Using Developers Contributions on Software Vocabularies to Identify Experts

Katyusco de F. Santos
Federal Institute of Paraíba
Campus Campina Grande
Academic Dept. of Information Technology
Email: katyusco.santos@ifpb.edu.br

Dalton D. S. Guerrero
and Jorge C. A. de Figueiredo
Federal University of Campina Grande
Department of Systems and Computer
Email:{dalton, abrantes}@dsc.ufcg.edu.br

***Abstract*—Developers choose identifiers to name entities during software coding. While these names are lexically restricted by the language, they reflect the understanding of the developer on the requirements that the entity is devoted for. In this paper, we analyze the use of such vocabularies to identify experts on code entities. For a real software development, e-Pol (Management Information System for Federal Police of Brazil), we observed around 30% of its code entities has more than 0.3 of similarity with at least one developer vocabulary.**

We propose an approach to catch this potential expertise that vocabularies carries on. Also, we built an oracle of source code entities per developer that allowed us to assess our approach accuracy compared with two others ones: based on commit and based on percentage of modified *Lines of Codes*. One advantage of our approach is to disregard changes in formatting or indentation of source code as acts of expertise acquisition.

We achieve an accuracy ranging from 0.16 to 0.32 depending on the assumed period of developers' contributions and the *top-k* experts we are interested on. These results confirm similarity between vocabularies might be explored to point out code experts. Moreover, for orphaned entities, expertise approach based on vocabularies can recommend among current team members one whose vocabulary is closest to the entity.

***Index Terms*—software vocabulary, expertise, oracle of experts**

I. Introduction

A constant challenge in software development is to support team leaders to find efficiently (quick and accurate) points in the source code where developers should perform maintenance tasks. But, for a task be implemented with minimal effort (cost and time) it is also necessary to define who among the members of a team is the most appropriate developer to do it.

Maintenance costs represent between 70% and 90% of a software project [1], [2]. During maintenance tasks, code understanding takes 50% [1] to 78% of programmer time, while 20% is to fix bugs and write code just 2%, [3]. Identify the right members of a development team that are experts for each of entities project aims to source code comprehension that is an activity that not only depends on code properties, but also on person experience [4]. Also, code snippets created by a developer can be modified by another one. The contributions, made changes, can exchange the code snippet expert from the original author to the largest contribute developer [5].

The most used approaches to identify experts are based on mining logs records from Version Control System (VCS). For instance, extracting the number of commits [6], [7], or computing percentage of modified *Lines of Code* (*%LoC*) [8], [9]. However, these known approaches endorse it is necessary to add other different aspects they already use to improve their accuracy.

Studies about identifiers and comments used to code software systems, show that they capture different aspects of those who are traditionally captured by static analysis (based on structural relationships) and by dynamic ones (to trace system executions) [10]. Identifiers in part, are named according to concepts, abstractions and representing features of a software system [11], and when summed to comments they represent about 70% of the source code [12]. Furthermore, they can be used to assess code quality [13] and reflect mental map of developers about a project [14]. Source code vocabulary can be used to narrow down the bug-locating task according the results we obtained an empirical experiments we have collaborated [15].

In summary, studies on software vocabularies report they are a valuable source of information about the developers and the system itself, however in the best we know, they have not been explored to model code expertise. Besides, both before performing changes as during new code implementation developers act on vocabulary of entities. For these reasons we question ourselves whether the vocabulary used by developers to name identifiers and to write comment and javadocs texts catch project expertise able to point out entities experts.

To evolve whether developers and source vocabularies can be used to identify experts, first we must be sure they are related. This way we define our first Research Questions (***RQ-1***): *Is there similarity between source code of entities and developers vocabularies?*

Since there is similarity between them, is worth additional effort to measure the accuracy of this relationship. So, we define the second Research Question, (***RQ-2***): *Is similarity can be used to identify source code entities experts?*

In this paper we propose a novel approach to identify experts of Java source entities based on similarity between developers vocabulary and system software vocabulary. We also present the approach accuracy, in terms of precision and recall, compared with two others baselines ones: based on commit and based on *%LoC*.

978-1-4799-8828-0/15 $31.00 © 2015 IEEE
DOI 10.1109/ITNG.2015.149

II. Background

A. Software Vocabulary

Program instructions are written according to strict formal grammar rules to avoid ambiguity, and part of them are formed by reserved words and operators of a language [16].

Developers are the authors of characters sequences, called identifiers, which are used to define, to reference and to manipulate both basic structural elements (*e.g.*: attributes, local variables, methods and parameters names), and more complex ones. In turn, comments are texts also written by developers using natural language with the goal to document explicitly source code elements. Naming structural elements and defining comments contents occur during programming process, and although they are limited to compiler restrictions, they are not tied to the formal dimension of grammar rules [16]. So, developers make use of those mechanisms to transfer their own ideas to and perceptions about software projects

In previous studies [17] we have developed a formal definition of vocabularies to aid the understanding and communication more effective and less ambiguous. Here we briefly present our formalization of vocabularies.

1) Vocabulary definition: We define a software vocabulary as a multiset of strings, i.e. an application $V : \mathbb{S} \to \mathbb{N}$ that maps strings to natural numbers. Elements of a vocabulary are called *terms*. For any term t, $V(t)$ denotes the number of occurrences of the term t in the vocabulary V. If $V(t) > 0$ we say that t is a term of the vocabulary.

As an example, consider the excerpt of java code in Listing 1. According to our definition, the vocabulary can be expressed as the following multiset of terms:

$$V = 4'SampleClass + 2'sa + 2'sc + 1'me + 1'mt$$

Listing 1. Excerpt of Code

```
public static int mt(){
    SampleClass sa = new SampleClass();
    SampleClass sc = new SampleClass();
    sa.me(sc);
}
```

Observe we can adopt existing multiset notations for vocabularies. Above, we have used formal sums, in which each sum term expresses the number of occurrences n of one vocabulary term t in the form $n's$. Other notations can be convenient for other purposes, as well.

2) Vocabulary Properties: For our purposes of expertise identification, size of vocabularies are relevant. For that reason, we have defined two metrics. The first is the total number of terms of a vocabulary, $TT(V)$, i.e. the sum of the multiplicities of all terms in the vocabulary. The second one is the number of distinct terms, $DT(V)$, which is the number of terms whose multiplicity is at least one. For our example above, V, we have

$$TT(V) = 10 \quad and \quad DT(V) = 5$$

3) Vocabulary operations: In practice, one typically needs to process vocabularies prior to performing actual analysis. Typical information retrieval (IR) processing operations, as well as other vocabulary operations, can be easily and unambiguously specified as functions over vocabularies. Take, as an example, a tokenization operation that splits terms according to a given formation rule (camel case, for instance). This operation can easily be specified as a function[1] $cc : \mathbb{V} \to \mathbb{V}$, that maps each pair $(t, m) \in V$ to a set of terms $\{(t_1, m_1), \ldots, (t_n, m_n)\}$, where each term t_i is one of the words that composes the original term t and m_i its respective number of occurrences. For instance, the vocabulary V_1 above maps, by such operation, to

$$cc(V) = 4'Sample + 4'Class + 2'sa + 2'sc + 1'me + 1'mt$$

4) Vocabulary of Source Code: An **entity code** is a complex static structure that encapsulates others one (basic or not). In this study, we have considered entities just classes and interfaces. Associated to an entity there are the identifier that naming it, its own comments and javadocs, as well the identifiers, comments and javadocs of its inner structures. Then, an entity is a vocabulary container of code snippet who is hierarchically on the entity's scope.

The vocabularies union of each entitiy that makes part of a system defines the vocabulary of this software system. We have modeled a software vocabulary as a Term-Entity matrix, TE, where each cell (T_i, E_j) is term frequency of t_i in entity E_j. In formal sums, vocabulary of E_1 is exemplified in matrix TE above, Table I, and is given by: $V(E_1) = 2't_1 + 1't_2 + 1't_3$.

TABLE I
Term-Entity, *TE*, example matrix.

Terms	Entities				
	E_1	E_2	E_3	E_4	E_5
t_1	2	0	1	0	0
t_2	1	2	0	0	0
t_3	1	0	1	0	1
t_4	0	0	1	1	0
t_5	0	0	0	0	1

5) Vocabulary of Developers: The vocabulary of a developer, D, is a result of the developer contributions on source code vocabulary [18]. Each contribution is captured by difference (*diff*) between a previous vocabulary of a given commit and a subsequent one. Like source code vocabulary, Term-Entity matrix represents a contribution. Each new contribution is accumulated to the previous one. Thus, the vocabulary of a developer is a result matrix, TE^D, that means the accumulation of all contributions of developer D.

B. Expertise

Expertise is defined as the ability of the developer to be an expert on given source code, and, if interpreted quantitatively reflects ability degree of developers have to perform a encoding task [6]. Expertise is difficult to measure directly because depends on how it affects development process and software product. The literature then studying expertise indirectly through observation of a variety of process and product measures.

In this study we are interested on assessing the usage of software vocabulary for experts identification. For this, we

[1]We denote the set of all vocabularies by $\mathbb{V}$.

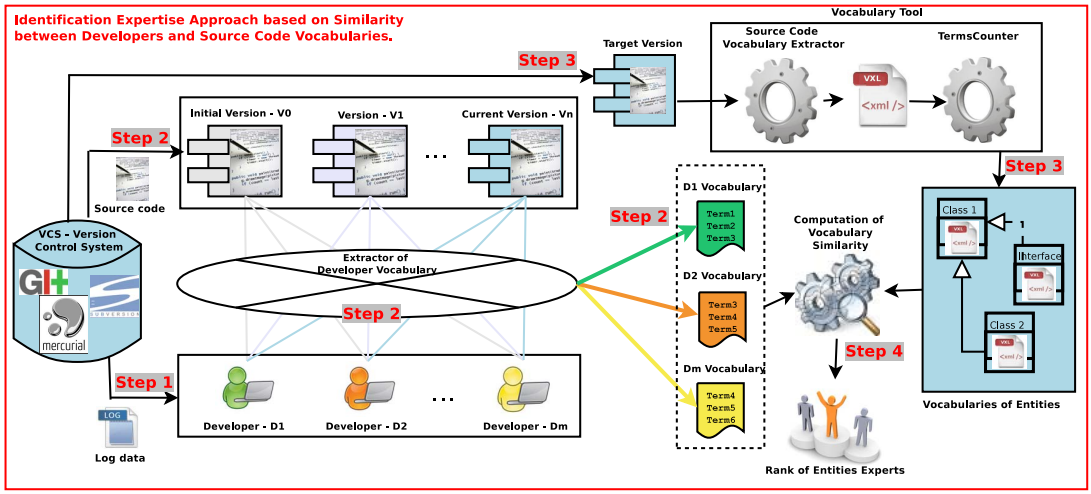

Fig. 1. Identification Expertise Approach Based on Similarity Vocabularies.

compare it with two most used approaches to identify experts. The first one is based on mining logs from *Version Control System* (*VCS*) to compute the number of commits performed by each member of development team on source code files [6], [7]. The second one also scans *VCS* logs, but defines an entity specialist according the percentage of *Lines of Code* (*%LoC*) modified by each team member [8], [9].

III. Our Approach - Based on Vocabulary Similarity

It is common for code snippets created by an author to be modified (code contributions or changes) by other developers, during projects life-cycle [5]. The vocabulary surrounding these code first must be read and understood by a developer responsible for. A change may also imply transformations on vocabulary: since an adding of term which different piece of code already has, to a replacement of all occurrences of a given term to a new one. An example for the latter is a full renaming refactory of a identifier that happens according concepts who performs the change.

Each developer has its own vocabulary that is accumulation of terms manipulated (added, removed or replaced) by a developer during its contributions on project. Each source entity also has its own vocabulary, but it differs from developers vocabularies. Because, in simplified way, a developer who will perform a change requires to understand an entity just like it is at specific target version, and not in versions before.

In the best of our knowledge the supposed similarity relationship between developers and source code vocabularies has not yet been investigated focusing on experts identification. Thus, we propose a strategy to explore this issue. Figure 1, diagrams four steps that we considered necessary to compute similarity between vocabularies, as follows:

1) extracts from *VCS* logs, all developers ID (identifiers) who have contributed to source project;
2) scans *VCS* commits extracting developers contributions. Each contribution is accumulated to its respective developer vocabulary as describe in Section II-A5;
3) extracts vocabulary of source code grained by entity for a target version[2];
4) computes the similarity among vocabularies of all target entities with vocabularies of all developers.

Whoever has the highest similarity value is considered the developer specialist of code entity.

IV. Experimental Design

A. Case Study Design

Trying to answer *RQ-1* and eventually *RQ-2,* we propose a case study that was divided into the following steps: 1) selecting Java project and sampling entities; 2) building oracle of entities experts; and 3) measuring expertise.

1) Selecting Java Project and Sampling Entities: As software projects evolve, so can change their specialists. Knowledge about expert of code spreads among developers. Factors captured by current expertise approaches are not enough to define best suited developer to perform a given task coding [19]. Beyond having access to source code, we must conduct meetings and interviews with stakeholders to construct oracles that reliably reflect who are the experts of a software project. Depending on team size and how its members are geographically distributed, building of oracle is an unaffordable cost. For this, we perform a case study on ePol[3] project. The ePol version we had access was 0.5b released on February 18, 2014. On that time developer team consisted of twelve programmers and one project leader. In terms of size, ePol has 907 Java entities (classes and interfaces) in more than 96*KLOC*, and its software vocabulary is comprised of 2751 distinct terms.

[2]source code on which a task coding will be performed.

[3]ePol is the Management Information System of Brazil Federal Police. It is under development at Software Practices Laboratory (SPLab) of Federal University of Campina Grande, Brazil.

Because we are interested on the best suited developer to realize tasks maintenance on system functionalities, we discard 436 entities whose role in ePol code is to test the other remaining 471.

2) Building Oracle of Entities Experts: It is not economically feasible to allocate a development team to answer questionnaires, discuss consensus in meetings, for a population of 471 individuals, *i.e.* entities. Therefore, we have random extracted a sample comprised of 50 entities ($n = 50$) representing more than 10% of population. Although sampling is random it follows the same entities distribution in the population according to percentiles 33º, 66ºand 100ºof three entities metrics: FanIn, LOC and Cyclomatic Complexity. It makes sample more representative. Combining each percentile for each metric, resulting in 27 ($3^{3metric}$) groups of entities. Then, each of the 471 entities belongs to one of these groups. The distribution function of ePol entities is the relative frequency of these groups for population of entities ($m = 471$).

A group G_g is defined as a tuple of entities metrics $<F_f, L_l, CC_c>$, where $f, l, c \in \{33º, 66º, 100º\}$ percentiles. The distribution function density, *dfd*, is given by:

$$dfd(G_g) = \frac{\#\{E_e\}}{m}$$

where $E_{e=\{1,...,m\}} \in G_{g=\{1,..,27\}}$ and m is the population of entities.

Due to continuous evolution of software systems associated with large amount of developers coding the same project, development teams spend some considerable time to decide who is the most suitable member to perform a maintenance task on a given code entity. Despite log information of *VCS* and of Bug Report provide useful tips, the experiential knowledge about the expertise of each developer is what really point out most appropriate developer [19]. Moreover, this experimental knowledge is spread over all team members.

To overcome the challenge of building an expert-by-entity oracle, as reliable as possible, we submit our entities sample to development team. Source code of each entity, its callers and callees were presented to all team members at same time during a predefined meeting. Voluntarily, any developer have pointed among team members out who she thought was the expert of the analyzed entity that even could be herself. Then, remaining members adhered or not the indication. In case of disputes, new indications emerged until consensus expert was reached to.

In 28 of the 50 entities, the team have took consensus and pointed for each entity at least one expert. However, there were 22 entities for which the team did not appoint any experts. These ones were discarded to compound the oracle.

3) Measuring Expertise: To process ePol vocabulary we use VocabularyTools[4] that for this study purpose we configured it to be able: to extract identifiers that name classes, interfaces, enumerations, attributes, methods, parameters, and local variables; to collect comments and javadocs texts; to split identifiers coded both in camelcase as in underscore notation style; to extract root of words written in Portuguese language; and, to discard terms which had less than 3 characters.

[4]It is an open source suite tool that is able to extract and process software vocabulary of Java projects. More details in www.softwarevocabulary.org.

The result of processing on source code vocabulary is represented by a Term-Entity matrix as we exemplified in Table I and described in Section II-A4. The vocabulary of each developer is also represented by a frequency matrix TE^D as was defined in Section II-A5.

Experts are pointed out according to similarity value between developers and entities vocabularies . For every developer D that contributes with a project P, we calculate the cosine similarity between its vocabulary with the vocabulary of each entities present in the matrix TE_P. Thus, a similarity matrix Entity-Developer of project P, ED_P, is derived and exemplified by Table II. ED_P contains i lines where each one represent an entity of target version of project, and columns j indicates the number of contributing developers.

TABLE II
SIMILARITY MATRIX FOR ENTITIES AND DEVELOPERS VOCABULARIES OF A PROJECT P, ED_P.

Entities	Developers			
	D_1	$D_{...}$	D_{j-1}	D_j
E_1	$sim(E_1, D_1)$	$sim(E_1, D_{...})$	$sim(E_1, D_{j-1})$	$sim(E_1, D_j)$
E_2	$sim(E_2, D_1)$	$sim(E_2, D_{...})$	$sim(E_2, D_{j-1})$	$sim(E_2, D_j)$
$E_{...}$	...	...	...	...
E_i	$sim(E_i, D_1)$	$sim(E_i, D_{...})$	$sim(E_i, D_{j-1})$	$sim(E_i, D_j)$

For a given entity E_i, the developer D_j who has the greatest similarity value is recommended as the expert developer to E_i. Besides, it also possible to pointed not just a single expert but the k developers who have knowledge about E_i. It makes sense in situations where the principal expert is not available, for instance.

V. RESULTS AND DISCUSSION

Developer team has performed maintenance tasks on source code of ePol versions until the target one 0.5b. We have collected developers vocabularies of four development periods: two, four, six and nine months before the day when version 0.5b of ePol were released. We compute cosine-similarity of each developer vocabulary with source code vocabulary of target ePol version. Then, we have counted the total of entities whose similarity values were greater than 0.3, 0.4 and greater than 0.5. The graphic drawn in Figure 2 summarizes the results we have achieved. For every similarity limit we identified the same behavior: on first two months regardless of entities number the similarity between vocabularies is about 2.5% greater than the next two periods: four and six months. We believe this decreasing reflects the vocabulary decay phenomenon where newest words are weighted slightly more than older ones [18]. However for the four periods we observe, in average, the percentage of entities whose vocabulary has more than 0.5 of similarity with developers vocabularies is 20.63%, for more than 0.4 and more 0.3 the percentage increases for 25.47% and 29.38% respectively.

After we have confirm similarity relationship between entities and developers vocabularies, we used our approach, Sec-

tion III, to identify entities experts. We compared it, in terms of precision an recall, with two others traditional approaches: based on number of commits, and based on *%LoC*.

To be part of a project an entity had been inserted in *VCS* at some point by a given developer. Thus, from the perspective of both based on commit and based on *%LoC* approaches every entity always has its respective expert, even it has been manipulated only once: in its creation by its author. In case of our approach, not always an entity will have an expert associated with. Depending on the developmental period has been observed none developer vocabulary will share terms with entities vocabulary for a given target version.

Since a random strategy was used to define the sample that compound the oracle, Section IV-A2, we have no guarantee that each sampling entities had been manipulated during developmental period we chose to assess our approach. Because this, we have measured the accuracy of our approach regarding the same four different periods of developers vocabularies we used to observe similarity relationship behavior. Besides, we have computed accuracy considering both just one expert by entity (*top-1*) as two (*top-2*) and three (*top-3*) experts.

The oracle we have built allow us to measure approaches accuracy, and consequently to compare them. Table III contains precision and recall values we have achieved for each experimented approach. As expected, the accuracy for based on commit and based on *%LoC* has kept same values regardless parameters *top-k* and number of months have been changed.

In case of our approach, its accuracy depend on parameters values. First, taking *top-1* expert, as more contributions are considered to constitute vocabularies of developers, better the accuracy of our approach. When developers vocabularies are comprised by just two months of contributions precision and recall values is 0.25, but accumulation of nine months of contributions produces a precision and recall of 0.3214.

Scenarios in which more than one expert is considered, in general, we find that: as the period considered to generate the vocabulary of developers increases, the accuracy of our approach is close to values reached by the two baseline approaches. In our view this trend indicates that similarity of vocabularies is a promising way to identify experts.

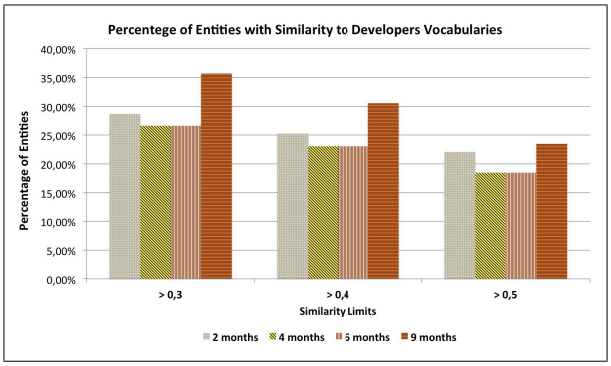

Fig. 2. Percentege of Entities with Similarity to Developers Vocabularies.

Generation of developers vocabulary does not count nor wrapping long statements lines or indentation adjustments. Disregard these types of formatting settings prevents improper expertise attribution by our approach. For example, most of development tools have text editing options to automatically adjust a source code formatting. Considering the two approaches we use as baseline, if this feature is performed and change code is committed, the expertise credit would be mistakenly attributed for whom have selected the feature on.

One other advantage of using similarity of vocabularies is its potential to recommend experts for orphaned entities of a system. In this scenario, would identify among the current team members one whose contributions on the vocabularies of other entities could quickly understand the code of orphan entity over which a given maintenance task would be executed.

A. Threats to Validity

Construct Validity: Vocabulary operations consider that identifiers are coded in camelcase or underscore notation style, and that comments and javadocs are written in Portuguese. Thus, our findings are limited by accuracy of those algorithms. The meeting with development team to build the oracle was scheduled to avoid lack of members, set its start end time to ensure its completion, and performed in presence of researchers to mediate possible conflicts among members. Besides, all meeting audio was recorded for eventually later audit of any ePol project stakeholder. However, we can not guarantee that groupthink phenomenon has not influenced on oracle result information.

External Validity: The apparently project-specific influences on the usage of software vocabularies to identify experts, suggest that, general conclusions may not be derived from our findings, although our approach seems to be promising to improve state-of-practice. Caution is necessary when applying our proposal to other projects.

VI. Related Work

Researches have stressed the importance of software vocabulary to facilitate program maintenance. Identifiers are chosen in accordance with stakeholders' personal preferences and experience [20], and related to problem domain concepts [11] to promote software understanding. When identifiers do not follow good convention rules [13] maintenance tasks are degraded.

In addition to the *VCS*, other repositories are mined to map expertise. For instance, mining *e-mails* exchanged among developers whose subject were related to code changes or bug reported indicates experts [6]. Learning machines techniques have also been used. Taking slots of bug repository information to learn patterns that indicate who should solve a kind of bug [21]. The *Degree-Of-Knowledge - DOK* model combines authority information with edition interactions performed by all others developers for point out who are the *experts* for a given source entity [5].

The aforementioned studies endorse its necessary to add other different aspects they already use to improve their accuracy. Besides, in the best of our knowledge in none of them

TABLE III
APPROACHES COMPARISON FOR *Top-k* EXPERTS CONSIDERING 9 MONTHS OF DEVELOPMENT.

Approach Based on	*top-k*	2 months		4 months		6 months		9 months	
		precision	recall	precision	recall	precision	recall	precision	recall
Commit	1	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571
	2	0.2142	0.2142	0.2142	0.2142	0.2142	0.2142	0.2142	0.2142
	3	0.1547	0.1547	0.1547	0.1547	0.1547	0.1547	0.1547	0.1547
%LoC	1	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571	0.3571
	2	0.2321	0.2321	0.2321	0.2321	0.2321	0.2321	0.2321	0.2321
	3	0.1667	0.1667	0.1667	0.1667	0.1667	0.1667	0.1667	0.1667
Cosine-Similarity (Ours)	1	0.2500	0.2500	0.2500	0.2500	0.2857	0.2857	0.3214	0.3214
	2	0.1964	0.1964	0.1607	0.1607	0.1964	0.1964	0.1964	0.1964
	3	0.1905	0.1905	0.1548	0.1548	0.1786	0.1786	0.1667	0.1667

no association between developers and software vocabularies was used to model code expertise. The closest of our study was developed by Matter and colleagues [18]. In it, the similarity between the vocabularies of developers and content of bug reports is used to assign bugs to be fixed by one developer. In our proposal, we extract the vocabulary of source entities (classes and interfaces) and through their similarity with developers vocabulary we pointed the expert who will make a change or a code review whose location is known.

VII. CONCLUSIONS AND FUTURE WORK

Literature on software vocabulary provides evidences of developers' skills, personal preferences, as well as mental map of a project, and carries design aspects not captured by traditional structural metrics. The results we have achieved in this study increases the list of vocabulary usage: identifying code experts. Based on the accuracy we found, its obvious our approach based on vocabularies similarity must be refined. For instance, considering decay factor for old terms of a vocabulary and for inactivity time of a given developer.

By itself vocabulary usage does not seams to be sufficient to produce a revolutionary technique to automatic identify code experts. However, we must point out if additional cost of vocabularies usage is affordable or not. For this, we are conducting a new study to assess whether vocabularies capture different aspects of expertise than those captured by commonly used approaches.

We are extending the concept of vocabulary for groups of entities that share terms. We intend to investigate whether similarity between entities clusters and developers vocabularies point concerns experts.

REFERENCES

[1] K. Bennett and V. Rajlich, "Software Maintenance and Evolution: A Roadmap," *ICSE '00 Proceedings of the Conference on The Future of Software Engineering*, pp. 73 – 87, 2000.

[2] R. P. L. Buse and W. R. Weimer, "Learning a Metric for Code Readability," *IEEE Transactions on Software Engineering*, vol. 36, no. 4, pp. 546–558, Jul. 2010.

[3] P. Hallam, "What Do Programmers Really Do Anyway?" in *The Microsoft Developer Network (MSDN). http://blogs.msdn.com/b/peterhal/archive/2006/01/04/509302.aspx*, 2006.

[4] J. Feigenspan, S. Apel, J. Liebig, and C. Kästner, "Exploring Software Measures to Assess Program Comprehension," in *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2011.

[5] T. Fritz, J. Ou, G. C. Murphy, and E. Murphy-Hill, "A Degree-of-Knowledge Model to Capture Source Code Familiarity," *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, p. 385, 2010.

[6] A. Mockus and J. Herbsleb, "Expertise Browser: a quantitative approach to identifying expertise," *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pp. 503–512, 2002.

[7] L. Hattori and M. Lanza, "Mining the History of Synchronous Changes to Refine Code Ownership," *Mining Software Repositories, 2009. MSR'*, pp. 141–150, 2009.

[8] T. Girba, A. Kuhn, M. Seeberger, and S. Ducasse, "How Developers Drive Software Evolution," *Proceedings of the 2005 Eighth International Workshop on Principles of Software Evolution (IWPSE05)*, 2005.

[9] F. Rahman and P. Devanbu, "Ownership, experience and defects: A fine-grained study of Authorship," *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, p. 491, 2011.

[10] D. Lawrie, H. Feild, and D. Binkley, "Quantifying identifier quality: an analysis of trends," *Empirical Software Engineering*, vol. 12, no. 4, pp. 359–388, Dec. 2006.

[11] S. L. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the Evolution of the Source Code Vocabulary," *Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on*, pp. 189–198, 2009.

[12] F. Deissenboeck and M. Pizka, "Concise and consistent naming," *Software Quality Journal*, vol. 14, no. 3, pp. 261–282, Sep. 2006.

[13] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Exploring the Influence of Identifier Names on Code Quality: an empirical study," *oro.open.ac.uk*, 2010.

[14] L. Guerrouj, "Automatic Derivation of Concepts Based on the Analysis of Source Code Identifiers," *2010 17th Working Conference on Reverse Engineering*, pp. 301–304, Oct. 2010.

[15] D. Cavalcanti, K. Santos, D. Serey, and J. Figueiredo, "Using Software Vocabulary to Rank Classes that are Probably Impacted by a Bug Report," in *1st Workshop on the Next Five Years of Text Analysis in Software Maintenace - ICSM2012*, 2012, pp. 0–4.

[16] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, Mar. 2007.

[17] K. d. F. Santos, D. D. S. Guerrero, J. C. A. de Figueiredo, and R. A. Bittencourt, "Towards a Prediction Model for Source Code Vocabulary," in *1st Workshop on the Next Five Years of Text Analysis in Software Maintenace - ICSM2012*, 2012, pp. 0–4.

[18] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," *6th IEEE International Working Conference on Mining Software Repositories*, pp. 131–140, May 2009.

[19] F. Servant, "Supporting bug investigation using history analysis," *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 754–757, Nov. 2013.

[20] S. Haiduc and A. Marcus, "On the Use of Domain Terms in Source Code," *The 16th IEEE International Conference on Program Comprehension*, vol. 0, pp. 113–122, 2008.

[21] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" *28th international conference on Software engineering - ICSE*, p. 361, 2006.