Relating Developers' Concepts and Artefact Vocabulary in a Financial Software Module

Tezcan Dilshener Michel Wermelinger
Center for Research in Computing, Department of Computing
The Open University
Milton Keynes, United Kingdom

Abstract—**Developers working on unfamiliar systems are challenged to accurately identify where and how high-level concepts are implemented in the source code. Without additional help, concept location can become a tedious, time-consuming and error-prone task. In this paper we study an industrial financial application for which we had access to the user guide, the source code, and some change requests. We compared the relative importance of the domain concepts, as understood by developers, in the user manual and in the source code. We also searched the code for the concepts occurring in change requests, to see if they could point developers to code to be modified. We varied the searches (using exact and stem matching, discarding stop-words, etc.) and present the precision and recall. We discuss the implication of our results for maintenance.**

Keywords-business software maintenance; domain vocabulary; change requests; empirical study

I. INTRODUCTION

Prior to performing a maintenance task, the designated developer has to become familiar with the application in concern. She has to search the application source code to identify the program elements implementing the concepts referred by a change request (CR) document. If unfamiliar with the application, she has to read the source code to understand how those program elements interact with each other to accomplish the described use case. One approach is to debug the described application flow and step through the executed code to identify the program elements called during the execution stage of the referred use case. However, such dynamic trace may not always be adequate because not all the relevant sections of the code may get executed. To compensate for the missing program elements, without program slicing tool support, she would have to perform search tasks using the terminology found in the CR document. Subsequently, she would then be in a position to correlate both results to determine how to implement the change.

During application development, it would have been ideal for program comprehension to use the same words found in the requirements documentation when declaring identifier names. However, the developers often choose the abbreviated form of the words and names found in the text documentation as well as use nouns and verbs in compound format to capture the described actions. In addition, the layered multi-tier architectural guidelines better known as Parnas' information hiding principle [1], which advocate the separation of concerns, in turn cause the concept implementations to be scattered across the application. This design principle leads to loss of information and creates challenges during maintenance when linking the application source code to the text documentation.

Also, the separation of concerns coupled with the abstract nature of OOP (Object Oriented Programming) obscures the implementation and causes additional complexity for programmers during concept location and comprehension tasks [9]. Shepherd *et al.* argue that OOP promotes the decomposition of concepts into several class files scattered across multiple layers of an application's architecture opposed to procedural programming languages where all of the implementation of a concept is usually done in one source file. Furthermore, if a program is coded by using only abbreviations and no meaningful words such as the ones from its text documentation, then searching for the vocabulary found in the supporting documentation would produce no results. According to Lawrie *et al.* [2], industrial software tends to use more abbreviations than open source code. In such circumstances, the developer, responsible for performing a change request, is now confronted with the task of comprehending the words represented by the abbreviations before being able to link them to those described in the text document.

In this paper we undertake a preliminary investigation of a commercial financial application's module to see whether vocabulary alone provides a good enough leverage for maintenance when abbreviations are less used. More precisely we are interested in comparing the vocabularies of text documentation, change requests and source code to determine whether (1) the source code identifier names properly reflect the domain concepts in developers' minds and (2) identifier names can be efficiently searched for concepts to find the relevant classes for implementing a given change request.

The rest of this paper is organized as follows: Section II describes the current research efforts related to our work,

Section III describes our work, Section IV presents our results, Section V highlights the threats to validity, Section VI discusses the results and Section VII concludes.

II. Related work

How vocabulary is distributed amongst the program elements of an application as well as recovering traceability links between source code and textual documentation has been recognised as an underestimated area [3]. The case study conducted by Lawrie *et al.* [10] investigated how usage of identifier naming styles (abbreviated, full words, single letters) assisted in program comprehension. They concluded that although full words provide better results than single letters, use of abbreviations are just as relevant and report high confidence. Additionally, the work experience and the education of the developers also play an important role. They lobby for use of standard dictionaries during information extraction from identifier names and argue that abbreviations must also be considered in this process. We investigate further in an environment where recognisable names are used, if the change request and domain concept terms result in higher quality of traceability between the source code and the text documentation.

Haiduc and Marcus [4] created a list of graph theory concepts by manually selecting them from the literature and online sources. They extracted identifier names and comments representing those concepts from the source code. They then checked if the terms extracted from the comments are identifiable in the set of terms extracted from the identifiers. In addition they measured to see the degree of lexical agreement between the terms existing in both sets. They concluded that although comments reflect more domain information, both comments and identifiers present a significant source of domain terms to aid developers in maintenance tasks. We also check whether independently elicited concepts, in our case from the financial domain, occur in identifiers, but we go further in our investigation: we compare different artefacts beyond code, and we check whether the elicited concepts can be used to map change requests to the code areas to be changed.

In that, our work is similar to the efforts of Antoniol *et al.* [5]. Their aim was to see if the source code classes could be traced back to the functional requirements. The terms from the source code were extracted by splitting the identifier names, and the terms from the documentation were extracted by normalising the text using transformation rules. They created a matrix listing the classes to be retrieved by querying the terms extracted from the text document. The method relied on vector space information retrieval and ranked the documents against a query. Applying precision and recall validated their results. Although the authors compare two different retrieval methods (vector space and probabilistic), they conclude that semi-automatically recovering traceability links between code and documentation is achievable despite the fact that the developer has to analyse a number of sources during a maintenance task to get high values of recall. Our work differs in two main ways. First, it is geared towards maintenance, because we attempt to recover traceability between change requests and source code classes, instead of between requirements and code. Second, because we improve the precision of the search by using project specific *stop-word* filtering and vocabulary mapping. Stop-words are those without any significant meaning in English, e.g. ‘a’, ‘be’, ‘do’, ‘for’.

III. Methodology

The subject of this study is an industrial web-based financial application developed using the Java programming language at our industrial partner as a proprietary application and is not publicly available. Its functionality is to calculate economical capital to evaluate operational risk. It consists of four modules that are clones of each other: they implement the same business concepts, but differ in how the calculations are performed and parameterized. It has been in production for 4 years and consists of about 2,043 artefacts including source code, configuration files, and user guide documentation.

The application has been maintained by five developers, including in the past the first author, none of them being one of the initial developers. Change requests and maintenance tasks in the form of business functionality enhancements and problem corrections are documented in a change management and source control system and performed on an on-going basis. The designated developer is responsible for obtaining the assigned task and searching for the relevant artefacts. The application is entirely developed by further extending the in-house developed frameworks. So, prior to starting the maintenance task, a developer who is new to the technical architecture and vocabulary of the application is faced with the challenge of searching the application artefacts to identify the relevant sections. In order to assist the developer, we attempt to see what clues can be obtained from the vocabulary of the application domain.

TABLE I. Change request descriptions

CR	Description
1088	Change the layout of not editable fields in the calculation mask to formatted text.
1090	Allow to edit the market values at the asset level, calculation mask with edit.
2002	Export data to an importable excel format.
2003	Pdlgd export data to an importable excel format.
2010	Allow volatility values greather than 1.
2017	The get changed values doesn't update the time base for volatilities.
2049	Out of date of the baseline calculation is not displayed in the planning overview page.
2063	New reallocation method "Use Asset Diversified Risk".
2068	Show in both sub systems Market and PD/LGD all calculation states similar to the Roundup module.
2074	Dialog to distribute lambda factors similar to other module.
2081	Show approx. group values and diversification effects.
2095	Error during risk calculation. The market values should be set to null instead of 0 until the next release.

In the first stage of our process, we obtained, for one of the modules, the complete source code, the user guide, and 12 change requests. The code comprises 80,517 LOC over 282 classes and 29 packages. The user manual is 80 pages and 25,069 words long. The change requests are those implemented most recently, for the latest production release. Change requests are very terse, as shown in Table I.

Using the source code mining tool JIM [6], we parsed the code, extracted the identifiers and split them into single terms referred to as *hard words* [7] (their component words and abbreviations). JIM automates the extraction and analysis of identifiers from Java source code files. First, the identifiers and metadata from the Java source code abstract syntax tree (AST) are extracted and added to a central store, with information about their location. Second, the tool INTT [8] within JIM is used to tokenise the identifier names by using camel case, separators (assumed to mark boundaries) and algorithms to split ambiguous boundaries, digits and lower case, but no abbreviation expansion is performed. The extracted information, the identifier names, their tokenisations and metadata, including their source code location, are stored in a Derby[1] database. Parsing the code, extracting the identifiers, splitting them, and storing all information in the database took 33 seconds on a dual core Mac Book Pro with 4Gb memory. The resulting database size is 31Mb, containing 12,020 identifiers and 30,873 hard word instances forming 677 unique hard words.

In the second stage, we saved the user guide (in Microsoft Word format) as a text file to ignore images, graphics, and tables. Confidential information, such as names, email addresses and phone numbers was then manually removed from the text. We next extracted the words from the resulting text document. For this task, we developed a simple Java application using the Lucene[2] framework to analyse and tokenise the sentences into single *terms*. We use the word 'term' because it covers non-English words, prefixes, abbreviations and business terminology [4]. We chose Lucene's *StandardAnalyzer* class because it tokenises alphanumerics, acronyms, company names, email addresses, etc. using a JFlex-based lexical grammar. It also includes stop-word removal. We used a custom stop-words list[3] to filter them out. Running our Java program over the user manual text, we obtained 697 unique terms with a total of 13,801 instances.

We applied the same process as described above to extract the terms from the change requests (CRs). We obtained 169 unique terms with 1,602 occurrences. The reason for such a high number of terms is because the CRs are forms containing fields for tracking purposes e.g. Priority, Assigned Date, Defect Id and terms are repeated in the different fields. We also compiled a list of business concepts used in this financial application domain based on our experience. The business concepts are made up of multiple words (*n*-grams), e.g. "Investment Market Risk", "Market Value Calculation", "Lambda Factors". The list was distributed as a Likert type survey with a "strongly agree"-"strongly disagree" scale amongst three other developers and a business analyst to rate each concept and to make suggestions in order to reduce any bias we might have introduced. After evaluating the survey results and consolidating the suggestions, we took the 45 unique single words (like 'market' and 'lambda') occurring in the business concepts as basis for further analysis. Henceforth, those 45 words are called *concepts* in the paper. The turn around time for the whole process was less than 3 days.

Finally, in the last stage of our process, we developed a search application in Java to read the comma separated text files containing the concepts and terms obtained in stage 2, and then run SQL queries to (1) search for occurrences of the concepts in the three artefacts (CRs, user guide and code) and (2) search for all classes that included hard words matching the concepts found in CRs. For each search, we performed exact match and then stem match to see if we obtained more accurate results. For the stem searches, the Java application in stage 2 was modified to use Lucene's *PorterStemmer* class to compute the term's or concept's stem word by removing the common and morphological endings (a.k.a. inflections). This takes lexical variations into account, e.g. all words in Table IV have stem 'calcul'. In addition, for search (2), we manually listed the classes involved by each CR in a traceability matrix based upon our development experience with the application to then compute the search's precision and recall.

Figure 1 illustrates the extraction search and analysis stages of our process. The top part represents the extraction and storing of *hard words* from the source code using the JIM tool and the lower part shows the extraction of *terms* from the user guide and CR documents. The search stage is shown in between. The search results are saved in a comma separated file and imported into the spreadsheet program for analysis.

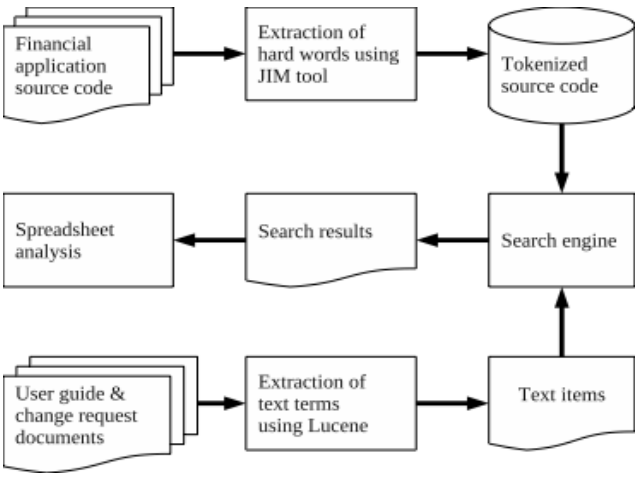

Figure 1. Search and Analysis processes.

1. http://db.apache.org/derby/
2. http://lucene.apache.org/java/docs/index.html
3. http://armandbrahaj.blog.al/2009/04/14/list-of-english-stop-words/

IV. RESULTS

Searching for exact occurrences of concepts in the artefacts, we found that while each concept occurred in at least one artefact, only 16 concepts occurred in all three artefacts. Table II shows the 16 common concepts, sorted by frequency in CRs, and their respective frequency in the other two artefacts.

TABLE II. CONCEPT SEARCH RESULTS USING EXACT TERMS

Concept (exact search)	Instances in CR	Rank CR	Instances in Guide	Rank Guide	Instances in Code	Rank Code
market	32	1	605	1	558	2
value	24	2	198	7	472	3
calculation	14	3	513	2	56	14
risk	12	4	259	4	371	4
asset	8	5	49	12	171	8
roundup	8	6	8	16	5	15
diversification	4	7	11	15	59	13
time	3	8	205	6	297	5
lambda	3	9	101	9	187	7
base	3	10	104	8	127	9
volatility	3	11	208	5	124	10
group	3	12	13	14	61	12
factors	3	13	92	11	5	16
index	2	14	322	3	661	1
unit	1	15	44	13	271	6
portfolio	1	16	100	10	86	11

Subsequently, we wanted to identify if the concepts also have the same degree of importance across artefacts, based on their occurrences. For example, among those concepts occurring both in the code and in the guide, if a concept is the *n*-th most frequent one in the code, is it also the *n*-th most frequent one in the guide? We applied Spearman's rank correlation coefficient, to determine how well the relationship between two variables in terms of the ranking within each artefactual domain can be described [11]. The correlation was computed pair-wise between artefacts, over the instances of the concepts common to both artefacts, i.e. between the CRs and the user guide, then between the CRs and the source code, and finally between the user guide and the source code. Table III shows the results using the online Wessa statistical tool[4] and the number of common concepts occurring in pairs of artefacts.

TABLE III. SPEARMAN CORRELATION FOR EXACT AND STEM SEARCH

	Exact Search / Stem Search		
Correlation Between =>	CR & Guide	CR & Code	Guide & Code
Common concepts	17	16	36
Spearman rank correlation	0.32 / 0.52	0.093 / 0.13	0.55 / 0.67
p-value	**0.19 / 0.037**	**0.72 / 0.62**	**0.0016 / 0.0002**

The correlation is low and not statistically significant (p-value > 0.05) between CRs and the other two artefacts, because there are relatively few common concepts and they have few exact occurrences in CRs. The correlation between user guide and code is much greater and statistically significant.

We searched again for concepts in the terms and hard words extracted from the artefacts, but using stemming. This did not increase the number of common concepts between artefacts. However, it changed the number of instances found, as Table IV illustrates: there are 56 exact occurrences of concept 'calculation' in the code's hard words, but searching for the concept's stem returns 29 additional instances. This changed the relative ranking of the common concepts. The Spearman correlation became stronger and statistically more relevant, as Table III shows. Only the correlation between CRs and code remains statistically not significant.

TABLE IV. STEMMING EXAMPLE

Term	Instances Guide	Instances Code
calculate	50	11
calculated	27	5
calculating	1	4
calculation	513	56
calculations	129	2
calculator	0	7

Next we identified the domain concepts each CR refers to (compare Tables I and V) and then did an exact search of those concepts among the hard words belonging to class identifiers. The retrieved classes were compared to those that should have been returned, i.e. those that were affected by implementing the CR as listed in the traceability matrix described in section III. The results for the CRs of Table I are shown in Table V.

TABLE V. SEARCH RESULTS USING EXACT CR CONCEPTS

CR	concepts searched	relevant classes	relevant retrieved	recall (%)	retrieved classes	precision (%)
1088	calculation, market	8	8	100	148	5.41
1090	calculation, asset, market	11	10	90.91	148	6.76
2002	roundup	15	0	0	0	0.00
2003	pdlgd	6	0	0	0	0.00
2010	volatility, market	6	6	100	141	4.26
2017	base, market, time	4	4	100	144	2.78
2049	calculation, market	7	7	100	148	4.73
2063	asset, index, market, risk	7	7	100	161	4.35
2068	calculation, market, diversification, holding, roundup	5	3	60	149	2.01
2074	factors, lambda	6	4	66.67	11	36.36
2081	group, roundup, diversification	6	0	0	0	0
2095	market, risk	8	8	100	161	4.97

Exact CR concept search had very high recall but very low precision. Since stemmed search returns a superset of

4. http://www.wessa.net/rankcorr.wasp

exact search, it likely deteriorates precision but it could improve recall. In fact, the precision did deteriorate as shown in Table VI, e.g. for CR #2074 it declined from 36.36% to 30.77%. The reason for this is that a stemmed term for 'factors' is 'factor', resulting in 2 additional classes to be retrieved. The stemmed search did not improve recall either, as shown in Table VI, e.g. no additional relevant classes were found for CRs #2002 and #2074.

TABLE VI. SEARCH RESULTS USING STEMMED CR CONCEPTS

CR	stemmed concepts searched	relevant classes	relevant retrieved	recall (%)	retrieved classes	precision (%)
1088	calcul, market	8	8	100	150	5.33
1090	calcul, asset, market	11	11	100	150	7.33
2002	roundup	15	0	0	0	0.00
2003	pdlgd	6	0	0	0	0.00
2010	volatil, market	6	6	100	141	4.26
2017	base, market, time	4	4	100	144	2.78
2049	calcul, market	7	7	100	150	4.67
2063	asset, index, market, risk	7	7	100	161	4.35
2068	calcul, market, diversif, holding, roundup	5	3	60	151	1.99
2074	factor, lambda	6	4	66.67	13	30.77
2081	group, roundup, diversif	6	0	0	0	0
2095	market, risk	8	8	100	161	4.97

We looked further at the reasons for low precision. In the case of CR #2002 (0% recall and precision), the request is about a generic action (exporting) on the concept (roundup), and as such the concept does not appear in the relevant class names. Other CRs involve the frequent concept 'market' (see Table II), which due to the project naming conventions occurs in almost every class name of the module, causing many false positives.

TABLE VII. SEARCH USING CR VOCABULARY, STOP-WORDS AND MAPPING

CR	vocabulary searched	relevant classes	relevant retrieved	recall (%)	retrieved classes	precision (%)
1088	calculation, helper	8	8	100	57	14.04
1090	calculation, asset, adapter, data, edit, operation, report, version, workflow	11	6	54.55	98	6.12
2002	data, export	15	15	100	45	33.33
2003	pdlgd, data, export	6	6	100	45	13.33
2010	volatility,	6	4	66.67	20	20
2063	asset, index, risk, common, method	7	5	71.43	52	9.62
2074	copy, distribute, lambda	6	6	100	69	8.70
2095	risk	8	8	100	161	13.16

To improve precision, so that developers have to inspect fewer classes for their relevance to the CR, we prepared a customized search for a subset of the CRs from Table I. First, we searched the classes' hard words using the actual words of the CR, rather than its associated concepts, because they better describe the concept's aspects or actions to be changed. However, the CR and the class identifiers may use different words. For example, the CR #1088 term 'mask' refers to the GUI, which is implemented by the Helper pattern, explicitly referred to in class names. Hence we introduced a project specific mapping mechanism, which in our case includes 'mask'→'helper'. Finally, we discarded from searches project specific stop-words, like 'market' in our case. The new results obtained are shown in Table VII. We see that in 4 out of 8 cases precision increased by 50% compared to Table V, while the impact to recall has remained minimal. To the contrary, the previously not detected classes for CR #2002 are now all retrieved.

V. THREATS TO VALIDITY

The *internal validity* addresses the relationship between the cause and the effect of the results to verify that the observed outcomes are the natural product of the implementation. A single developer (the first author) listed the concepts. This threat to internal validity was partly addressed by having the concepts validated by other stakeholders.

The *construct validity* addresses whether the conclusions can legitimately be made from the operationalization of the theories. We only used single-word concepts, while business concepts are usually compound terms. This threat to construct validity will be addressed in future work: we will see if term co-occurrence improves precision.

The *external validity* addresses the possibility of applying the study and results to other circumstances. The characteristics of this project (the domain, the terse CRs, the naming conventions, the kind of documentation available) are a threat to external validity, and we intend to repeat the experiment with other projects and artefacts, but still within the financial domain for comparison.

VI. DISCUSSION

Regarding our first aim (Section I), we note that, together, the three artefacts explicitly include all the domain concepts agreed upon by four developers and a business analyst. This indicates (a) full business concept coverage, and (b) in this project abbreviations are not required to retrieve such concepts from the artefacts. Those are two good indicators for maintenance. However, only 36/45 or 80% of concepts occur both in the code and the documentation. Since the latter is consulted during maintenance, this lack of full agreement between both artefacts, regarding the concepts in the developers' heads, may point to potential inefficiencies during maintenance. On the other hand, and using stemming to account for lexical variations, those 36 common concepts correlate well (with a high statistical significance of $p=2\cdot10^{-4}$) in terms of relative frequency, taken

as proxy for importance, i.e. the more important concepts in the user guide tend to be the more important ones in the code. This good conceptual alignment between documentation and implementation eases maintenance, especially for new developers. The weak conceptual overlap and correlation between CRs and the other two artefacts is not a major issue for us. Change requests are usually specific for a particular unit of work and may not necessarily reflect all implemented or documented concepts.

Regarding our second aim, we found that mapping a CR's wording to domain concepts and using those to search for classes to be changed, is enough to achieve very good recall, but precision is poor. We found both recall and precision can be improved by (a) using the actual CR vocabulary, (b) mapping some of it to different terms used in class identifiers and (c) ignoring frequent concepts, which act as stop-words. We note that such project specific, simple, and efficient techniques can drastically reduce the false positives a developer has to go trough to find the classes affected by a CR. We also note that in projects like this, where class identifiers are more descriptive than abbreviations, the use of stem search is useless, as it decreases precision, while not increasing recall.

VII. CONCLUDING REMARKS

This paper presents an efficient approach to relate the vocabulary of information sources for maintenance: change requests, code, documentation, and the concepts in the stakeholders' minds. The approach consists in first extracting and normalising (incl. splitting identifiers and removing stop words) the terms from the artefacts, while independently eliciting domain concepts from the stakeholders. Secondly, by doing exact and stemmed searches – to account for lexical variations – of the concepts within the terms extracted from artefacts, one can check whether (a) the artefacts explicitly reflect the stakeholders' concepts and (b) pairs of artefacts have good conceptual alignment. Both characteristics help maintenance, e.g. (b) facilitates locating code affected by given CRs.

The importance of descriptive and consistent identifiers for program comprehension, and hence software maintenance, has been extensively argued for in the academic [3] and professional literature [12]. We applied the approach to industrial code that follows good naming conventions, in order to investigate whether they could be leveraged during maintenance. We observed that the conceptual alignment between documentation and code could be improved, and that descriptive identifiers support high recall of classes affected by CRs, but precision is low, which is detrimental on maintenance. We found simple techniques to improve precision, but further research is needed. For example, the use of project specific stop-word filtering, as well as project specific vocabulary mapping between concept and class identifiers requires manual effort. However, the mappings and stop-words can be added incrementally as developers refine their searches, or automatic heuristics (like looking for very frequent words) could be developed.

Although this work is only a preliminary exploration of the vocabulary relationships between artefacts and the developers' concepts, it highlights that better programming guidelines and tool support are needed beyond enforcing naming conventions within code, because that by itself doesn't guarantee a good traceability between the concepts and the artefacts, which would greatly help maintenance tasks and communication within the team.

ACKNOWLEDGMENTS

We thank Simon Butler for his assistance in using the JIM tool, and our industrial partner, a global financial IT solutions provider located in southern Germany, for providing the artefacts and their input on diverse information required. Also, we are grateful to the ICSM'11 reviewers for their constructive comments and suggestions.

REFERENCES

[1] D. Parnas, "On Criteria To Be Used in Decomposing Systems Into Modules," *Communications of the ACM*, 14(1):221-227, April 1972.

[2] D. Lawrie, H. Feild, D. Binkley, "Quantifying identifier quality: an analysis of trends," *Empirical Software Eng.* 12:359-388, Feb. 2007.

[3] F. Deissenböck and M. Pizka, "Concise and consistent naming," in *Proc. 13th Int'l Workshop on Program Comprehension*, 2005, pp. 97– 106.

[4] S. Haiduc and A. Marcus, "On the Use of Domain Terms in Source Code," *16th Int'l Conf. on Program Comprehension*, 2008, pp. 113-122.

[5] G. Antoniol, G. Canfora, G. Casazza, A.D. Lucia, and E. Merlo, "Recovering traceability links between code and documentation," *IEEE Transactions on Software Engineering*, 28:970-983, 2002.

[6] S. Butler, M. Wermelinger, Y. Yu, H. Sharp, "Exploring the influence of identifier names on code quality: an empirical study," in *14th European Conf. on Software Maintenance and Reeng.*, 2010, pp. 159–168.

[7] H. Feild, D. Lawrie, D. Binkley, "An Empirical Comparison of Techniques For Extracting Concept Abbreviations from Identifiers" *Proc. Int'l Conf. on Software Engineering and Applications*, 2006

[8] S. Butler, M. Wermelinger, Y. Yu, and H. Sharp, "Improving the tokenisation of identifier names," in *Proc. European Conf. on Object- Oriented Programming*, LNCS 6813, Springer-Verlag, 2011, pp. 130-154

[9] D. Shepherd, L. Pollock, and k. Vijay-Shanker, "Towards Supporting On-Demand Virtual Remodularization Using Program Graphs", AOSD 2006, ACM, pp. 3-14.

[10] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What is in a Name? A Study of Identifiers," *Proc. 14th Int'l Conf. on Program Comprehension*, IEEE, , 2006, pp. 3-12.

[11] S. Boslaugh, P. Watters, "Statistics in a nutshell", O'Reilly, 2008, pp. 176-179.

[12] R. C. Martin, "Clean Code: A Handbook of Agile Software Crafttsmanship", Prentice Hall, 2008, pp. 17-30.