

Toward a Metrics Suite for Source Code Lexicons

Lauren R. Biggers, Brian P. Eddy, Nicholas A. Kraft
Department of Computer Science
The University of Alabama
Tuscaloosa, Alabama, USA
{lbiggers, nkraft}@cs.ua.edu; bpeddy@crimson.ua.edu

Letha H. Etzkorn
Department of Computer Science
The University of Alabama in Huntsville
Huntsville, Alabama, USA
etzkorl@uah.edu

Abstract—In this paper we present an empirical study of relationships between three source code lexicons: the identifier, comment, and literal lexicons. We conjecture that shared and unique properties of these lexicons for the given subject system can inform the configuration of a source code retrieval technique for a particular software understanding activity or software evolution task. Thus, we seek to discover these lexicon properties, and so we investigate five lexicon measures that consider term frequency, term density, and term provenance.

Keywords—software lexicon; text retrieval; software metrics

I. INTRODUCTION

Automation of software evolution tasks that require software understanding is key. Source code retrieval techniques apply text retrieval models to corpora built from text embedded in source code, and these techniques are highly configurable. For example, when extracting text from source code, we must decide whether to extract terms from comments in addition to terms from identifiers (e.g., method names) [1].

Kawaguchi et al. [2] exclude comments because they “cannot account for amount or quality across projects.” Yet, Liu et al. [3] report that “a significant amount of domain knowledge is embedded in the comments and identifiers,” and Haiduc and Marcus [4] find that “comments are a richer source of domain terms than identifiers.” Vinz and Etzkorn [5], [6] find that most comments define or provide an operational description of the software and that more than 88% are at a medium/high abstraction level.

Few empirical studies of source code retrieval techniques address the decision to extract comments in addition to identifiers, and the results are mixed. For example, Marcus and Poshyvanyk [7] include comments and report no change in results for their study of conceptual cohesion of classes, but Abadi et al. [8] exclude comments and report a performance decrease for their study of traceability link recovery.

Given the conflicting literature, our conjecture is: when using source code retrieval techniques it is beneficial to extract comments in some cases but ineffectual (or even disadvantageous) in other cases. Thus, our goal is to discover a suite of metrics that can help decide which text to extract from source code to configure a source code retrieval technique for a particular software system and usage scenario.

A. Terminology

We adopt terminology similar to that of Abebe et al. [9]:

- *Term*: a seq. of letters and the basic unit in a lexicon
- *Token*: a seq. of chars that comprises one or more terms
- *Entity*: a source element such as a class or method
- *Identifier*: a token representing the name of an entity
- *Comment*: a sequence of tokens
- *String Literal*: a sequence of tokens
- *Word*: the smallest free form in a language

A term is one of: word, abbreviation of a word, contraction of one or two words, acronym of a series of words.

B. Software Lexicons

A software lexicon is the set of terms used by a particular software system and is denoted by $L = \{t_1, t_2, \dots, t_n\}$, where $|L| = n$. Biggerstaff et al. [10] propose a conceptual decomposition in which $L = L_H \cup L_P$, where L_H is the set of human (domain) terms, and L_P is the set of programming (implementation) terms. Another possibility is an operational decomposition such as $L = L_{Doc} \cup L_{Src}$, where L_{Doc} is the set of documentation terms, and L_{Src} is the set of source code terms. Haiduc and Marcus [4] use the multidimensional decomposition $L_{Src} = L_{SrcH} \cup L_{SrcP}$ in their study.

In this paper we focus on $L_{Src} = L_{Id} \cup L_{Co} \cup L_{Li}$, where L_{Id} is the set of identifier terms, L_{Co} is the set of comment terms, and L_{Li} is the set of (string) literal terms.

C. Source Code Retrieval

In source code retrieval, a text retrieval (TR) method indexes a document corpus that is extracted from the text embedded in the source code for a software system. A user queries the resulting TR model, and a query engine uses a classifier to compare the query to each of the indexed documents and to compute a ranking of those documents. Studies support the efficacy of this process in aiding tasks such as feature location [3] and traceability link recovery [8].

II. EMPIRICAL STUDY

In this section we describe an empirical study aimed at exploring measures for quantifying relationships between source code lexicons and at analyzing how the measures can inform configuration of source code retrieval techniques.

A. Definition and Context

Our primary goal is to compare identifier, comment, and literal lexicons and to determine whether the measured relationships can help to discriminate among software systems. That is, we wish to establish whether the measures reveal variations among software systems with regard to the three lexicons.

1) *Measures for quantifying relationships between source code lexicons:* We compute a total of five measures in the study. We use the Jaccard index (Eq. 1), to measure overlap between the sets of unique identifiers and unique comments at the system level of granularity.

$$J(L_{Id}, L_{Co}) = \frac{|L_{Id} \cap L_{Co}|}{|L_{Id} \cup L_{Co}|} \quad (1)$$

Like Binkley et al. [11] we use cosine similarity (Eq. 2), to measure pairwise similarity between term-frequency vectors for identifiers, comments, and literals at the system level of granularity.

$$\cos(\theta) = \frac{V_i \cdot V_j}{\|V_i\| \|V_j\|} \quad (2)$$

where V_i and V_j are distinct vectors in $\{V_{Id}, V_{Co}, V_{Li}\}$.

We compute the three remaining measures for identifiers, comments, and literals. These measures are adaptations of the traditional comment density measure:

$$CD = \frac{CLOC}{CLOC + SLOC} \quad (3)$$

where CLOC is the number of comment lines of comment and SLOC is the number of source lines of code. Our measures use term counts rather than line counts.

The first density measure is LEXICON DENSITY (Eq. 4). For each lexicon we compute the percentage of the terms in the source lexicon that appear in the lexicon (i.e., the amount of the source lexicon that a lexicon contains).

$$LD(L_i) = \frac{|L_i|}{|L_{Src}|} \quad (4)$$

where $L_i \in \{L_{Id}, L_{Co}, L_{Li}\}$. The second density measure is UNIQUE TERM DENSITY (Eq. 5). For each lexicon we compute the percentage of the lexicon's terms that are unique to the lexicon (i.e., the amount of unique information that a lexicon contains).

$$UTD(L_i) = \frac{|L_i - (L_j \cup L_k)|}{|L_i|} \quad (5)$$

where L_i , L_j , and L_k are distinct lexicons in $\{L_{Id}, L_{Co}, L_{Li}\}$.

The remaining measure is UNIQUE TERM CONTRIBUTION (Eq. 7). To compute this measure, we need the set of terms in the source lexicon that appear in exactly one of $\{L_{Id}, L_{Co}, L_{Li}\}$. We compute this set, U_{Src} , using the symmetric difference operator:

$$U_{Src} = L_{Id} \triangle L_{Co} \triangle L_{Li} \quad (6)$$

Next, for each lexicon we compute the percentage of the terms in U_{Src} that appear in the lexicon (i.e., the amount of U_{Src} that a lexicon contributes).

$$UTC(L_i) = \frac{|L_i - (L_j \cup L_k)|}{|U_{Src}|} \quad (7)$$

where L_i , L_j , and L_k are distinct lexicons in $\{L_{Id}, L_{Co}, L_{Li}\}$.

2) *Subject software systems:* The context of the study spans 125 open source Java systems. We retrieved 117 systems from four source code repositories^{1,2,3,4} and the remaining 8 systems directly from project homepages.

Table I lists summary statistics for the 125 systems that are the subjects of our study. The complete list of systems, including individual summary statistics, is available online⁵. Of the 125 systems, we have chosen 36 (ASF, EP, PH) specifically for their similarities to systems developed in industry. For example, 28 of these systems (ASF, EP) are maintained by multiple developers who are obligated to follow coding standards set forth by project organizers. Further, the source code for each of the 36 systems is stored in a change management system (e.g., CVS), and issue reports for many of these systems are stored in an issue tracking system (e.g., Bugzilla). We have randomly chosen the remaining 89 systems (GC, SF) from open source project hosting services. Many of the systems, including all eight PH systems, have been subjects of multiple previous studies in program comprehension or software evolution.

B. Research Questions

We address the following research questions within the context of our study:

RQ1: How similar are the three lexicons?

RQ2: How much unique information do each of the lexicons contain or contribute?

Please see our technical report⁶ for details of the study settings and of the measures and analyses used.

C. Empirical Results

1) *RQ1 — Similarity measures:* We computed the correlation between the system level Jaccard index values and the system level cosine similarity values. We used Spearman's rank correlation coefficient because neither sample is normally distributed according to the Anderson-Darling test with $\alpha = .05$. The computed Spearman correlation coefficient is not significantly different from zero, meaning the two samples are not correlated.

¹ <http://apache.org>

² <http://eclipse.org>

³ <http://code.google.com/hosting>

⁴ <http://sourceforge.net>

⁵ <http://software.eng.ua.edu/data/icsm11era-lexicon-metrics>

⁶ <http://software.eng.ua.edu/reports/SERG-2011-02>

Table I
SUMMARY STATISTICS

ID	Repository	Systems	Files	SLOC	CLOC	Classes	Methods	Unique Terms
ASF	Apache Software Foundation	20	20,003	2,432,269	1,433,021	28,492	225,223	347,496
EP	Eclipse Projects	8	4,723	846,878	508,802	6,519	76,496	101,869
GC	Google Code	8	3,181	319,493	162,139	5,446	31,698	54,119
PH	Project homepages	8	5,657	665,509	337,643	9,477	61,919	78,859
SF	SourceForge	81	41,222	4,574,840	2,625,325	52,723	461,339	531,003
All		125	74,786	8,838,989	5,066,930	102,663	856,675	1,113,346

We observed that for system level identifier/comment Jaccard index, values generally are grouped tightly and are consistent across the 125 systems. In contrast, for system level identifier/comment cosine similarity, values generally cover a range about twice as large as that of system level Jaccard index values. This indicates that for our sample, lexicons (term sets) are less useful for discriminating between systems than are term-frequency vectors.

We also observed that for the industry-like systems (ASF, EP, and PH), cosine similarity values for the identifier and comment lexicons seem to be larger than for the systems selected randomly from open source project hosting services (GC and SF). So, we divided the systems into two groups, IND (ASF, EP, and PH) and OSS (GC and SF). Median cosine similarities in groups IND and OSS are 0.5590 and 0.4243, respectively. We compared IND and OSS using a Wilcoxon rank-sum test, and the results indicate that the distributions differ significantly ($W = 809, n_1 = 36, n_2 = 89, p < 0.001$ two-tailed) with $\alpha = .05$.

Larger values for system level identifier/comment cosine similarity suggest more consistent term use across the two lexicons. That is, larger values suggest that more terms are shared by the identifier and comment lexicons and that the terms used most frequently within identifiers and comments are shared by both lexicons.

As with the identifier and comment lexicons, we observed that for the industry-like systems (ASF, EP, and PH), cosine similarity values for the identifier and literal lexicons seem to be larger than for the systems selected randomly from open source project hosting services (GC and SF). So, we divided the systems into two groups, IND (ASF, EP, and PH) and OSS (GC and SF). Median cosine similarities in groups IND and OSS are 0.5763 and 0.4944, respectively. We compared IND and OSS using a Wilcoxon rank-sum test, and the results indicate that the distributions differ significantly ($W = 1072, n_1 = 36, n_2 = 89, p < 0.005$ two-tailed) with $\alpha = .05$.

2) *RQ2 — Density measures:* LD measures (as a percentage) how many of the unique terms in the source code lexicon appear in a lexicon. Our results indicate that about 75% of the terms in a system’s source code lexicon appear in the system’s identifier lexicon, whereas only about 40%

of the terms in a system’s source code lexicon appear in the system’s comment lexicon. Our results also indicate that literals in the EP subject systems serve different roles than do literals in the other subject systems. In particular, the results indicate that literals are expressed using tiny subsets of the overall source code lexicons of the EP subject systems. We also computed the correlation between the comment density and $LD(L_{Co})$ measures. The computed Spearman correlation coefficient ($\rho = 0.3854, p < 0.001$) indicates that these two samples are moderately correlated.

UTD measures (as a percentage) how many of the terms in a lexicon appear only in the lexicon. Our results highlight the consistency of the $UTD(L_{Id})$ measure across all 125 subject systems, whereas there is much variation among the 125 subject systems with regard to the unique content of the comment and literal lexicons.

UTC measures (as a percentage) how many of the unique terms in the source code lexicon are contributed by a lexicon. For the subject systems, the identifier lexicon is the primary source of unique terms and the literal lexicon generally contributes few unique terms. However, the similarity between the comment lexicon values for ASF and SF is somewhat surprising, as is the low minimum value for each of those repositories. A low value for $UTC(L_{Co})$ indicates that comments introduce few new terms into the source code lexicon, possibly indicating that they are expressed at a low level of abstraction.

We computed Spearman correlation coefficients between the LD , UTD , and UTC metrics. The results indicate strong correlations between $UTD(L_{Co})$ and $UTC(L_{Co})$ and between $UTD(L_{Li})$ and $UTC(L_{Li})$. Interestingly, the corresponding correlation for identifiers (i.e., between $UTD(L_{Id})$ and $UTC(L_{Id})$) is only moderate.

D. Threats to Validity

We controlled for threats to internal validity by testing our tool chain and by assessing the quality of our data. Because we applied the same tool chain to all subject systems, any errors are systematic and are unlikely to affect our results substantially. Additional threats to internal validity are due to the preprocessing steps we applied to textual information extracted from source code. For example, applying a stemmer

can cause terms with different meanings to be mapped to the same stem, thus causing overfitting of the data [4]. Applying a more advanced stemmer may mitigate this issue. However, we believe that such a change is unlikely to affect our results substantially. Finally, the subjects of our study comprise 125 open source Java systems, so we cannot generalize our results to systems implemented in other languages.

III. RELATED WORK

Other than our preliminary study of six Java systems [12], we are aware of few previous works on source code lexicon metrics and relationships between source code lexicons. However, properties of source code lexicons have been studied in the context of program comprehension [4], [5], [13] and quality assessment [9], [11], [14]. These works are closely related to our own in that we are interested in how comments/literals affect models built to address these issues.

Abebe et al. [13] note that the comment vocabulary tends not to reflect changes to the name vocabularies and that frequent terms are associated with domain concepts. Haiduc and Marcus [4] report that across their test cases, 23% of the domain terms in the source code are unique to the comment lexicon, but only 11% of the domain terms are unique to the identifier lexicon. Vinz and Etzkorn [5] report that > 88% of comments are at a medium to high abstraction level, and thus are likely to provide useful information about the software.

Abebe et al. [9] detect five kinds of bad lexical smells and note that views on what constitutes a lexical bad smell may differ between developers. Arnaoudova et al. [14] use statistical analyses to show that the number of high entropy and high context coverage terms in a method helps to explain the method's fault proneness. Binkley et al. [11] use cosine similarity between L_{Id} and L_{Co} to detect faults.

IV. CONCLUSIONS AND FUTURE WORK

With regard to similarity between lexicons, we found two interesting results. First, Jaccard index values, which are computed using term sets, varied little across the 125 subject systems, whereas cosine similarity values, which are computed using term frequency vectors, varied much. Second, system level identifier/comment and identifier/literal cosine similarity values were significantly larger for the industry-like systems than for the randomly selected open-source systems. These three results indicate that system level cosine similarity values can help to discriminate between software systems and thus could help to decide which text to extract when configuring a source code retrieval technique for a particular usage scenario.

With regard to unique information contained or contributed by each lexicon, the data reveal several interesting results. The most surprising finding was that the UTD values for identifiers (and to a lesser extent, comments) were consistent across all 125 subject systems. UTD measures how much of a lexicon appears only in that lexicon, and

we expected that the values for this measure would vary roughly the same amount as the UTC values. However, unlike for comments and literals, the UTD and UTC values for identifiers are not even strongly correlated.

The results of our empirical study provide the basis for a number of future studies. We would like to study whether the measures in this paper can help to inform the configuration of source code retrieval based techniques such as the PROMISIR approach to feature location. One such study would involve varying the text extraction phase (e.g., creating the corpus using identifiers only, comments only, literals only, identifiers plus comments, etc.) and building a statistical model to determine whether our measures help to explain differences in performance when using the various corpora. Further, we would like to aggregate our measures with other source code lexicon metrics, such as identifier dispersion [14] and lexicon bad smells [9], and to study whether such aggregate measures can predict software quality characteristics such as fault proneness.

REFERENCES

- [1] A. Marcus and T. Menzies, "Software is data too," in *Proc. of the FSE/SDP Wksp. on Future of Software Engineering Research*, 2010.
- [2] S. Kawaguchi, P. Garg, M. Matsushita, and K. Inoue, "MUDABlue: An automatic categorization system for open source repositories," *Journal of Systems and Software*, vol. 79, no. 7, pp. 939–953, 2006.
- [3] D. Liu, A. Marcus, D. Poshyvanyk, and V. Rajlich, "Feature location via information retrieval based filtering of a single scenario execution trace," in *Proc. of the 22nd Int'l Conf. on Automated Software Engineering*, 2007, pp. 234–243.
- [4] S. Haiduc and A. Marcus, "On the use of domain terms in source code," in *Proc. of the Int'l Conf. on Program Comprehension*, 2008.
- [5] B. Vinz and L. Etzkorn, "A synergistic approach to program comprehension," in *Proc. of the 14th IEEE Int'l Conf. on Program Comprehension*, 2006, pp. 69–73.
- [6] —, "Comments as a sublanguage: A study of comment grammar and purpose," in *Proc. of the Int'l Conf. on Software Engineering Research and Practice*, 2008, pp. 17–23.
- [7] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in *Proc. of the 21st IEEE Int'l Conf. on Software Maintenance*, 2005.
- [8] A. Abadi, M. Nisenson, and Y. Simionovici, "A traceability technique for specifications," in *Proc. of the 16th IEEE Int'l Conf. on Program Comprehension*, 2008, pp. 103–112.
- [9] S. Abebe, S. Haiduc, P. Tonella, and A. Marcus, "Lexicon bad smells in software," in *Proc. of the 16th Working Conf. on Reverse Engineering*, 2009, pp. 95–99.
- [10] T. Biggerstaff, B. Mitbender, and D. Webster, "The concept assignment problem in program understanding," in *Proc. of the Int'l Conf. on Software Engineering*, 1993, pp. 482–498.
- [11] D. Binkley, H. Feild, D. Lawrie, and M. Pighin, "Increasing diversity: Natural language measures for software fault prediction," *Journal of Systems and Software*, vol. 82, no. 11, 2009.
- [12] L. Biggers and N. Kraft, "Quantifying the similarities between source code lexicons," in *Proc. of the 49th ACM Southeast Conf.*, 2011.
- [13] S. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the evolution of the source code vocabulary," in *Proc. of the 13th European Conf. on Software Maintenance and Reengineering*, 2009, pp. 189–198.
- [14] V. Arnaoudova, L. Eshkevari, R. Oliveto, Y.-G. G. andhéandneuc, and G. Antoniol, "Physical and conceptual identifier dispersion: Measures and relation to fault proneness," in *Proc. of IEEE Int'l Conf. on Software Maintenance*, 2010, pp. 1–5.