Abstract—Tool support is crucial to modern software development, evolution, and maintenance. Early tools reused the static analysis performed by the compiler. These were followed by dynamic analysis tools and more recently tools that exploit natural language. This later class has the advantage that it can incorporate not only the code, but artifacts from all phases of software construction and its subsequent evolution.

Unfortunately, the natural language found in source code often uses a vocabulary different from that used in other software artifacts and thus increases the vocabulary mismatch problem. This problem exists because many natural-language tools imported from Information Retrieval (IR) and Natural Language Processing (NLP) implicitly assume the use of a single natural language vocabulary. Vocabulary normalization, which goes well beyond simple identifier splitting, brings the vocabulary of the source into line with other artifacts. Consequently, it is expected to improve the performance of existing and future IR and NLP based tools. As a case study, an experiment with an LSI-based feature locator is replicated. Normalization universally improves performance. For the tersest queries, this improvement is over 180% ($p < 0.0001$).

Keywords—vocabulary normalization; information retrieval; concept location

I. INTRODUCTION

Modern software evolution requires significant tool support. Effective tools exploit a variety of information and techniques to aid a software maintainer. One area of recent interest in tool development exploits the natural-language information found in source code. Such Information Retrieval (IR) and Natural Language Processing (NLP) based tools, which compliment traditional static and dynamic analysis tools, have been used to tackle problems that previously required considerable human effort such as (re)establishing links between a program and its documentation [1], assessing program quality [14], and performing concept location [15]. This paper takes as a representative example a tool that performs one of the most frequent and manually time-consuming software maintenance activities: concept location [4], the activity of identifying the location in the source code of a desired functionality.

To reap the full benefit of IR-based and NLP-based techniques, the language used across all software artifacts (e.g., requirement and design documents, test plans, as well as the source code) must be the same. This is because when retrieving documents using a search engine, having an accurate estimate of the importance of a query word is paramount to achieving good performance. In particular, if the importance of a query word is underestimated for a particular document, that document can be buried among the non-relevant documents.

Unfortunately, when applied to source code, this under-estimation is a great concern because programs contain a significant proportion of invented vocabulary in the form of abbreviations and acronyms. In effect, source code uses a different language than other software artifacts [8]. The negative impact of this vocabulary mismatch stems from the implicit assumption of IR and NLP techniques that the same words are used whenever a particular concept is described [12].

A recently proposed algorithm for performing vocabulary normalization [10] brings the vocabulary of the source code in line with that of a project’s other artifacts, making it more appropriate for use with IR-based tools (in the remainder of the paper references to IR implicitly include NLP-based tools as well). While a simplification, the algorithm aligns the vocabulary using two steps: breaking identifiers up into parts and then expanding any parts that are abbreviations or acronyms to full words. For example, average-length is split into avg-len and then expanded to average-length. The first step, identifier splitting, has received significant attention [6]. However, prior empirical study has indicated that splitting alone can be insufficient [3] (a result confirmed in Section III). Normalization’s more challenging second step, expansion, has received less attention [9], [7], [11].

Existing studies evaluate normalization by comparing normalized identifiers against a human created oracle. While this is an important validation of any technique, the ultimate goal is to improve IR-based tools. To date there has been insufficient empirical study of normalization’s impact on IR-based tools. That is, comparing tool performance on the original source with its performance on source having normalized identifiers. In theory the replacement should improve the performance of (existing and future) IR tools as such tools will be able to better estimate each document’s important vocabulary. This paper presents the first
such study of Normalize, a recently presented normalization
algorithm [10] in Sections II and III.

II. STUDY DESIGN

To avoid experimenter bias, it is important to replicate an
existing study, which avoids, for example, the possibility
that in a newly-devised study, queries favor the normalized
source code. The particular experiment chosen applies
Latent Semantic Indexing (LSI) to the problem of concept
location [13]. Hereafter this study, performed by Marcus et
al., is referred to as the original study.

The investigation of vocabulary normalization includes
two research questions. Research Question 1 considers
the impact of applying Normalize. While a secondary concern,
Research Question 2 considers the impact of the expansion
step by comparing full normalization with the impact of
aggressive splitting alone.

As a preliminary step it is necessary to replicate the
conditions of the original study as closely as possible. The
replication provides a baseline for measuring the impact
of normalization. This section lays out the necessary pieces
for replication including the preparation of the corpus (including
the identification of relevant documents), forming the query
set, and choosing appropriate performance metrics.

A. Preparation of the corpus

Classically, IR attempts to retrieve documents relevant
to a query from a (large) collection of documents. When
applying such an approach to concept location, the source
code is first broken into chunks that form the IR system's
documents. These documents are then indexed creating a
corpus. As in the original study, the corpus is built from the
source code for NCSA Mosaic.

In the replication three corpora are used. The first conser-
vatively splits identifier [5]. This is the corpus used in
the original study. To form the second the representative
aggressive splitter GenTest [11] is used, and to form the final
corpus identifiers are split and expanded using Normalize.

To evaluate tool performance, the corpus also includes
a set of relevant documents. In this case there are seven
relevant documents identified through manual inspection
of the Mosaic source code by the original study’s authors [13].
While there exist alternative test sets with more relevant
documents and alternate queries, these sets (e.g., Semeru [4])
include automatically mined relevance data, which con-
founds the impact of normalization with the imprecision
of non-human relevance judgments where non-relevant
documents can be marked as relevant. Creating good relevance
judgments (e.g., the TREC datasets) is expensive and very
time consuming.

B. Query Sets

The original study made use of two sets of queries:
user generated and automatically generated queries. For the
user-generated queries, Marcus et al. asked six members
of their research lab to formulate a query that describes

font properties motivated by a change request that includes
“Add a new font to Mosaic.” This set was augmented by
two additional queries: the word “font” all by itself and the
union of the words found in the six user generated queries.
Table I lists each of the eight queries in this set, which will
be referred to as the Original Set.

The automatically generated query set was constructed by
querying the LSI space for a ranked list of the top 40 terms
most similar to the term “font”. Using this list, 40 queries
were generated using the formula

\[ \text{query}(n) = \text{“font”} + \text{the top n ranked terms from the list}. \]

In addition to the two query sets from the original study,
two additional sets are used in this study. The motivation for
this addition is that the first two query sets are dominated by
unnaturally long queries compared to typical web queries.
Thus two sets of shorter queries, shown in Table II, are also
considered. The first, pairs, includes ten queries composed of
the word “font” paired with each of the ten unique words
from the original set. The second, triples includes 45 queries
composed of the word “font” and two of the ten unique
words.

C. Performance Metrics

While precision and recall are appropriate performance
metrics for unranked retrieval, for evaluating ranked retrieval
Average Precision (AP) and Mean Average Precision (MAP)
are the appropriate measures [12]. These two should be used
for ranked retrieval because precision and recall do not take
the ranks into account. Consider, for instance, the top ten
documents in a ranked list, precision and recall remain the

<table>
<thead>
<tr>
<th># Query</th>
<th># Query</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 font</td>
<td>1 font size style</td>
</tr>
<tr>
<td>2 font size style</td>
<td>2 font size small</td>
</tr>
<tr>
<td>3 font style small</td>
<td>3 font size regular</td>
</tr>
<tr>
<td>4 font style</td>
<td>4 font size large</td>
</tr>
<tr>
<td>5 font large</td>
<td>5 font size family</td>
</tr>
<tr>
<td>6 font family</td>
<td>6 font size bold</td>
</tr>
<tr>
<td>7 font bold</td>
<td>7 font sizeItalics</td>
</tr>
<tr>
<td>8 font Italics</td>
<td>8 font size medium</td>
</tr>
<tr>
<td>9 font medium</td>
<td>...</td>
</tr>
<tr>
<td>10 font type</td>
<td>45 font medium type</td>
</tr>
</tbody>
</table>
same whether the relevant documents are the first five or the last five, even though the former is preferable.

Average precision, AP, is calculated using precision at \( k \): given a rank \( k \), precision at \( k \) is defined as the precision calculated over the set of retrieved documents with a rank of \( k \) or less. Calculating AP involves iterating down the ranked list, stopping at each relevant document, and calculating the precision at \( k \). These precisions are then averaged. Finally, given a set of queries, MAP is the mean of the AP values taken over the set of queries. MAP is the standard metric used to report the performance of an information retrieval system that produces ranked output [12]. In addition, given that performance on individual queries can be highly variable [12], MAP better represents an IR system’s expected performance on individual queries can be highly variable [12]. MAP better represents an IR system’s expected performance as it smooths over variations and idiosyncrasies possible on any given query.

Finally, following the advice of Smucker et al. [16] who studied statistical testing of IR results, Student’s \( t \)-test is used in the analysis. Therefore, only the (paired) \( t \)-test \( p \)-values are reported.

III. STUDY REPLICATION

The original study used the Bell Communications Research implementation of LSI to index the document collection and process the queries [2]. In preparing the corpus (following the steps described in Section II-A), the replication extracted the same blocks of code from the source as the original study. From each block, comments and identifiers were extracted. The same thirty-five word stop-list of C keywords was used to filter the extracted words.

To evaluate the impact of Normalize, the four different query sets described in Section II-B are used with the three variants of the Mosaic source. The variants capture a spectrum based on how identifiers are treated. At one end of this spectrum, conservative splitting [5] provides the baseline or original variant. At the other end, Normalize provides the most dramatic treatment. In between the aggressive splitter, GenTest, which has similar or superior precision when compared to other aggressive splitters [6], is used. Comparison’s with GenTest help discern the relative impact of splitting versus expansion.

Normalize’s impact is measured relative to the first two variants. Thus, two statistical comparisons are made with Normalize’s output: first with the output obtained using conservative splitting and second with the output obtained using GenTest (note that the comparison between conservative splitting and GenTest does not address either of the two research questions and is thus never made). While one could argue that the two comparisons are independent, the analysis is conservative and applies Bonferroni’s correction for multiple comparisons. Uncorrected, the following \( p \)-values would be halved.

The results for all four query sets, shown in Figure 1, are considered independently. To begin with, the improvement for the Original Set is 10 percentage points. While this improvement might be considered practically significant, Student’s \( t \)-test (paired), applied to the average precisions, shows no statistical difference between conservative splitting and Normalize \( (p = 0.392) \). Similarly, the 16 percentage point difference when comparing GenTest with Normalize is not statistically significant \( (p = 0.137) \). The \( p \) value is slightly lower primarily because the average precisions for GenTest have a lower standard deviation. The lack of statistical significance is perhaps not surprising given the low power available with only eight queries (this first set is the smallest of the four).

Next turning to the auto-generated query set, also shown in Figure 1, Normalize dramatically improves the performance of the concept locator when processing these queries. For this set, paired \( t \)-tests show that the 52 and 49 percentage point differences that Normalize achieves are significantly better than conservative splitting \( (p < 0.0001) \) and GenTest \( (p < 0.0001) \). One caveat with this result, while the four query sets generally meet the sample independence assumption of the \( t \)-test, this query set is constructed by adding one new word to the end of each query set, which likely impacts the independence assumption. On the other hand \( t \)-tests have been shown equivalent to more general tests when applied to IR tool comparisons and thus have been recommended for such comparisons [16].

MAP values for the final two sets, the pairs and triples, are also shown in Figure 1. Given that pair and triple queries are on average significantly shorter in length than queries in the original set, this data demonstrates an interesting trend regarding the performance of the concept locator on queries of different lengths. Without normalization, short queries negatively affect the performance of the concept locator. However, normalization allows the concept locator to perform equally well with both long and short queries.
Statistically, for the pair queries $t$-tests reveal that Normalize produces significantly better performance than conservative splitting ($p < 0.0001$) and GenTest ($p < 0.0001$). It is here for the shortest queries, that Normalize produces its largest improvement: the differences are 58 and 55 percentage points, respectively. In other words, the mean average precision using Normalize is 182% better than conservative splitting. For the triples the differences are 42 and 40 percentage points; both significant with a $p$ value less than 0.0001.

To further highlight the impact of query length, Figure 2 shows the average precision for all three points on the spectrum using the data from the auto-generated query set where query length ranges from two to 41 words. While using Normalize, there is some value to longer queries, long queries are almost required for the other two approaches.

Finally, the effect size captured using Cohen’s $d$ statistic, ranges from medium to large (0.43 to 0.85) for the Original Set and is large (1.42 to 1.71) for the other three datasets. In all cases, the effect size mirrors the $p$-value. The key characteristic that predicts the influence of Normalize is the length of the query. The longer queries of the Original Set yield smaller differences, while for shorter queries only with normalized vocabulary are good results attained.

IV. SUMMARY

This work contributes to the state of the art in understanding the impact of vocabulary normalization as the first experiment to apply Normalize to an IR-based tool. Even the smallest improvement, for the Original Set, portends the benefit of vocabulary normalization. Because normalization is able to recover key domain terms that were shrouded in invented vocabulary, it is able to improve the ranks of relevant documents in an IR environment. This improvement is most pronounced for shorter, more natural, queries, where there is a 182% improvement. In support of previous findings [3], the empirical data also demonstrates that splitting alone, whether conservative or aggressive, does not expose all of the important vocabulary present in the source code.

V. ACKNOWLEDGEMENTS

Thanks to David Christo for his help with the experimentation and to Andrian Marcus for his assistance replicating the original study. Support for this work was provided by NSF grant CCF 0916081.

REFERENCES