Analysing the Vocabulary to Identify Knowledge Divergence

Jan Nonnen, Paul Imhoff

University of Bonn
Computer Science III
Bonn, Germany

{nonnen, imhoffj}@cs.uni-bonn.de

Keywords: vocabulary, active vocabulary, vocabulary evolution, software evolution, history mining, program comprehension

1 Introduction

During the development of a project, words used in source code add up to a big vocabulary, which may lead to a divergent word-understanding and word-knowledge between developers. Even the drop out of a single developer may lead to a big loss of knowledge about words and their meaning. By keeping track of the active developers vocabulary one is able to identify and react upon such situations. In this work we propose a way to identify these by analysing the words contained in identifiers obtained through the commit history in a version control system.

A person's vocabulary contains all words that they have used or have knowledge about, and can be subdivided in an active and passive part. The active side are all the words that the person uses in speech or in written form. In contrast, the passive vocabulary contains all words that one knows. The content of the vocabulary constantly changes. On the one side, it increases due to knowledge gaining, e.g. by reading books, and on the other side it decreases due to forgetting words.

Using the commit history of a project one is able to compute the active vocabulary of each developer in the team and to verify if a shared team language can be identified.

This work represents an excerpt of the study presented by the authors in [6].

2 Related work

In programming not only the code structure can be used to express concepts, but also the identifiers carry semantics for a developer [2]. In [8] Deißenböck and Ratiu considered the knowledge in source code to be represented by the identifiers and the code structure.

Abebe et al. [1] studied changes of the team vocabulary by analysing two software systems. Their results indicate that the code size increases faster than the size of vocabulary. Thus developers tend to use a low word variance for identifiers.

Hindle et al. [5] analysed commit comments and used a time windowed topic analysis to locate trending topics in a project.

Grant et al. [3] compared two techniques for analysing topic evolution. Topics are collections of words that co-occur frequently in the project history. They could identify global changes in their evolution visualisation.

3 Methodology

Our current approach utilizes the Git source code management system to obtain the history of a project. For each commit we calculate the changes based on its parent. This so called *change set* consists of the time and the changed lines of code.

From the change sets we preserve only identifiers. Afterwards, every identifier is splitted into constituent words using camel case rules (e.g. 'addQueryResults' gives {add, query, results}). These words form the used vocabulary of the commit. In our current approach we do not apply normalization on the words, e.g. normalizing 'aborted' to 'abort' and we do not remove non dictionary words, e.g. 'i'. We use this to also analyse how word forms evolve over time.

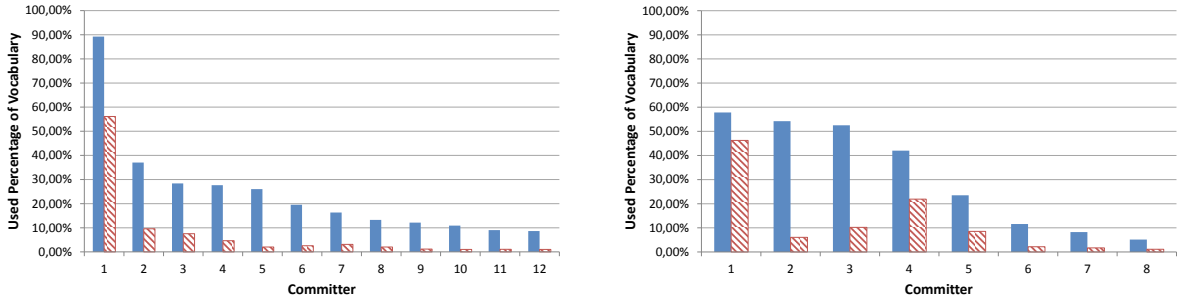
Our history model H is a set of tuples (c, a, w), where c is the commit time, a the author, and w the word. The *individual vocabulary* $V(a)$ of committer a can then be defined as $V(a) = \{w \mid \exists(c, a, w) \in H\}$.

4 Study and Results

For a study of vocabulary evolution we analysed two projects: JGit[1] and Cultivate[2]. JGit is an implementation of Git in Java and is in development since 2006. Cultivate is a static code analysis project for Eclipse that is developed at the University of Bonn and is in development since 2003.

For both projects we obtained a repository containing the commits of 25 months, see Table 1.

[1] http://www.eclipse.org/jgit/
[2] http://roots.iai.uni-bonn.de/cultivate



(a) JGit (b) Cultivate

Figure 1: Committers and their percentage of the used vocabulary (solid) in relation to the percentage of commit sizes in words (slashed).

Interestingly, the number of commits was higher in Cultivate (970) than JGit (695). Still the size of commits in JGit was bigger. Thus, the commits in Cultivate were more frequent, but also smaller. Using an observation from Hattori and Lanza [4], this means that in JGit more new features were added and the development in Cultivate focussed more on maintenance.

	JGit	Cultivate
Committers	52	15
Commits	695	970
Vocabulary Size	3391	1872

Table 1: Comparison of JGit and Cultivate between October 2009 till November 2011.

RQ: Does the knowledge of committers in both projects differ?

Figure 1 visualises how much of the total vocabulary each committer used and how many words she used in her commits. In JGit a single committer used nearly 90% of the total vocabulary, the second highest committer used only 36%. In contrast to this, in Cultivate three committers have similar vocabulary percentage of over 50%. This could indicate a single point of knowledge in the JGit project. More research is required in identifying the reasons for the disparity between both projects and whether this really represents a threat to the project health. An extension of the performed evaluation to more projects should provide a higher sample set. This may then be used to explicitly define the conditions under which a divergence occurs.

5 Conclusion and Future Work

In this work we presented the novel idea to detect knowledge divergence through analysis of the individual vocabulary usage of committers in source code. Also, we presented a study on two projects on which we evaluated our research question. We were able to identify a single point of knowledge in form of a disparity of vocabulary usage in the JGit project.

In the future we want to analyse how far program comprehension can be improved by visualising trending or stagnating words and showing each developer words that he has not used, but that all others have. These visualisations could improve the overall project comprehension and provide a knowledge improvement.

Further these words can be linked with an introduction location proposed by us in [7]. Introduction locations are locations in source code that show the meaning of a certain term. Information about "interesting" words for a developer in combination with an introduction location for each word is a seamless integration of both approaches.

References

[1] S. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol. Analyzing the Evolution of the Source Code Vocabulary. In *Proc. of the European Conference on Software Maintenance and Reengineering*, 2009.

[2] N. Anquetil and T. Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *Proc. of the Centre for Advanced Studies on Collaborative Research*, 1998. Proc. of the 1998 conference of the Centre for Advanced Studies on Collaborative Research.

[3] S. Grant, J. Cordy, and D. Skillicorn. Reverse Engineering Co-maintenance Relationships Using Conceptual Analysis of Source Code. In *Proc. of the Working Conference on Reverse Engineering*, 2011.

[4] L. Hattori and M. Lanza. On the nature of commits. In *Proc. of the International Conference on Automated Software Engineering-Workshops*, 2008.

[5] A. Hindle, M. Godfrey, and R. Holt. What's hot and what's not: Windowed developer topic analysis. In *Proc. of the International Conference on Software Maintenance*, 2009.

[6] J. Nonnen and P. Imhoff. Identifying Knowledge Divergence by Vocabulary Monitoring in Software Projects. In *Proc. of the European Conference on Software Maintenance and Reengineering*, 2012.

[7] J. Nonnen, D. Speicher, and P. Imhoff. Locating the Meaning of Terms in Source Code - Research on 'Term Introduction'. In *Proc. of the Working Conference on Reverse Engineering*, 2011.

[8] D. Ratiu and F. Deissenboeck. Programs are Knowledge Bases. In *Proc. of the International Conference on Program Comprehension*, 2006.