2010 17th Working Conference on Reverse Engineering

# Linguistic driven refactoring of source code identifiers

Laleh Mousavi Eshkevari
SOCCER Lab- Ptidej Team
École Polytechnique de Montréal, Québec, Canada
laleh.mousavi-eshkevari@polymtl.ca

***Abstract*—Identifiers are an important source of information during program understanding and maintenance. Programmers often use identifiers to build their mental models of the software artifacts. We have performed a preliminary study to examine the relation between the terms in identifiers, their spread in entities, and fault proneness. We introduced *term entropy* and *context-coverage* to measure how scattered terms are across program entities and how unrelated are the methods and attributes containing these terms. Our results showed that methods and attributes containing terms with high entropy and context-coverage are more fault-prone. We plan to build on this study by extracting linguistic information form methods and classes. Using this information, we plan to establish traceability link from domain concepts to source code, and to propose linguistic based refactoring.**

## I. Introduction

Many studies in the domain of software engineering applied object-oriented metrics, such as Chidamber and Kemerer (CK) metrics suite [1], to evaluate software quality [2], [3], maintenance efforts [4], [5], [6], [7], and to predict faults [8], [9], [10]. These metrics are calculated based on the structural data extracted from source code.

We believe that linguistic data extracted from source code might also help to improve code quality. Indeed, several studies showed the impact of identifiers on program comprehension (*e.g.*, [11], [12], [13]) and code quality [14].

Identifiers are among the most important sources of information to understand source code [11], [12]. Haiduc and Marcus [13] studied several open-source programs and found that about 40% of the domain terms were used in the source code. Unfortunately, in collaboration environments the probability of having two developers use the same identifier for different entities is between 7% and 18% [15]. Thus, naming conventions are crucial for improving the source code comprehensibility. Furthermore, applying meaningful and consistent names for source code identifiers can improve the precision of information retrieval (IR) techniques in traceability links recovery between artifacts of different types.

Identifiers must be sufficiently distinctive yet must relate to one another and to the context in which they appear [12]. We concur with Deißenböck and Pizka's observation that proper identifiers improve quality and that identifiers should be used consistently [12]. Most of the refactoring proposed in the literature aim to improve code quality via code restructuring. To the best of our knowledge, the only work on linguistic refactoring is by Caprile *et al.* [11]. They propose refactoring based on compliance of terms in identifiers with standard lexicon and grammars. Our main hypothesis is to build on this work and use the linguistic information extracted from method body for refactoring. To do so, we apply summarization techniques on method body to extract the implicit concepts implemented and use them for refactoring. We believe that linguistic refactoring can as well improve the code quality and therefore, enhance program comprehension.

The rest of the paper is organized as follows: Section II presents the preliminary study as well as the research questions. Section III describes the proposed methods to address the research question. Section IV discusses the related work, and finally Section V provides a brief summary of this work.

## II. Preliminary Results and Research Questions

In this section we explain the preliminary study and the research questions.

### A. Preliminary Results

We have performed a preliminary study [16] to investigate the impact of identifiers on fault proneness of the context in which they are applied. The subject of the study were two open source programs, ArgoUML [1]

[1] http://argouml.tigris.org/

1095-1350/10 $26.00 © 2010 IEEE
DOI 10.1109/WCRE.2010.44

(v0.16) and Rhino[2] (v1.4R3).

We have defined entropy and context based-metrics to quantify term scattering and context similarity. The identifiers found in class attributes and methods (*e.g.*, names of variables, method calls, user defined types, formal and actual parameters) were extracted and split into terms (*e.g.*, Camel-case splitting) to build the term dictionary. Next, the linguistic information was summarised into frequency matrix, *i.e.*, a term-by-entity matrix, where entry $a_{i,j}$ in the matrix denotes the number of occurrences of the $i^{th}$ term in the $j^{th}$ entity. We further normalized each entry in the matrix by dividing it over the sum of the entries in its row ($\widehat{a}_{i,j}$ is the normalization of element $a_{i,j}$ ). Term entropy is derived from entropy in information theory and measures the physical dispersion of a term in a program, *i.e.*, the higher the entropy, the more scattered across entities (methods and attribute) is the term. We computed the term entropy as:

$$H(t_i) = -\sum_{j=1}^{n}(\widehat{a}_{i,j}) \cdot log(\widehat{a}_{i,j}) \quad i = 1, 2, \ldots, m$$

With term entropy, the more scattered among entities a term is, the closer to the uniform distribution is its mass probability and, thus, the higher is its entropy. On the contrary, if a term has a high probability to appear in few entities, then its entropy value will be low.

Term context coverage measures the conceptual dispersion of the entities in which the term appears, *i.e.*, the higher the context coverage of a term, the more unrelated are the entities containing it. It is computed as:

$$CC(t_k) = 1 - \frac{1}{\binom{|C|}{2}} \sum_{\substack{i=1\ldots|C|-1 \\ j=i+1\ldots|C| \\ e_i, e_j \in C}} sim(e_i, e_j)$$

where $C = \{e_l | a_{k,p} \neq 0\}$ is the set of all entities in which term $t_k$ occurs and $sim(e_i, e_j)$ represents the textual similarity between entities $e_i$ and $e_j$. A low value of the context coverage of a term means a high similarity between the entities in which the term appears, *i.e.*, the term is used in consistent contexts.

By analyzing the distribution of the values computed for the two metrics, we have defined thresholds to categorize terms into groups of low, medium, and high entropy and context coverage.

In the preliminary study, our objective was to investigate if there is a relationship between the fault proneness of entities and the values computed for the two proposed metrics. We examined if those entities using terms with high entropy and context coverage are more likely to be fault prone. The empirical study showed that there is a statistically significant relation between attributes and methods whose terms have high entropy and high context coverage, on the one hand, and their fault proneness, on the other hand. The study is limited to two programs, ArgoUML 0.16 and Rhino 1.4R3; however, results are encouraging but replications are needed.

[2]http://www.mozilla.org/rhino/

### B. Research Questions

We consider an identifier as anomaly if it has higher frequency and-or inconsistent name. The results of the preliminary study showed that there is a relation between entities containing identifiers with anomalies and their fault proneness. This motivate us to refactor such identifiers toward names that better reflect their functionality.

Thus the research question of this paper can be summarized as: *How to pinpoint identifiers with anomalies and to provide linguistic based refactoring of such identifiers.*

This research question can be further divided to the following research questions:

- **RQ1:** How to identify concepts implemented by methods in the system?
- **RQ2:** How to define linguistic refactoring based on concepts identified?
- **RQ3:** Do the proposed linguistic refactoring strategies improve code quality?

## III. Methodology

We plan to answer the research questions defined in the Section II in the following steps:

*a) RQ1: Concept identification:* We draw inspiration from previous works on text summarization [17], [18] techniques. As explained in [18], the goal of summarization activity is to identify the main topics of a given document while minimizing the redundancy. We believe that each method in the system should implement a concept or part of a concept. Summarization techniques enable us to identify the concept which describe the main responsibility of the method. In our case, each method represents a document, while split identifiers of the method (*e.g.*, names of variables, method calls, user defined types, formal and actual parameters, comments) correspond to terms. Statements, method signature and the comments will correspond to the sentence. Therefore, a method can easily be transformed to term by sentence matrix, where each element $a_{i,j}$ in the matrix is the weighted frequency of term $i$ in sentence $j$. We would like to distinguish the terms according to the role that the corresponding identifier plays in the method. That is, to give more weights to terms coming from

identifiers that are part of formal parameters and return statement. Moreover, we plan to evaluate the precision of our concept identification technique by conducting an empirical study on a system that has proved to have good internal quality, and consistent identifier naming (*e.g.*, JHotDraw[3]), and ask experts to validate our technique. The objective would be to verify if concepts identified by our techniques match the ones identified by experts.

*b) RQ2: Linguistic-based refactoring:* Once methods are labeled with concepts, we would like to provide recommendations for refactoring. It is a general rule that a method name should reflect its responsibility. The concepts identified in previous step will enable us to evaluate the appropriateness of the method name and to provide suggestions for better naming in case the original name is not well suited. The same approach can be used for refactoring class names. Moreover, we can also identify possibility for structural refactoring. For example, by analyzing the concepts extracted from a method body we can evaluate the degree to which these concepts are related. In other words, we can evaluate the cohesion of a method and suggest extract method as a possible solution to increase the cohesion.

*c) RQ3: Evaluation of code quality improvement:* Finally, we would like to evaluate if the proposed refactoring strategies indeed increase the code quality and thus enhance the code comprehension. To verify if the comprehensibility is improved, we plan to perform an experiment and ask experts to evaluate the degree of comprehensibility before and after refactoring. Subjects will be given two fragments of code (before and after refactoring) and will be asked to identify the purpose of the code only by reading the code.

The expected contributions of this work can be summarized as following:

- Extracting domain concepts from source code, which can improve establishing traceability links between requirements and implementation,
- Refactoring method names toward semantic information which is implicit in method bodies,
- Evaluating the impact of this refactoring on program comprehension.

## IV. Related Work

Haiduc and Marcus [13] studied several open-source programs and found that about 40% of the domain terms were used in the source code. The role played by identifiers and comments on source code understandability has been empirically analyzed by Takang *et al.* [19], who compared abbreviated identifiers with full-word identifiers and uncommented code with commented code. Similar results have also been achieved by Lawrie *et al.* [20]. Recently, Binkley *et al.* [21] performed an empirical study of the impact of identifier style on code readability and showed that Camel-case identifiers allow more accurate answers. Gong and Liu [18] proposed two methods for text summarization. In both methods a given document is transformed to a term by sentence matrix. The first method which is based on IR technique, select sentences with the highest relevance score. The relevance score for each sentence is the result of inner product of the sentence vector (column of term by sentence matrix) and the whole matrix (document). The second method is applying Latent Semantic Analysis for selecting sentences for summary. Sentences with the largest index value with the most important singular value are selected for summary. The authors showed that these techniques have the same performance when their results were compared against the summarization of three human evaluators. Steinberger and Jezek [17] proposed an LSA based technique for text summarization. Moreover, they proposed a technique to evaluate the quality of the summaries. In [22] Haiduc *et al.* used text summarization technique for program comprehension.

## V. Summary

We introduced term entropy and context-coverage to measure, respectively, how rare and scattered across program entities are terms and how unrelated are the entities containing them. We provide mathematical definitions of these concepts based on terms frequency and combined them in a unique measure. We then studied empirically the measure by relating terms with high entropy and high context-coverage with the fault proneness of the entities using these terms. We used ArgoUML (v0.16) and Rhino (v1.4R3) as object programs. Results of the empirical study showed that the number of high entropy and high context coverage terms contained in a method or an attribute helps to explain the probability of it being faulty. Results are encouraging but replications are needed. We are currently applying the metric on larger software system (e.g Eclipse v2.0, v2.1, v3.0). We plan to apply summarization techniques on source code to extract domain concepts in method body. Linguistic refactoring then can be proposed by comparing these extracted concepts with the method names. We believe that naming conventions are crucial for improving the source code comprehensibility. Furthermore, applying meaningful and consistent names for source code identifiers can improve the precision of information retrieval

[3] http://www.jhotdraw.org/

(IR) techniques in traceability recovery links between artifacts of different types.

## REFERENCES


[1] S. R. Chidamber and C. F. Kemerer, "A Metrics Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, June 1994.

[2] J. Bansiya and C. G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[3] K. K. Chahal and H. Singh, "Metrics to Study Symptoms of Bad Software Designs," *SIGSOFT, Software Engineering Notes*, vol. 34, no. 1, pp. 1–4, 2009.

[4] G. Poels and G. Dedene, "Evaluating the Effect of Inheritance on the Modifiability of Object-Oriented Business Domain Models," in *Proceedings of the Fifth European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2001.

[5] L. Prechelt, B. Unger, M. Philippsen, and W. Tichy, "A Controlled Experiment on Inheritance Depth as a Cost Factor for Code Maintenance," *Journal of Systems and Software*, vol. 65, no. 2, pp. 115–126, 2003.

[6] R. Harrison, S. Counsell, and R. Nithi, "Experimental Assessment of the Effect of Inheritance on the Maintainability of Object-Oriented systems," *Journal of Systems and Software*, vol. 52, no. 2-3, pp. 173–179, 2000.

[7] M. Dagpinar and J. H. Jahnke, "Predicting Maintainability with Object-Oriented Metrics- An Empirical Comparison," in *Proceedings of the* $10^{th}$ *Working Conference on Reverse Engineering (WCRE)*. IEEE Computer Society, 2003, p. 155.

[8] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[9] H. M. Olague, L. H. Etzkorn, S. L. Messimer, and H. S.Delugach, "An Empirical Validation of Object-Oriented Class Complexity Metrics and Their Ability to Predict Error-Prone Classes in Highly Iterative, or Agile Software: A Case Study," *Journal of Software Maintenance and Evolution*, vol. 20, no. 3, pp. 171–197, 2008.

[10] S. Kanmani, V. R. Uthariaraj, V. Sankaranarayanan, and P. Thambidurai, "Object-Oriented Software Quality Prediction Using General Regression Neural Networks," *SIGSOFT, Software Engineering Notes*, vol. 29, no. 5, pp. 1–6, 2004.

[11] B. Caprile and P. Tonella, "Restructuring Program Identifier Names," in *Proceedings of the* $16^{th}$ *IEEE International Conference on Software Maintenance*. San Jose, California, USA: IEEE CS Press, 2000, pp. 97–107.

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

[13] S. Haiduc and A. Marcus, "On the Use of Domain Terms in Source Code," in *Proceedings of the* $16^{th}$ *IEEE International Conference on Program Comprehension*. IEEE CS Press, 2008, pp. 113–122.

[14] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Relating Identifier Naming Flaws and Code Quality: An Empirical Study," in *Proceedings of the* $16^{th}$ *Working Conference on Reverse Engineering*. IEEE CS Press, October 2009, pp. 31–35.

[15] G. Butler, P. Grogono, R. Shinghal, and I. Tjandra, "Retrieving Information From Data Flow Diagrams," in *Proceedings of the* $2^{nd}$ *Working Conference on Reverse Engineering*. IEEE CS Press, 1995, pp. 84–93.

[16] V. Arnaoudova, L. Eshkevari, R. Oliveto, Y.-G. Guéhéneuc, and G. Antoniol, "Physical and Conceptual Identifier Dispersion: Measures and Relation to Fault Proneness," in *Proceedings of the* $26^{th}$ *International Conference on Software Maintenance (ICSM'10) - ERA Track*. IEEE Computer Society, 2010.

[17] J. Steinberger and K. Jezek, "Text summarization and singular value decomposition," in *Proceedings of third International Conference on Advances in Information Systems (ADVIS)*, 2004, pp. 245–254.

[18] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, 2001, pp. 19–25.

[19] A. Takang, P. Grubb, and R. Macredie, "The effects of comments and identifier names on program comprehensibility: an experiential study," *Journal of Program Languages*, vol. 4, no. 3, pp. 143–167, 1996.

[20] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a name? a study of identifiers," in *Proceedings of 14th IEEE International Conference on Program Comprehension*. IEEE CS Press, 2006, pp. 3–12.

[21] D. Binkley, M. Davis, D. Lawrie, and C. Morrell, "To CamelCase or Under score," in *Proceedings of 17th IEEE International Conference on Program Comprehension*. IEEE CS Press, 2009.

[22] S. Haiduc and a. A. M. Jairo Aponte, "Supporting program comprehension with source code summarization," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010, pp. 223–226.