes the assumption that a better expertise measurement is more strongly (negatively) correlated to the time taken to perform a task. There may be other aspects to expertise that render this assumption incorrect.

Our expertise computation taking into account forgetting may be too naive. There are other aspects of expertise that we did not take into account, such as the size of the changes, etc. In particular suppose class A and B were both modified twice by developer X, they should have the same expertise value for X. If however class A has been furthermore heavily modified by other authors, X's expertise for B should be higher. This is achieved to some extent by the fact that the total expertise for a class is constant and shared between all developers, but this may have undesired side effects as well. Different development styles regarding committing of changes also influence the expertise metrics, that is, a developer committing very frequently even small changes to an entity will have a high expertise on this entity than a developer who commits less often. However this does not concern the evaluation framework itself, but the two metrics we define; additional metrics should sidestep these issues.

VI. Conclusions and future work

The expertise of a developer is a valuable asset for a company. However, the best way to measure the expertise of a developer on a piece of code is still up for debate. As a result, several expertise metrics have been proposed, but how to evaluate their quality and accuracy is not trivial.

In this paper, we proposed to mine developer interaction data repositories to evaluate the respective merits of various expertise metrics. These interaction data repositories contain the sequence of activities that developers took to finish a given task, including the overall time elapsed to perform the task.

We hence propose to use the time taken to finish a task as a way to measure the effectiveness of tentative expertise metrics. The rationale behind this is that a good expertise metric should be both negatively correlated with the time taken to perform a task, and weakly correlated with the actual task activity.

We defined two expertise metrics based on the past activity on the entities involved in the actual task, and found that both metrics performed according to our expectations. That is, both metrics exhibited the characteristics described above. Furthermore, the expertise metric that discounts older activity on an entity was found to be a better expertise metric than the one that did not: it exhibited a stronger negative correlation to time, and a lower correlation to the task activity.

In future work, we plan to use the same framework to evaluate additional expertise metrics, with the aim to find better estimators of the expertise of developers. We plan to investigate several aspects that have been proposed in the literature, such as finer characterizations of the changes made to entities, the usage of entities in other tasks, and other metrics such as expertise based on vocabulary.

References

[1] M. Kersten and G. C. Murphy, "Using task context to improve programmer productivity," in *SIGSOFT FSE*, M. Young and P. T. Devanbu, Eds. ACM, 2006, pp. 1–11.

[2] L. Hattori and M. Lanza, "Syde: a tool for collaborative software development," in *ICSE (2)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 235–238.

[3] R. Robbes and M. Lanza, "Spyware: a change-aware development toolset," in *ICSE*, W. Schäfer, M. B. Dwyer, and V. Gruhn, Eds. ACM, 2008, pp. 847–850.

[4] R. Robbes and M. Lanza., "Versioning systems for evolution research," in *IWPSE*, 2005, pp. 155–164.

[5] D. W. McDonald, "Evaluating expertise recommendations," in *GROUP*. ACM, 2001, pp. 214–223.

[6] A. Mockus and J. D. Herbsleb, "Expertise browser: a quantitative approach to identifying expertise," in *ICSE*, W. Tracz, M. Young, and J. Magee, Eds. ACM, 2002, pp. 503–512.

[7] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse, "How developers drive software evolution," in *IWPSE*. IEEE Computer Society, 2005, pp. 113–122.

[8] A. S. Vivacqua and H. Lieberman, "Agents to assist in finding help," in *CHI*, T. Turner and G. Szwillus, Eds. ACM, 2000, pp. 65–72.

[9] D. Ma, D. Schuler, T. Zimmermann, and J. Sillito, "Expert recommendation with usage expertise," in *ICSM*. IEEE, 2009, pp. 535–538.

[10] J. Anvik and G. C. Murphy, "Determining implementation expertise from bug reports," in *MSR*. IEEE Computer Society, 2007, p. 2.

[11] T. Fritz, G. C. Murphy, and E. Hill, "Does a programmer's activity indicate knowledge of code?" in *ESEC/SIGSOFT FSE*, I. Crnkovic and A. Bertolino, Eds. ACM, 2007, pp. 341–350.

[12] T. Fritz, J. Ou, G. C. Murphy, and E. R. Murphy-Hill, "A degree-of-knowledge model to capture source code familiarity," in *ICSE (1)*, J. Kramer, J. Bishop, P. T. Devanbu, and S. Uchitel, Eds. ACM, 2010, pp. 385–394.

[13] L. Hattori, M. Lanza, and R. Robbes, "Refining code ownership with synchronous changes," *Empirical Software Engineering*, vol. 17, no. 4-5, pp. 467–499, 2012.