Empirical Software Engineering – An International Journal manuscript No.
(will be inserted by the editor)

Experimental Study On the Effect of Context on Identifier Splitting and Expansion

Latifa Guerrouj · Massimiliano di Penta · Yann-Gaël Guéhéneuc · Giuliano Antoniol

Received: date / Accepted: date

Abstract ;

Context: The literature reports that source code lexicon plays a paramount role in program comprehension, especially when software documentation is scarce. Developers often composed identifiers with abbreviated words and acronyms. They sometimes do not use consistent mechanisms to separate such words. Such choices and inconsistencies impede the work of later developers who must understand identifiers by decomposing them to their component terms and mapping them onto dictionary or domain words. Developers therefore must use the available contextual information to understand source code identifiers.
Aim: This paper aims at investigating on how developers split and expand source code identifiers, and, specifically, on the extent to which different kinds of contextual information could support such a task. In particular, we consider (i) an internal context consisting of the content of functions and source code files in which the identifiers are located, and (ii) an external context involving external documentation: more precisely a thesaurus of acronyms and abbreviations. Furthermore, we investigate how identifier characteristics and context co-factors, such as subjects' background, influence the splitting and expansion performances.
Method: We conducted an experiment with 42 subjects, including Bachelor, Master, Ph.D. students, and Post-Docs. Subjects were asked to split and expand a set of 40 identifiers in absence and presence of internal and external contexts. In the later case, we provided subjects with source code functions, source code files, and source code files, plus a thesaurus of acronyms and abbreviations. The aim is to analyze the effect of increasing the context while performing the given tasks. We randomly sampled the identifiers from a corpus of open-source C programs.

L. Guerrouj · Y. G. Guéhéneuc · G. Antoniol
SOCCER Lab.
DGIGL, École Polytechnique de Montréal, Québec, Canada
E-mail: {latifa.guerrouj, yann-gael.gueheneuc.}@polymtl.ca,
E-mail: antoniol@ieee.org

M. Di Penta
University of Sannio, Italy
E-mail: dipenta@unisannio.it

Results: We report evidence on the usefulness of context for identifier splitting and acronym and/or abbreviation expansion. We observe that the source code files are more helpful than functions. The availability of external sources of information does not significantly impact identifier splitting and expansion. However, we observed that when the source of information increase also the entropy increase but this does not negatively affect the attained accuracy.
Conclusions: The obtained results confirm the conjecture that contextual information is useful in program comprehension, including when developers must split and expand identifiers to understand them. Also, results are in line with performances of automatic identifier splitter algorithms, which improve their performance when contextual information is available. Both developers and automatic tools could benefit from these results: developers could reduce the time and effort deployed when performing such tasks by focusing on the relevant contextual levels. Identifier splitting and expansion tools could enhance their accuracy by considering the levels of interest *i.e.*, the source code files as they have been proven to be the most helpful for identifier splitting and expansion tasks.

Keywords Program understanding, identifier splitting and expansion, identifier context

1 Introduction

As programs evolve, the only up-to-date source of information is often the source code in which identifiers (*e.g.*, names of classes, methods, or attributes) account for approximately more than half of linguistic information [DP05]. Previous work on program comprehension [SBE83, MV95] highlighted the importance of textual information to capture and encode developers' intent and knowledge; they showed that identifiers convey relevant information about the application domain [CT99, DP05, JM01], and have influence on software quality [TGM96, LMFB07, LMFB06].

When the identifiers are made up of full (natural language) words and–or meaningful abbreviations, *e.g.*, *predictBugs*, *execQuery*, *imagePntr*, developers have little trouble identifying component terms, *i.e.*, predict and bugs, execute and query, or image and pointer, and understanding the concepts conveyed by such identifiers. However, identifiers can be acronyms, *e.g.*, SOA, or abbreviations, *e.g.*, *cntr*. Developers knowledgeable enough with the application domain can expand them without excessive doubts, while less knowledgable developers could face off some difficulty—*e.g.*, does *cntr* map to "control" or to "counter"? In some other cases, the abbreviations are quite cryptic, *e.g.*, *cwdfn* and their expansion becomes therefore non-trivial. To understand such identifiers, developers can use the neighbor source code (including other identifiers) or comments to split and expand identifiers. Tools developed for automatic splitting and expansion of identifiers, such as Samurai [EHPVS09], TIDIER [GPAG11], or Normalize [LB11] also exploit contextual information in the splitting and expansion process.

In this paper, we experimentally investigate factors that influence developers' performances in identifiers splitting and expansion. We conduct an experiment with 42 subjects, including graduate students and post-docs at the École Polytechnique de Montréal. Each subject splitted and expanded a set of identifiers extracted from various C programs with several type of contexts.

In the experiment, subjects have to disambiguate concepts embedded in source code identifiers by *splitting* and *expanding* them: The splitting consists of dividing compound identifiers into their constituent terms refereed to as "hard-words". Hard-words are terms reflecting domain concepts [LMFB06] *e.g.*, domain-specific terms or English words. Hard words are usually concatenated to form compound identifiers, using the Camel Case naming convention, *e.g.*, *fixBug*, or underscore, *e.g.*, *fix_bug*. When no Camel Case convention or other separator (*e.g.*, underscore) is used, *e.g.*, *fixbug* or *pntrapplicationgid*, the component words *fix*, *bug*, *application*, the abbreviation *pntr*, and the acronym *gid* (*i.e.*, group identifier) are called "soft-words" [LFB06]. Identifier expansion consists of mapping soft-words to their corresponding domain concepts; it is helpful for programming languages (*e.g.*, C and C++) that favor the use of short identifiers. In such languages, the use of abbreviations and acronyms is likely an heritage of the past when certain operating systems and compilers limited the maximum length of identifiers. In fact, a C developer may use the soft-word *dir* instead of the hard-word *directory*, *pntr* instead of *pointer*, or *net* instead of *network*.

In the experiment, we specifically investigate:

1. The effect of two types of contextual information, namely, the internal context, *i.e.*, source code functions and files, and the external context, *i.e.*, a thesaurus of acronyms and abbreviations.
2. The difficulty in expanding abbreviated identifiers, acronyms, and compound identifiers, *i.e.*, identifiers composed of full English words, acronyms and–or abbreviations.
3. The effect of co-factors: subjects' gender, experience, and knowledge background.

Results show that internal contextual information significantly impacts subjects' performance when they split and expand identifiers. Specifically, the availability of source code files significantly help to better split and expand identifiers. Furthermore, even if the entropy (number of possible choices) increases, accuracy also increases; we explain this observation with the assumption that a richer context is more important than a limited one (source code functions) and that there is no better information than more information. Indeed, human-based splitting benefits of contextual information as automatic tools (TIDIER in particular) do, although automatic tools could rely on even richer contexts (*e.g.*, program using the identifier instead of the source code file or function where it appears) which is not feasible for developers. Results also indicate that the knowledge of English influence subjects' capability to split and expand identifiers but also their ability to benefit of the increase of contextual information. Other characteristics, such as gender and domain knowledge do not have a significant effect.

This paper is organized as follows. Section 2 describes our experimental study and the analysis method we followed to address our research questions. Section 3 reports and discusses the quantitative results of our experiment. A qualitative analysis of our results is provided in Section 4 while section 5 presents the threats to validity related to our work. In Section 6, we relate this work to the existing literature. Finally, Section 7 concludes and outlines future work.

2 Experiment Definition and Planning

This section describes our experiment following to the templates provided by Basili *et al.* [BCR94] and by Wohlin *et al.* [CPM+00].

2.1 Experiment Definition

The *goal* of this study is to investigate how developers split and expand source code identifiers, with the *purpose* of evaluating the impact of contextual information as well as of other factors, such as identifier characteristics and developers' background, on such task. The *quality focus* is program understanding, which could be eased by adopting meaningful identifiers. The *perspective* is of researchers and practitioners interested to (i) investigate the extent to which contextual information helps developers properly understand and map identifiers to dictionary (or domain) words, (ii) determine the developers' characteristics that are relevant for the identifier splitting and expansion task. The former information reveals the most important contextual information for the studied task and hence reduces the time and effort costs of attempting to investigate several sources of information to perform a similar task. The latter information provides practitioners with information about the human characteristics required to achieve identifier splitting and expansion.

The *context* of this study consists of *subjects*, *i.e.*, Bachelor, Master, Ph.D. students, and Post-Docs involved in the experiment, and *objects*, *i.e.*, identifiers to be split and expanded, and their related context (*e.g.*, source code files or functions in which they appear).

2.1.1 Context Selection

The subjects are students of the Computer and Software Engineering department of École Polytechnique de Montréal. The population is composed of 28 Ph.D., eight Master, and five Bachelor students, plus one Postdoctoral fellow. According to a pre-experiment questionnaire we conducted before running the experiment, some subjects are native English speakers while others are not. In addition, all subjects have at least a basic knowledge in the C programming language and have performed at least one maintenance task the previous years (*i.e.*, it was not the first time they tried to deal with source code). Table 1 reports the main descriptive statistics of subject's C programming experience. The subjects' population includes a variety of training levels (from Bachelor to Post-Doctoral) and can be considered representative of young developers hired by companies in the Montreal area.

Table 1 Subject's C Programming Experience: Descriptive Statistics

C Programming Experience				
Min	Max	Mean	Median	Std. Dev.
5 months	20 years	3 years	2	4

The objects are 50 identifiers randomly extracted from 340 open-source programs: 337 C programs from the GNU repository (also used as benchmark in a previous paper [NLP+10]), two operating systems (the Linux Kernel release 2.6.31.6 and FreeBSD release 8.0.0), and the Apache Web server release 2.2.14. The randomly-extracted identifiers include function, parameter, and structure names. We dealt with such a large number of C applications to show that our findings are valid for different domains.

In this paper, we focus on C programs only because, as found in our previous work [GPAG11], Java identifiers strictly adhere to the Camel Case convention and

identifier construction rules as opposite to C and C++ where developers tend to use short identifiers.

2.2 Research Questions and Hypothesis Formulation

In the following, we present the research questions that we address in our paper, and formulate the related null hypotheses.

1. **RQ1:** *To what extent does contextual information impact the splitting and expansion of source code identifiers?* This research question analyzes the developers' performance when splitting and expanding identifiers in absence and presence of contextual information. In fact, when developers split identifiers and expand abbreviations, they may do so relying on information (*e.g.*, comments or other identifiers) in the neighbor of the identifiers, or else on other external sources of information (*e.g.*, an acronym dictionary). In this paper, we consider two types of contextual information (*i.e.*, internal and external contexts). These context' types lead to three possible context levels: contextual information related to the function where the identifier appears; contextual information related to the file that contains the identifier; and contextual information related to both the source code file and a dictionary of abbreviations and acronyms. We also study the case where no contextual information is provided. Hence, we have a total of four possible levels of context, representing the increasing level of information the developer could gather when splitting and expanding identifiers: no contextual information; function, file, and file plus acronyms and abbreviations. The null hypothesis being tested to address this research question is:
   - H_{01}: *There is no significant effect of the given context on the subjects' performances when splitting and expanding source code identifiers.*
2. **RQ2:** *To what extent do the characteristics of source code identifiers affect splitting and expansion performances?* This research question investigates whether particular characteristics of an identifier may favor or hinder the developers' capability of splitting and expanding it. Specifically, we consider whether identifiers are abbreviations, full English words, or acronyms only, and whether they are combinations of these types. In addition, we compute a metric—named *splitting and expansion entropy*—aiming at capturing the difficulty in splitting and expanding an identifiers, related to the number of possible splittings and expansions (possible choices or degrees of freedom) one has when performing such a task.
   Entropy was first introduced by Shannon [Sha48] in the early 50s to model information sources; given a source of information, say X, emitting symbols, x_i (*e.g.*, letters or words), the source entropy (or simply entropy) measures the symbol information or conversely the symbols' uncertainty. Let $p(x_i)$ be the probability of observing the symbol x_i, entropy is formally defined as:

$$H(X) = \sum p(x_i) log[p(x_i)] \tag{1}$$

   $H(X)$ is also often interpreted as the number of bits needed to encode the source information because $2^{H(X)}$ source code words are needed to represent and transmit X symbols. Shannon estimated the entropy of English characters in about 2.14 bits per letter and, because the average English words is 5.5 character long, written

English word entropy is on average of 11.82 bits per word. Given one English word, there are about 4096 (*i.e.*, 2^{12}) possible subsequent words. More recent, results slightly reduce the entropy of English words to about 10 bits per word but essentially the set of possible words following a given one is on the order of thousands.

When splitting and expanding identifiers, symbols x_i are terms composing identifiers (*i.e.*, acronyms, jargon terms, abbreviations and English words) and $p(x_i)$ is the probability of observing such a term. We conjecture that the difficulty in splitting and expanding identifiers is related, among other factors, to entropy. It is worth underlying that entropy cannot fully explain the difficulty of developers in splitting and expanding an identifiers: other factors such as developer's knowledge, familiarity with the domain, source code, as well as experience play a fundamental role, which we explore in RQ3

Let us first assume the identifier is split into composing terms (by removing the underscore *mem_pntr_val* is mapped into *mem*, *pntr*,*val*), and further assume the correct split and expansion is known, *i.e.*, the availability of an oracle. The distance between the identifiers terms (*mem*, *pntr*,*val*), and the oracle (*memory*, *pointer*, *value*) is eighth, because a total of eighth letters are removed from (*memory*, *pointer*, *value*) to obtain (*mem*, *pntr*,*val*). This defines an upper bound when searching into the dictionary D for words possibly generating identifier terms. We believe it is extremely unlikely that the user or a tool will go any further in searching for a match of a fraction of the oracle (*i.e.*, correct identifier split and expansion) length as it will imply changing all oracle characters. Indeed, in general, the upper bound is smaller than the length of the oracle, 18, or the identifier itself. Our conjecture is that in expanding an identifier term, the developer tend to use dictionary words having closest possible distance from the term. According to this conjecture, we define a split-expansion entropy $H_{SE}(w)$ for a given word w, a dictionary D and an oracle identifier distance d as:

$$H_{SE}(w) = \sum_{t} \sum_{w \in D_t(d)} p(w) log[p(w)] \tag{2}$$

where t varies over the identifier terms (*mem*, *pntr* and *val* in our example), and $D_t(d)$ is the sub-dictionary extracted from D and containing all D terms at a distance lower or equal than the distance of d from t. $p(w)$ is the uniform distribution over the sub-dictionary $D_t(d)$, thus $p(w) = 1/|D_t(d)|$ where $|D_t(d)|$ is the cardinality of $D_t(d)$.

To compute H_{SE}, we use two basic configuration for the dictionary D: (i) the dictionary of English words extracted from the context and (ii) we augment this dictionary with the following information:

- English words, longer than two characters, extracted from WordNet and the Ispell dictionary (about 150,000 words);
- a list of well-known C abbreviations and acronyms (390 words).

In summary, for what concerns **RQ2**, we test two null hypotheses:

- H_{02a}: *the subject's performances in identifier splitting and expansion does not significantly vary among the types of identifiers we studied.*
- H_{02b}: *the subject's performances in identifier splitting and expansion is not correlated to the splitting and expansion entropy.*

3. **RQ3:** *To what extent subjects' background and characteristics impact the performance of identifier splitting and expansion?* This research question investigates how co-factors related to the developers performing the tasks impact the splitting and expansion performances and the extent to which such factors interact with the use of contextual information. Specifically, the factors that we consider are: gender, level of experience, programming language (C) knowledge, domain knowledge, and English proficiency. Given $factor_i$, one of these factors, we test two null hypotheses:
   - H_{03a}: *there is no significant effect of* $factor_i$ *on identifier splitting and expansion accuracy.*
   - H_{03b}: *there is no significant interaction between* $factor_i$ *and the studied levels of context on identifier splitting and expansion accuracy.*

2.3 Variable Selection

In the following, we describe the dependent and independent variables of our experimental study.

2.3.1 Dependent Variable

The experiment has one dependent variable, namely, the correctness of the provided splitting and expansion, which measure the subjects' performances when performing identifier splitting and expansion. Such a correctness is measured with respect to an oracle that we manually build, using precision and recall measures.

Given an identifier s_i to be split and expanded, $o_i = \{oracle_{i,1}, \ldots oracle_{i,m}\}$ the expansion in the manually-produced oracle, and $t_i = \{term_{i,1}, \ldots term_{i,n}\}$ the set of terms obtained by subjects, we define precision and recall as follows:

$$precision_i = \frac{|t_i \cap o_i|}{|t_i|}, \quad recall_i = \frac{|t_i \cap o_i|}{|o_i|}$$

To provide an aggregated, overall measure of precision and recall, we use the F-measure, which is the harmonic mean of precision and recall:

$$\mathrm{F-measure} = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

We manually built our oracle by associating each identifier with a list of terms obtained after splitting it and expanding them. We only create one oracle for both the splitting and expansion because we consider them as one task; we justify this assumption by the fact that identifier expansion involves identifier splitting. In fact, one has first to identify terms composing an identifier and then expand them to their corresponding domain concepts. For example, the oracle entry for *counterPntr* would be *counter pointer*, obtained by splitting the identifier after the seventh character and after expanding the abbreviation *Pntr* into *pointer*. We built the oracle following a consensus-based approach (*i.e.*, one author proposed an identifier split and expansion, which was then verified and validated by a second author). In a few cases, where there was a disagreement, a discussion was held among all the authors and a consensus was reached. If a consensus was not possible and contrasting evidence was found, we remove

Table 2 Null hypotheses and independent variables

RQs	Hyps.	Descriptions	Variables
RQ1	H_{01}	Effect of context	Context levels: no context, function, file, file plus acronyms and abbreviations
RQ2	H_{02a}	Effect of identifiers' type	Type of identifiers: full English words only (plain), acronyms only (acro), abbreviations (abbr) only, and identifiers using combinations of these types (comp)
	H_{02b}	Effect of splitting/expansion entropy	Splitting/expansion entropy
RQ3	H_{04a}	Effect of person characteristics/backgrounds	Co-factors: gender, experience, C knowledge, domain (Linux) knowledge, English proficiency
	H_{04b}	Interaction of co-factors with use of context	

the identifier from the oracle. We adapted this approach to minimize the bias and the risk of producing erroneous results. This procedure was motivated by the complexity of identifiers, which capture developers' domain and solution knowledge, experience, and personal preference. Indeed, sometimes it is difficult to decode the true meaning of identifiers and thus in the construction of the oracle, we performed a manual analysis of various sources of information, such as projects change logs, developers mailing lists, source-code comments, and user manuals.

2.3.2 Independent Variables

The independent variables of our study are all the factors that we considered when testing the null hypotheses formulated in Section 2.2 and summarized in Table 2.

As Table 2 shows, other than the independent variables of **RQ1** (context level), **RQ2** (types of identifiers and splitting entropy), **RQ3** considers how co-factors pertaining subjects' characteristics influence identifier splitting and expansion. Specifically, we considered the following co-factors:

1. **Program of studies**. Subjects are enrolled in/pursuing different programs. Their study level ranges from Bachelor to Post-Doc.
2. **Gender**. Female and male may differently perform the identifier splitting and expansion tasks.
3. **Knowledge of C**. The subjects may have different levels of expertise in C programming and hence different levels of C understanding.
4. **Level of English**. Vocabulary of the programs from where we sampled identifiers is written in English. Some subjects are native English speakers while others may have a different level of English proficiency.
5. **Linux knowledge**. C Applications from were we sampled identifiers are from Linux utilities and kernels. Some subjects are knowledgable in such a type of utilities while others are not.

Table 3 reports some statistics about these co-factors.

Table 3 Co-factors related to subjects' characteristics and background.

Additional Factors	Treatments	# Participants
Program of Studies	Bachelor	5
	Master	9
	PhD	28
	Post-Doc	1
Gender	Female	30
	Male	12
C Programming Experience	Basic	11
	Medium	23
	Expert	9
Level of English	Bad	8
	Good	8
	Very Good	18
	Excellent	8
Linux Knowledge	Occasional	12
	Basic Usage	13
	knowledgeable but not expert	17
	Expert	0

2.4 Experiment Procedure and Design

This section presents the design choices that we made to limit threats to our experiment's validity. We first detail the tasks performed by subjects, then we present co-factors collected to mitigate the effect of confounding factors, and finally we describe the experiment design and how the extraneous factors are controlled.

2.4.1 Experimental Tasks

During the experiment, we asked each participant to split and expand 40 identifiers that we presented to them using a Web application, in four possible configurations:

- without a context *i.e.*, just an identifier in an empty page;
- within a "source code function", *i.e.*, subjects could browse the source code of the function containing the identifier;
- within a "source code file", *i.e.*, subjects could browse the source code of the file containing the identifier;
- with an "external context", *i.e.*, subjects have access to a list of widely used acronyms and abbreviations, named AcronymFinder[1].

Source code was mapped into HTML via HTML-highlight, an open-source tool that highlights source code in HTML documents, thus helping subjects locate language elements (*e.g.*, for, if, while) via coloring. When performing the experiment, subjects had access to computers with two screens. On one screen, they had access to the Web application where all questions and tasks appear. On the second screen, they had access to the response form. We informed the subjects before the experiment that it takes maximum 120 minutes and that they are free to leave at any time without incurring any penalty. We did not measure the time and we did not impose any time constraint. Collected information was anonymous. We validated the response forms to make sure that subjects correctly follow the experiment procedure. Subjects were aware of the purpose of the experiment that is to perform program understanding tasks in the

[1]

presence or absence of context information but did not know the exact hypotheses tested.

2.4.2 Experiment Design

This section describes our experiment design where the main details are summarized in Table 4.

Table 4 Experimental design

Objects' sets	**Group**$_1$	**Group**$_2$	**Group**$_3$	**Group**$_4$	**Group**$_5$
Set$_1$	ids1 + cx_1	ids2 + cx_1	ids1 + cx_4	ids1 + cx_3	ids1 + cx_2
Set$_2$	ids2 + cx_2	ids3 + cx_2	ids3 + cx_1	ids2 + cx_4	ids2 + cx_3
Set$_3$	ids3 + cx_3	ids4 + cx_3	ids4 + cx_2	ids4 + cx_1	ids3 + cx_4
Set$_3$	ids4 + cx_4	ids5 + cx_4	ids5 + cx_3	ids5 + cx_2	ids5 + cx_1

Given Group$_i$ group i of subjects, ids_j a set j of ten identifiers to be splitted and expanded by subjects with $j \in \{1, 2, 3, 4, 5\}$, and cx_k effect of context k on the subjects' performance with $k \in \{1, 2, 3, 4\}$, $ids_j + cx_k$ is the set j of identifiers to be splitted and expanded using context k by group i of subjects. In our case, the different levels of context are cx_1: no contextual information; cx_2: contextual information related to the function where the identifier appears; cx_3: contextual information related to the files where the identifier appears; and cx_4: as cx_3, plus the availability of a dictionary to help expanding abbreviations and acronyms. As described in Table 4, each group of subjects dealt with the four introduced context levels, we gave each group i of subjects a set j of ten identifiers per context *i.e.*, a total of forty different identifiers to split and expand in the order shown in Table 4. We adopted a randomized block design to account for subjects' individual differences, increase the statistical power of our analysis, and ensure the design would not introduce any bias in the results of the experiment. Another advantage of proceeding with a block design is that we can study the interaction between subjects' characteristics and context. We therefore grouped subjects into blocks. We considered blocks of participants with a basic, medium, and expert knowledge in C. Subjects were also blocked according to their level of English and their gender. Then, we created five groups randomly assigning subjects from blocks in nearly identical proportions. Each of the five groups is of approximately the same size (either eight or nine subjects). Thus, in Table 4, the rows represent the set of identifiers that have been splitted and expanded by subjects with the used level of context whereas the columns show the five groups of subjects. The table cells therefore show, for each group of subjects, the set of identifiers on which they worked and the context level given to them. Each subject performs the identifier splitting and expansion individually; we prevented the subjects from collaborative work and we made sure all the tasks were individually performed. The rationale for making subjects deal with forty identifiers belonging to different domain applications was (1) to maximize the number of data points (observations) so as to increase statistical power, (2) to avoid bias from the differences in programs' complexity, and (3) to make general conclusions from the obtained results. When subjects perform several times similar tasks and use the same levels of context, they are subject to learning or fatigue effects. We limit the threats of such effects in two ways: First, each group of subjects splits and expands a specific

group of identifiers using all the contexts (functions, files, and files plus acronyms and abbreviations). Second, we presented to the subjects different identifiers belonging to different programs to avoid learning effects being confounded with context effects.

2.4.3 Post-experiment Questionnaire

Finally, we presented post-experiment questionnaires to the subjects to gain insight on the collected data. We asked each subject whether the context was helpful for him/her. A summary of the post-experiment questionnaire is shown in Table 5; answers were collected on a five point Likert scale plus a free text form where subjects were asked to provide further comments if any.

Table 5 Post-Experiment Survey Questionnaire

ID	**Question**
Question 1	Was the context helpful when splitting and expanding identifiers? If yes, what was the most helpful type of context among the 3 (function, file or file plus AcronymFinder) investigated ones?
Question 2	Does your C programming experience help you when splitting and expanding the identifiers in question?
Question 3	To what extent was your knowledge in Linux helpful when splitting and expanding source code identifiers?
Question 4	Were the comments provided with the source code helpful for you when splitting and expanding the identifiers in question?
Question 5	How do you rate the level of complexity of the programs given to you (Simple, Complex, Too complex)?
Question 6	How did you find the information provided in the procedure (Not helpful, Helpful, Very helpful)?
Question 7	What was the impact of information provided in the Post-experiment procedure on your performance in accomplishing the requested tasks (Negative, Positive, Required too much concentration, Correct)?

2.5 Analysis Method

RQ1 concerns the comparison of the precision, recall, and F-measure of identifier splitting/expansion provided by subjects for the studied levels of context. Other than showing boxplots and descriptive statistics, we test the null hypothesis H_{01} using the Wilcoxon paired tests to perform a pairwise comparison of the results obtained for each identifier (on which the test is paired) with the four different levels of context. Since, we apply the Wilcoxon test multiple times, we need to adjust p-values. We use the Holm's correction procedure [Hol79]. This procedure sorts the p-values resulting from n tests in ascending order of values, multiplying the smallest by n, the next by $n-1$, and so on. Finally, in addition to the statistical comparison, we compute the effect-size of the difference using Cliff's delta non-parametric effect size measure [GK05], defined as the probability that a randomly-selected member of one sample has a higher response than a randomly selected member of a second sample, minus the reverse probability. Cliff's delta ranges in the interval $[-1, 1]$ and is considered small for $0.148 \leq d < 0.33$, medium for $0.33 \leq d < 0.474$, and large for $d \geq 0.474$.

RQ2 concerns the comparison of the correctness, precision, recall, and F-measure of identifier splitting/expansion achieved for the studied types of identifiers, specifically (i) identifiers composed of plain English words only, (ii) identifiers containing abbreviations (and possibly English words), (iii) identifiers containing acronyms (and possibly English Words), and identifiers containing both acronyms and abbreviations. We perform the analyses of **RQ2** similarly to that of **RQ1**, *i.e.*,, using Wilcoxon paired test (with p-values corrected with Holm's correction) and Cliff's delta. Also, we used two-way ANOVA to test the interaction between the context levels and identifier types.

Furthermore **RQ2** analyzes to what extent the performance in identifier splitting and expansion correlate with the splitting and expansion entropy H_{se}. To this aim, we use the Spearman rank correlation to determine the presence of a significant correlation, and its magnitude.

RQ3 analyzes the interaction between the effect of context levels on subjects' performances when splitting/expanding identifiers and a set of investigated co-factors (gender, experience, Linux expertise, C knowledge, and English proficiency). To this aim, we use two-way ANOVA.

3 Experimental Results

In this section, we report the quantitative results of our experiment.

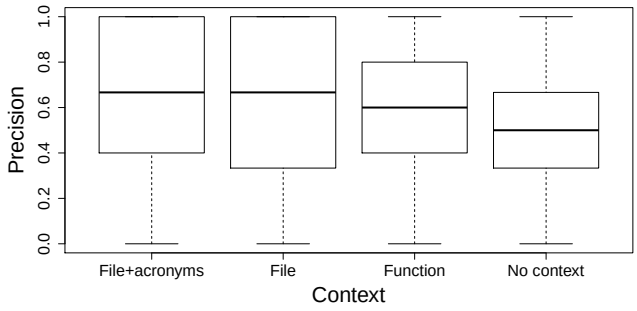
3.1 RQ1 – Context Relevance

The first quantitative results are related to the relevance of contextual information when splitting and expanding source code identifiers. In the following, AcronymFinder is the thesaurus of acronyms and abbreviations we used as external documentation.

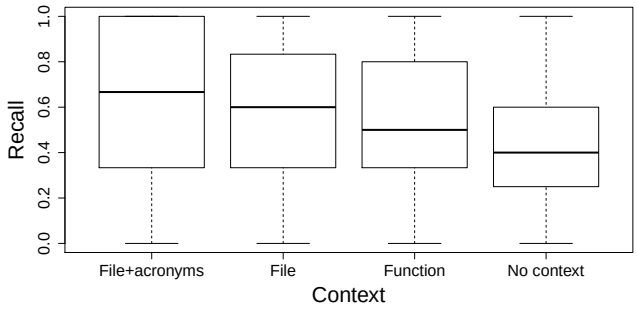
3.1.1 Identifier Context Levels Relevance

Fig. 1 shows the boxplots of precision, recall, and F-measure computed for different context levels *i.e.*, (i) functions, files, files plus AcronymFinder, and no contextual information. Descriptive statistics 1^{st} quartile, median, 3^{rd} quartile, mean and standard deviation) are reported in Table 6.

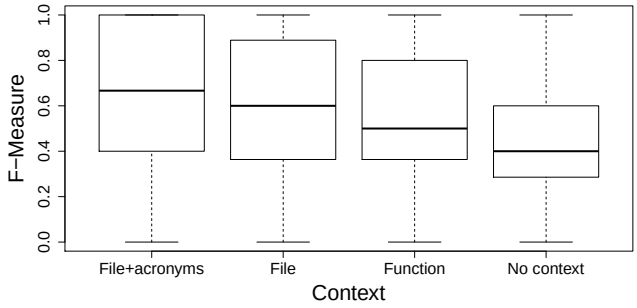
The boxplots and the table show that the subjects achieve the best performances in terms of precision, recall and F-Measure when using files plus AcronymFinder and file contexts. The recall (and consequently the F-measure) results are slightly higher for the File plus AcronymFinder context, while the precision is similar for the two contexts. This could be justified by the fact that when the AcronymFinder is given with the files, subjects use it to expand the identifiers' terms that they would not have been able to expand with the source code only. Function-level context yield lower subject's performances (precision, recall, F-measure) than the other cases and the performances are even lower when no contextual information is provided. Data also show that, as the available context increases (i.e., from no context to function-level context, file, and file plus AcronymFinder), the performance variation (in terms of interquartile range and of variance) increases, especially for precision. This result can be explained because, although a wider context can provide additional information which could, in some case, increase the uncertainly when performing the splitting and expansion. One could find



(a) Precision



(b) Recall



(c) F-Measure

Fig. 1 Boxplots of precision, recall, and F-measure for the different context levels.

in the additional context words that could drive splitting and expansion in the correct, but also, in some cases, in the wrong direction.

Table 7 reports results of the statistical comparison among the different context levels; it shows the difference in subject's performances when using a context level *Context 1* versus a second possible level of Context *Context 2* (adjusted p-values and Cliff's *d* effect size are positive when the effect is in favor of *Context 1*). Concerning precision, there is a significant difference in subjects' performances between file plus AcronymFinder and no context as well as between file and no context, in both cases with a *medium* effect size. There is a marginal difference (p-value = 0.09) between file

Table 6 Precision, recall and F-measure of identifier splitting and expansion with different contexts.

Metrics	Contexts	1q	Median	3q	Mean	σ
Precision	file plus Acronym Finder	0.40	0.67	1.00	0.63	0.31
	file	0.33	0.67	1.00	0.61	0.30
	function	0.40	0.60	0.80	0.59	0.29
	no context	0.33	0.50	0.67	0.46	0.28
Recall	file plus Acronym Finder	0.33	0.67	1.00	0.62	0.31
	file	0.33	0.60	0.83	0.60	0.31
	function	0.33	0.50	0.80	0.55	0.30
	no context	0.25	0.40	0.60	0.42	0.27
F-Measure	file plus Acronym Finder	0.40	0.67	1.00	0.62	0.31
	file	0.36	0.60	0.89	0.60	0.30
	function	0.36	0.50	0.80	0.56	0.30
	no context	0.29	0.40	0.60	0.43	0.27

plus AcronymFinder and function. In all other cases, there is no significant difference. For recall, there is a significant difference in subject's performances between files plus AcronymFinder and function (small effect size), file plus AcronymFinder and no context (medium effect size), between file and function (though the effect size is negligible), between file and no context (medium effect size), and between function and no context (medium effect size). Finally, for the F-Measure, differences are significant between file and no context and between file plus AcronymFinder and no context (in both cases with a medium effect size), between file plus AcronymFinder and function (small effect size), and between function and no context (medium effect size).

Table 7 Precision, recall, and F-measure for different context levels: results of Wilcoxon paired test and Cliff's delta.

Precision			
Context 1	Context 2	Cliff's d	adj p
file plus AcronymFinder	file	0.044	0.53
file	function	0.040	0.53
file	no context	0.28	<0.01
file plus AcronymFinder	function	0.084	0.09
file plus AcronymFinder	no context	0.32	<0.01
function	no context	0.25	<0.01
Recall			
Context 1	Context 2	Cliff's d	adj p
file plus AcronymFinder	file	0.036	0.36
file	function	0.097	0.03
file	no context	0.32	<0.01
file plus AcronymFinder	function	0.13	<0.01
file plus AcronymFinder	no context	0.35	<0.01
function	no context	0.23	<0.01
F-Measure			
Context 1	Context 2	Cliff's d	adj p
file plus AcronymFinder	file	0.041	0.31
file	function	0.069	0.16
file	no context	0.31	<0.01
file plus AcronymFinder	function	0.11	0.02
file plus AcronymFinder	no context	0.34	<0.01
function	no context	0.25	<0.01

In summary, we can conclude that *contextual information significantly increases the subject's performances when splitting and expanding identifiers*, in terms of precision, recall, and F-Measure. There is a slight difference between a file-level context and

function-level context, although not statistically significant. Also, the presence of an AcronymFinder also slightly increases the performances even though the difference is not statistically significant.

3.2 RQ2: Identifier Types and Splitting/Expansion Entropy Effect

In this section, we study the impact of identifier's type and splitting/expansion entropy on subjects' performances when splitting and expanding source code identifiers.

3.2.1 Identifier Types Effect

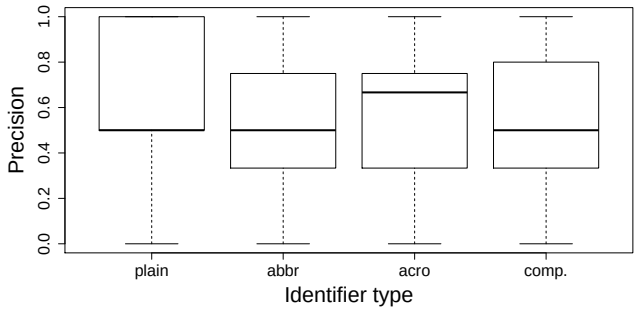
We first show the results of dealing with different types of identifiers *i.e.*, identifiers composed of plain English words only (plain), identifiers containing abbreviations (abbr), identifiers containing acronyms (acro), and identifiers containing both abbreviations and acronyms *i.e.*, composite identifiers (comp).

Fig. 2 shows boxplots of precision, recall, and F-measure obtained for the studied types of identifiers. Table 8 reports results of the comparisons between results achieved for different types of identifiers. We compare subjects' performances obtained with a type *Type 1* of an identifier versus those attained with a second possible type *Type 2*. In terms of precision, the table does not show significant differences. In terms of recall, identifiers in plain English are split/expanded significantly better—and with a *medium* effect size—than acronyms and composite identifiers (containing both abbreviations and acronyms). In all other cases, there is no significant difference. Finally, in terms of F-measure the table shows that there is no significant difference among different types of identifiers. Having a significantly lower recall for acronyms means that subjects were not able to split some acronyms, while the problem was relatively mitigated for abbreviations. A possible interpretation is that the expansion of abbreviations can be guessed as an abbreviation is just the full word without some letters (usually vowels). To expand an acronym, either one knows it, otherwise it is difficult to guess. Overall, we can say that types of identifiers significantly impact the recall measure because the type mainly dictates whether the subjects are able or not to expand all the terms.

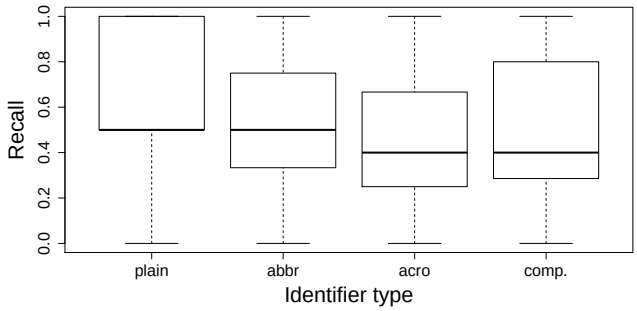
Table 9 shows the two-way ANOVA of recall (the only measure for which we obtained significant differences in subjects' performances) by identifier type (Id. Type) and context (Context). As the table shows, not only both variables have a significant effect but also there is a significant interaction (results of the interaction are those of the Id. Type:Context row) (p-value = 0.034).

To aid better interpreting such result, Fig. 3 shows the interaction plot of recall (y-axis) by type of identifier (x-axis) and used context (different traces). As it can be noticed from the figure:

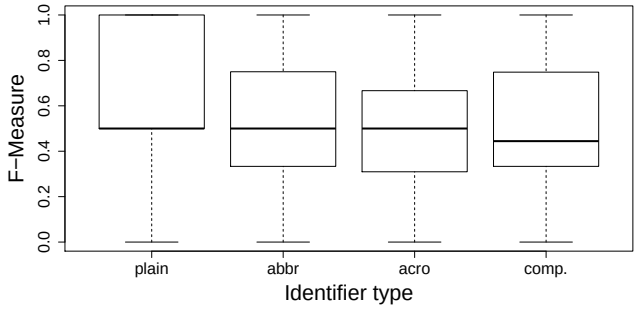
- the line related to no context clearly stays below the others, because as concluded in **RQ1**, identifier splitting and expansion performances are significantly lower when no context is provided. Similarly, the line of function-level context stays a bit below the others (except in correspondence of abbreviation where the mean recall is similar to file-level context);
- for file-level context, the mean recall is very high for plain English identifiers, then there is a sudden decrease in correspondence of abbreviations, then slightly increase again for acronyms and remains almost constant (with a slight decrease) for composite identifiers;



(a) Precision



(b) Recall



(c) F-Measure

Fig. 2 Boxplot of Precision, Recall, and F-Measure for different identifier types.

– for file plus AcronymFinder, while there is a decrease, again, between plain English words and abbreviations, such a decrease is less evident, even because the mean recall for plain English identifiers is lower than for file context. This observation is somewhat surprising, because the additional context (AcronymFinder) should not help nor create confusion during the splitting of plain-English identifiers. Very likely, this difference is just due to variability in subjects' performance. Then, again surprisingly, there is no increase in correspondence of acronyms, as like the AcronymFinder again does not help particularly. In this case, the conjecture is that a general-purpose AcronymFinder was not particular useful to expand specific

Table 8 Comparison between results achieved for different identifier types.

Precision			
Type 1	Type 2	Cliff's d	adj p
plain	abbr	0.15	0.25
plain	acro	0.10	0.82
plain	comp.	0.088	0.82
abbr	acro	-0.022	0.82
abbr	comp.	-0.081	0.28
acro	comp.	-0.060	0.82
Recall			
Type 1	Type 2	Cliff's d	adj p
plain	abbr	0.15	0.15
plain	acro	0.26	0.01
plain	comp.	0.26	0.01
abbr	acro	0.093	0.16
abbr	comp.	0.058	0.35
acro	comp.	-0.039	0.50
F-Measure			
Type 1	Type 2	Cliff's fd	adj p
plain	abbr	0.15	0.20
plain	acro	0.13	0.44
plain	comp.	0.20	0.07
abbr	acro	0.019	1.00
abbr	comp.	0.0065	1.00
acro	comp.	-0.045	1.00

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Id. Type	3	1.18	0.39	4.20	0.0057
Context	3	6.16	2.05	22.04	<0.01
Id. Type:Context	9	1.69	0.19	2.02	0.034
Residuals	1295	120.73	0.09		

Table 9 Recall: Two-way ANOVA by Identifier Type & Context

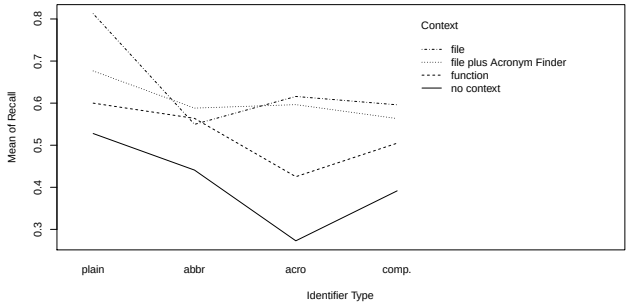


Fig. 3 Recall: Interaction plot of Identifier Type and Context.

acronyms of GNU utilities and Linux kernels (e.g., acronyms related to network protocols, drivers, etc.).

In summary, we can conclude that *acronyms are generally more difficult to expand than abbreviations and than identifiers made up of plain English words*. Also, for domain-specific acronyms, general-purpose glossaries are clearly useless or could even be misleading.

3.2.2 Identifier Splitting/Expansion Entropy Effect

Table 10 shows the Spearman rank correlation between precision, recall, F-measure and split-expansion entropy.

We experimented with different ways to distribute the distance d between identifier terms; reported correlations values were obtained by distributing d among terms proportional to the identifier term length. Other configurations (*e.g.*, dividing d by the number of terms) produced low correlation values. Although the correlation is low, in general, higher correlation values are observed for larger contexts (e.g., file rather than functions) and when additional knowledge is added to the context. The interpretation of this result is that if, on the one hand, a wider context increases the chances a developer has to get information helpful to perform splitting/expansion, on the other hand, this also increases the entropy, and, thus, the number of possible splits/expansions she can perform using terms from the context vocabulary and from the available knowledge.

Table 10 Results of Spearman rank correlation between Precision, Recall, F-Measure and split-expansion entropy H_{SE}

Precision		
Context	p-value	ρ
file plus knowledge	**<0.01**	0.22
file	0.18	0.08
function plus knowledge	**0.04**	0.13
function	**0.02**	0.15
Recall		
Context	p-value	ρ
file plus knowledge	**0.04**	0.13
file	**0.01**	0.16
function plus knowledge	**<0.01**	0.20
function	0.12	0.09
F-measure		
Context	p-value	ρ
file plus knowledge	0.07	0.11
file	**<0.01**	0.19
function plus knowledge	**0.01**	0.17
function	0.08	0.11

In summary, although the correlation is not high, we can conclude that *while a larger context increases the splitting/expansion performances, it also increases the splitting/expansion entropy.*

3.3 RQ3: Effect of Co-factors

Table 11 shows the two-way ANOVA of F-measure by context (Context row) and Linux knowledge (Linux row). As the table shows, the Linux knowledge has no effect on the F-measure. Also, there is no significant interaction (Context:Linux row) between the two factors (p-value = 0.2024). Thus, there is no evidence of a correlation between identifier splitting/expansion and the knowledge of Linux. This conclusion confirms also the one gained from the post-experiment questionnaire. In fact, subjects who have non basic experience on Linux, explain that they did not benefit a lot from their expertise in Linux because the given identifiers are specific and require a (too) specific technical Linux knowledge.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Context	3	8.81	2.94	34.02	0.0000
Linux	2	0.12	0.06	0.71	0.4940
Context:Linux	6	0.74	0.12	1.42	0.2024
Residuals	1668	143.90	0.09		

Table 11 F-Measure: Two-way ANOVA by Context & Knowledge of Linux

Table 12 shows the two-way ANOVA of F-measure by context (Context row) and C experience (Cexp row). As the table shows, C experience has no effect on the F-measure. Also, there is no significant interaction (Context:Cexp row) between the two factors (p-value = 0.7733). We conclude that there is no evidence of a correlation between identifier splitting/expansion and C experience. The experience in C does not intervene a lot because probably the subjects (most of them) do not try to understand deeply the source code when expanding identifiers, they try to have an idea about what a function/program is doing, they check comments that surround the code, and try to see where the same identifier could be used in the code given to them.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Context	3	8.81	2.94	33.95	0.0000
Cexp	2	0.29	0.14	1.67	0.1884
Context:Cexp	6	0.28	0.05	0.55	0.7733
Residuals	1668	144.19	0.09		

Table 12 F-Measure: Two-way ANOVA by Context & Knowledge of C

Table 13 shows the two-way ANOVA of F-measure by context (Context row) and gender (Gender row). As the table shows, the gender has a no effect on the F-measure. Also, there is no significant interaction (Context:Gender row) between the two factors (p-value = 0.8554). We conclude that there is no evidence of a correlation between identifier splitting/expansion and the co-factor gender.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Context	3	8.81	2.94	33.94	0.0000
Gender	1	0.11	0.11	1.23	0.2674
Context:Gender	3	0.07	0.02	0.26	0.8554
Residuals	1672	144.59	0.09		

Table 13 F-Measure: Two-way ANOVA by Context & Gender

Table 14 shows the two-way ANOVA of F-measure by context (Context row) and program of studies (Program row). As the table shows, the program of studies has no effect on the F-measure. Also, there is no significant interaction (Context:Program row) between the two factors (p-value=0.8373). We conclude that there is no evidence of a correlation between identifier splitting/expansion and the program of studies. This conclusion confirms the fact that identifier splitting/ expansion requires a knowledge about the domain of the applications rather than a high-level of studies. The latter

explanation is in accordance with what we noticed from the experiment: sometimes Bachelor and Master students performs better than PhD students when providing them with context.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Context	3	8.81	2.94	33.94	0.0000
Program	3	0.42	0.14	1.64	0.1789
Context:Program	9	0.43	0.05	0.55	0.8373
Residuals	1664	143.91	0.09		

Table 14 F-Measure: Two-way ANOVA by Context & Program

Table 15 reports the two-way ANOVA of F-measure by context (Context row) and knowledge of English (English row). As the table shows, not only the English knowledge has a significant effect on the F-measure (p-value=0.0356) but, there is also a significant interaction (Context:English row) between the context and the English knowledge.

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Context	3	8.81	2.94	34.22	0.0000
English	3	0.74	0.25	2.86	0.0356
Context:English	9	1.30	0.14	1.68	0.0888
Residuals	1664	142.73	0.09		

Table 15 F-Measure: Two-way ANOVA by Context & Knowledge of English

Fig. 4 shows the interaction plot of mean of F-measure (y-axis) by English knowledge (x-axis) and context (different traces):

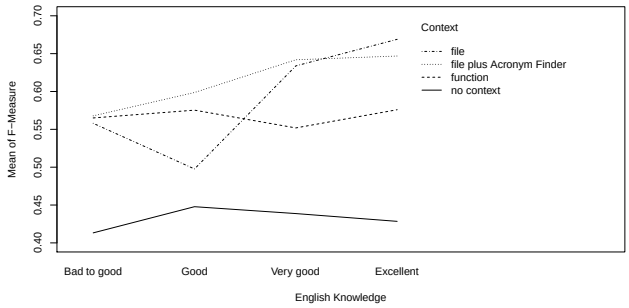


Fig. 4 F-Measure: Interaction plot of Context and English Knowledge.

– the mean of F-measure increases when the level of English increases for the graph related to no context. Clearly, the F-measure moves from 0.41 to 0.45 when the level of English goes from Bad to Good. Yet, this increase in the F-measure mean is

not observable when the level goes from Good to Excellent. Also, the graph related to no context stays below the others (graphs of respectively function, file and file plus AcronymFinder). Overall, we can conclude that, when no context is provided to subjects, the English knowledge helps but not a lot to improve the subjects' performance while splitting and expanding identifiers.
- for the function-level context graph, the mean F-measure increases when the level of English increases from Bad to Good. It goes from 0.56 to 0.59. Then, it decreases towards 0.55 when the level of English goes from Good to Very good. However, the F-measure mean increases to attain 0.56 when the level of English goes from Very good to Excellent. Similarly to the graph that represents no context, the graph of function-level context stays a bit below the others as we previously discussed in **RQ1**. Overall, the English knowledge helps improve the subjects' performance while expanding identifiers in presence of source code functions as context.
- for the file plus AcronymFinder level context graph, the F-measure mean increases when the level of English increases from Bad to Excellent. In fact, the mean of F-Measure increases from 0.57 to 0.59 when the level of English goes from Bad to Good. Then, it increases towards 0.65 when the level of English goes from Good to Very good. The F-measure mean remains stable (0.65) when the level of English increases from Very good to Excellent. Clearly, the F-measure significantly increases when the level of English increases for subjects performing identifier splitting/expansion using as a context the file level plus AcronymFinder.
- for the file level context graph, the F-measure mean decreases when the level of English increases from Bad to Good. It goes from 0.56 to 0.49. Then, it increases towards 0.64 when the level of English goes from Good to Very good. Then, it significantly increases moving from 0.64 to 0.68 when the level of English increases from Very good to Excellent. Overall, the level of English helps improve the subjects' performances when splitting and expanding identifiers using as a context the source code files.

Overall, the level of English does not matter when no context is given to the subjects. Yet, when the level of contextual information increases *i.e.*, when the context level goes from function to file to file plus acronyms and abbreviations, the help of the English Knowledge becomes more significant.

In summary, we can conclude that *there is a significant evidence of a correlation between identifier splitting/expansion and the English knowledge*. This conclusion reveals that the English knowledge is used besides the domain knowledge that developers have about the programs they are dealing with, to understand the source code, and, hence disambiguate the concepts conveyed by source code identifiers.

4 Qualitative Analysis

In this section, we present and discuss the observations reported in the post-experiment questionnaire plus others noticed during the experiment.

To gain insight to strengthen and explain our results, we asked subjects to fill a post-questionnaire which main questions are summarized in Table 5, presented in Section 2.4.3 of this paper. Regarding **Q1**, thirty subjects agreed that the most helpful context for them was the file-context level. Eight subjects found that the function level was the most useful for them. Yet, only four subjects found that the file plus

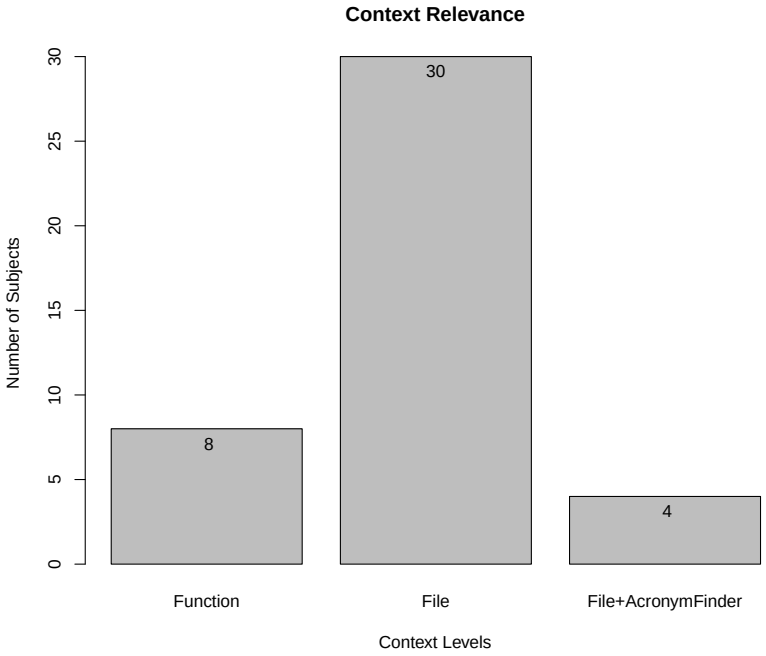


Fig. 5 Post-experiment Questionnaire: Context Relevance.

AcronymFinder level was the most helpful during the experiment. The latter analysis confirms our quantitative analysis, *i.e.*, the fact that the file level is the most important context level and that AcronymFinder does not help a lot when splitting and expanding source code identifiers and that in some cases AcronymFinder could be misleading by providing too generic expansions that do not fit in the context of the abbreviated identifiers. Some subjects (eight) found that the function context level was the most useful for them; which can be explained by the fact that some functions contain enough hints/indicators in their comments or body about what the substrings composing the identifier could be. This observation can be also justified by the presence of the possible expansions corresponding to the identifier in these functions (their comments or–and body), which makes it sufficient for subjects in these cases to figure out the correct splitting and expansion at the level of function.

Concerning **Q2**, Fig. 6 shows that eight subjects found that their experience in C programming was totally useless when splitting and expanding source code identifiers. Fourteen subjects found that their C experience was not very useful when doing such tasks. Seventeen subjects found that their C experience was a bit useful when performing such tasks. Only three subjects agreed that their C experience was helpful. Thus, the sum of subjects who disagree on the usefulness of C experience for such tasks (twenty two) is more than twenty *i.e.*, more than the number of subjects that found it a bit useful (seventeen) added to the number of subjects that found their C experience very useful (three). This graph confirms our quantitative findings regarding the non-usefulness of C experience when performing the identifier splitting and expansion.

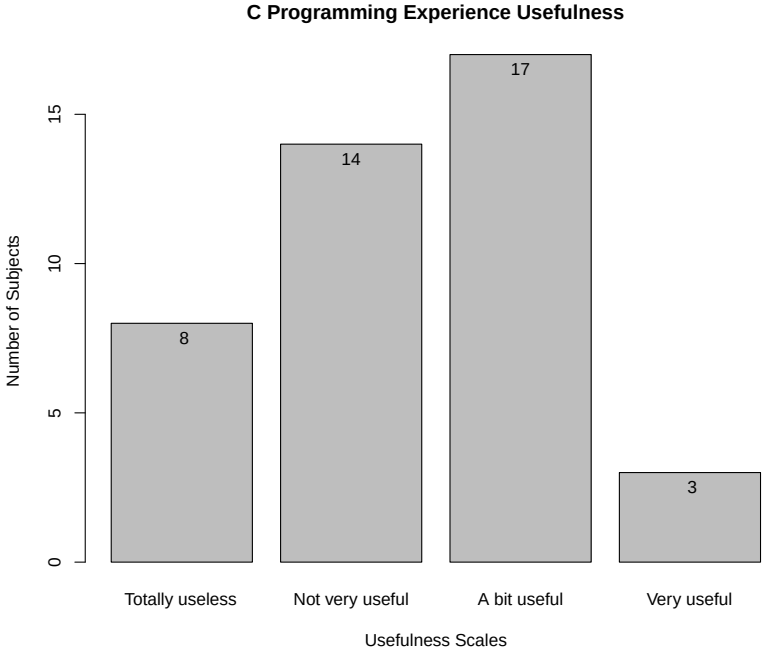


Fig. 6 Post-experiment Questionnaire: C Experience Usefulness.

Q3 concerns the knowledge of Linux utilities. As Fig. 7 shows, only two subjects found that their knowledge of Linux was helpful for them when splitting and expanding the identifiers given to them. Five subjects found it a bit useful. However, eleven subjects found that their knowledge in Linux was not very useful and twenty four subjects found it totally useless. Thus, the graph clearly confirms our quantitative results regarding the fact that the Linux knowledge is not useful when performing identifier splitting and expansion.

Regarding **Q4**, Fig. 8 shows that thirty three subjects found that the source code comments were helpful when performing their tasks. Only one subject found them a bit useful. Five subjects found the comments very useful and only three found the comments not very useful. Hence, the number of subjects that agreed on the usefulness of source code comments is significantly higher that the number of subjects who disagree on it. These reported numbers confirms our quantitative analysis about the usefulness of comments when splitting and expanding identifiers.

Q5 is related to the information provided in the procedure that we prepared and gave to the subjects before the experiment to give them an overview about the experiment and also the instructions to be followed when performing the tasks. As it can be noticed from Fig. 9, subjects agree on the usefulness of this information. In fact, twenty six subjects found this information helpful and sixteen found it very helpful.

Q6 concerns the impact of the information provided in the experiment procedure. Fig. 10 shows that no subject received a negative impact from the use of the experiment procedure when performing the tasks. Only one subject found that the procedure

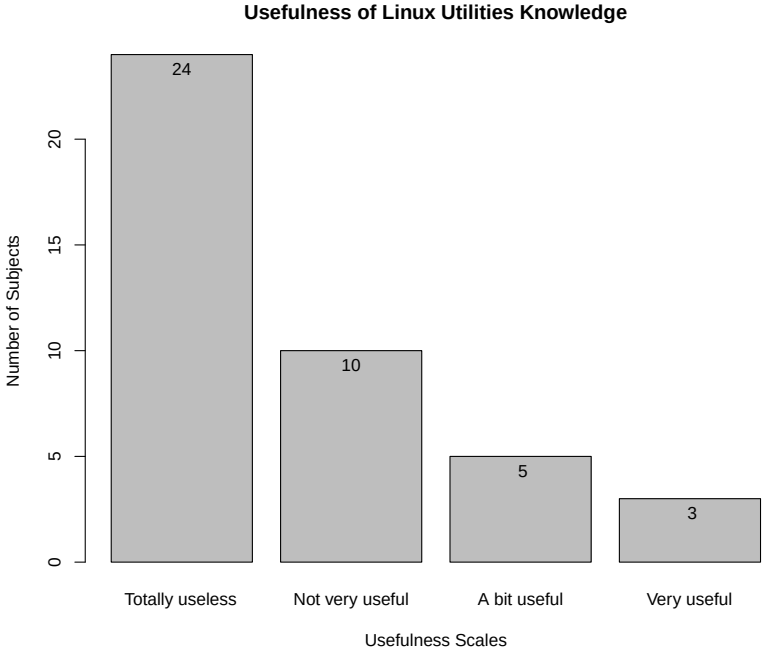


Fig. 7 Post-experiment Questionnaire: Linux Knowledge.

required too much concentration. Thirteen subjects have found that the information is correct. By correct, we mean that it has no impact on the subject's performance. However, twenty eight subjects found that it has a positive impact on them. These answers confirms the goal of the experiment procedure that is to make the experiment instructions clear for the subjects and avoid bias that can be introduced by not following the instructions mentioned in the procedure. As an example of bias that can be introduced if subjects did not follow the procedure of the experiment is the use of the Internet connection to search the expansions of the abbreviations and acronyms contained in the identifiers instead of using the AcronymFinder database embedded in the Web application designed for the purpose of the experiment.

We provide in the following part some observations that confirm both our quantitative findings and our post-experiment analysis. The first observation that we did based on our interview with the subjects is that when we provide subjects with function or file context level, in general, subjects do not try to understand the source code given to them when splitting and expanding identifiers. This observation is not the case for all subjects, but most of them seem to have such a trend: subjects try first to have an idea about what a function or a program is doing, they check the source code comments and then navigate through the source code to locate the identifier in question. Indeed, by locating all occurrences of an identifier in its corresponding source code, subjects try to search hints and clues that can help them map the identifier to its appropriate expansion. This observation could justify the non-significant effect of C experience on the subjects' performance, *i.e.*, on the F-measure (**RQ3** - Effect of co-factor C experi-

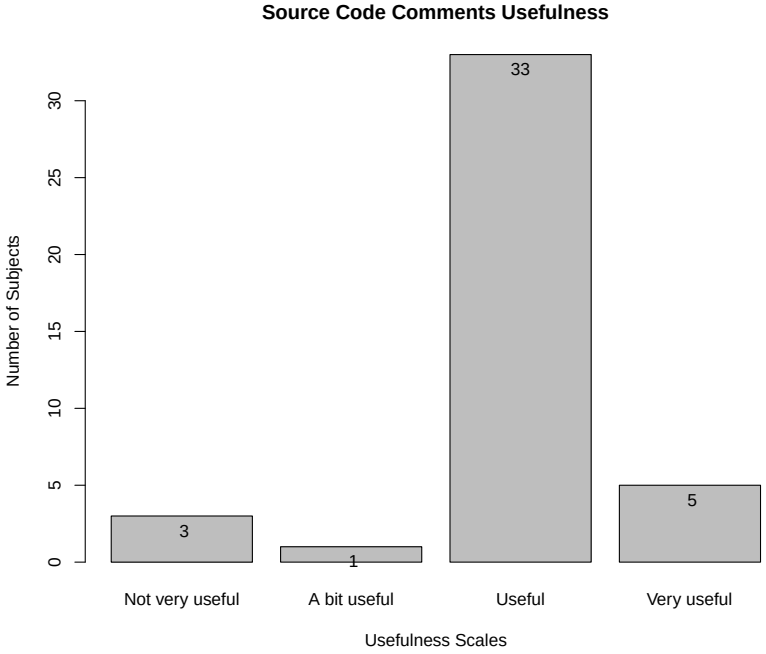


Fig. 8 Post-experiment Questionnaire: Source Code Comments Usefulness.

ence). When we provide subjects with the File plus AcronymFinder context level, we observed the following:

Some subjects used AcronymFinder to expand identifiers that are/contain acronyms and abbreviations. Yet, the expansions given by AcronymFinder can be so general and do not fit, in all cases, in the context that subjects were dealing with. Example of such identifiers is: *dbsm_start* that contains the acronym *dbsm*, which has been expanded by some subjects to *data*, *base*, *system*, *manager*, and *start* when using AcronymFinder. The latter expansion provided by subjects does not match the oracle of the identifier *dbsm_start* that is *decibel*, *per*, *square*, *meter*, and *start*. This mismatch is due to the fact that *data*, *base*, *system*, and *manager* is a generic expansion proposed by AcronymFinder to the acronym *dbsm_start*. In this case, either subjects had to search a more technical expansion to *dbsm* in AcronymFinder (via the option technical acronyms) and try to relate it to the given context or propose an expansion based on the source code only in case the ones proposed by AcronymFinder did not fit in the context. Examples of such a type can justify our quantitative findings *i.e.*, the fact that there is no statistical significant difference between File and File plus AcronymFinder levels (**RQ1** - Context relevance) even if AcronymFinder slightly increases the performances of subjects in some cases, *i.e.*, when subjects appropriately use it and relate its search results to the provided context.

According to our interview with the subjects at the end of the experiment, some of them did not use AcronymFinder very often because in some cases the source code file was sufficient to correctly expand the acronyms in question. This observation confirms

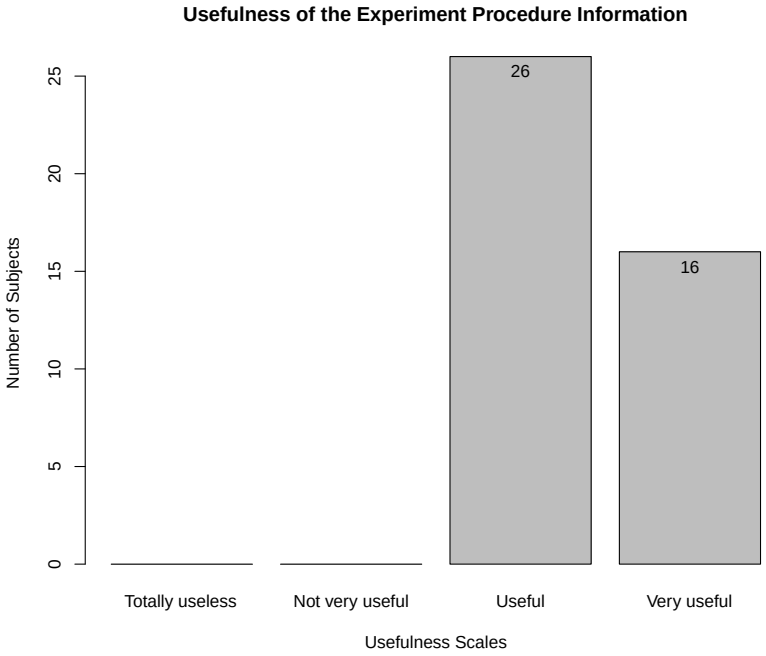


Fig. 9 Post-experiment Questionnaire: Procedure Information Usefulness.

what we stated in our previous observation (AcronymFinder is not always helpful) and strengths our quantitative results reported for (**RQ1** - Context relevance).

We also interviewed subjects concerning Linux knowledge and C experience. Regarding the Linux knowledge, most of subjects that have non-basic experience on Linux argue that their knowledge does not help them a lot when splitting and expanding identifiers as each task was related to a specific application and hence the expansion of identifiers requires a specific knowledge about the application domain. In fact, subjects argue that their knowledge in Linux was helpful only for few known identifiers such as *ptr* for pointer and *cpy* for copy. This observation confirms our quantitative findings, *i.e.*, the non-significant effect of Linux knowledge on subjects' performances (**RQ3** - Effect of co-factor Linux knowledge).

Concerning C experience, subjects (with non-basic experience) reported during our interview that their knowledge in C does not intervene when performing identifier splitting and expansion tasks. In fact, some subjects were experts in C but found difficulty in mapping the identifiers to their corresponding expansions. This category of subjects argue the difficulty of such tasks by the fact that they require a knowledge about the domain of the programs from where identifiers were sampled rather than skills in C programming. This statement confirms our quantitative findings, *i.e.*, the non significant effect of C experience on the subjects' performance (**RQ3** - Effect of co-factor C experience). It also show the relevance of domain knowledge to perform such tasks.

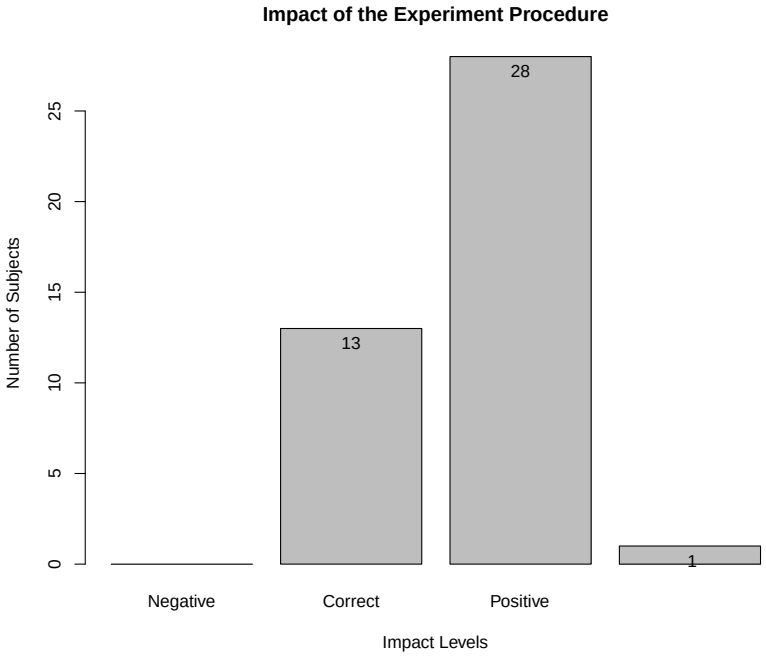


Fig. 10 Post-experiment Questionnaire: Impact of the Procedure Experiment Information.

Our interview to the subjects reveals also the importance of having a good level of English when performing identifier splitting and expansion tasks. In fact, subjects who are native speakers of English or who are, in general, good in English benefitted from their English knowledge as at least they did not deploy any effort to recognize full English words contained in identifiers. However, subjects who are not good in English face difficulties when trying to recognize terms composing identifiers. This observation confirms our quantitative findings for (**RQ3** - Effect of co-factor English knowledge). More precisely, it confirms the quantitative results reported in Table 15.

Another observation is related to the time allocated to perform the tasks. As we already mentioned, we did not impose any time constraint on subjects when performing the experiment. Yet, we observed two categories of subjects. The first (of few subjects) took time to perform the tasks, this category of subjects spend two to three hours *i.e.*, more time than what is estimated for the experiment (two hours). This category of subjects tries to carefully read the source code, navigate through it and use the external information, if any, is available. The other category spend one hour to one hour and thirty minutes, which is less than the time estimated to perform the tasks. This category probably tries to quickly read the source code and its comments and use the external information, if any. This observation is of practical importance as it clearly reveals the amount of time that such tasks require. In fact, even one hour and thirty minutes for forty identifiers is an important amount of time. Thus, identifier splitting and expansion is time consuming and challenging and hence it should be automatized to let developers, maintainers and Information Retrieval techniques fully benefit from it.

Also, subjects found that identifiers containing acronyms are, in general, more difficult than identifiers containing abbreviations to split/expand. Moreover all subjects agreed that, when context is available, the experiment tasks required concentration. Thus, identifier splitting and expansion is both time and effort consuming.

In summary, we can do the following conclusions:

- Identifier splitting and expansion is time and effort consuming. Hence, it is a challenging problem to tackle and automatize.
- Context and domain knowledge are relevant for identifier splitting and expansion. This statement confirms findings of previous work [GPAG11] that investigated identifier context when performing similar tasks.
- If manually performed either for research purposes or in companies, identifier splitting and expansion should be done by trainees (either female or male) having a good level of English.

4.1 Use of additional context: manual vs. automatic splitting

Results of this study showed that the presence of contextual information increases the performance of experiment subjects when performing identifier splitting and expansion tasks. A previous paper [GPAG11] indicated that algorithms for automatic identifier splitting and expansion benefit of contextual information. While developers use contextual information to understand the identifier domain/semantics or to seek words similar to terms contained in the identifier, automatic splitting algorithm try to match identifier terms onto other terms available in the context. Specifically, what the paper by *Guerrouj et al.* [GPAG11] showed was that:

- a small context (*e.g.*, function level) does not provide much useful information to the automatic identifier splitting and expansion tools;
- the performance of automatic splitting and expansion tools improves when an application-level context is available, *i.e.*, when using all possible terms in the application dictionary;
- performance of identifier splitting/expansion tools improves even further when additional knowledge—*e.g.*, known acronyms or abbreviations—are provided to the algorithm.

In summary, results of this paper not only provide indications about how and where developers find information to split and expand identifiers, but also further justifies the usage of contextual information in automatic splitting and expansion algorithms such as TIDIER [GPAG11], Normalize [LB11], and Samurai [EHPVS09]. Nevertheless, an important difference is that while automatic splitting/expansion could potentially benefit of the largest possible context—*e.g.*, application level—when a program comprehension task is performed by a human, the context must be kept relatively small to avoid information overload, i.e., while a file level context could be feasible, an application-level context could be too wide.

5 Threats to Validity

Threats to **internal validity** concern any confounding factor that could influence our results. One example of internal validity threats is the learning and fatigue effects. This

threat is addressed in our experiment by using different treatments for each group in an order that prevents fatigue or learning effects to be confounded with our treatment.

Another threat could be the one related to the subjectivity of the subjects when answering the pre-experiment questionnaire. For example, when we ask subjects to evaluate their level of English from Bad to Excellent, we are not sure if the provided answer is exactly the one corresponding to the level of subjects as we do not classify them according for example to their grades in English and this is due to the fact that even the grade do not usually reflect the real level/knowledge of a person.

To tackle the selection threat that is related to the variation in human performance, we identified a number of blocks to which the students were assigned. These blocks are based on the pre-experiment questionnaire. Students were selected from the different blocks to obtain a stratified random sampling over the different groups.

Another internal validity threat is the diffusion or imitation of treatments. This threat was also limited by controlling the experiment and preventing the access to the experiment material outside the experiment hours by other groups members. Also, although participants were aware of the lab objectives, they did not know exactly what hypotheses were tested.

Another internal threat could be due to the subjectivity of the manual building of the oracle and to the possible biases introduced by manually splitting identifiers. To limit this threat, the oracle was produced using a consensus approach (one author proposed an identifier split, which was then verified and validated by a second author). In a few cases, disagreements were discussed among all the authors. We adapted this approach in order to minimize the bias and the risk of producing erroneous results. This decision was motivated by the complexity of identifiers, which capture developers domain and solution knowledge, experience, personal preference, etc., thus, it is non trivial to decode the true meaning of identifiers in some cases.

Threats to **construct validity** concern the relation between the theory and the observation. This threat is mainly due to mistakes in the oracle. We cannot exclude that errors are present in the oracle. As the intent of the oracle is to explain identifiers semantics, we cannot exclude that some identifiers could have been splitted and expanded in different ways by the developers that originally created them. To limit this threat, we built context-dependent oracles using different sources of information, such as comments, source code, and online documentation.

Threats to **conclusion validity** are concerned with issues that affect the ability to draw the correct conclusion about relations between the treatment and the outcome of the experiment.

Proper tests were performed to statistically reject the null hypotheses. In particular, we used non-parametric tests, which do not make any assumption on the underlying distributions of the data such as the Cliffs delta tests. Also, we used the Wilcoxon unpaired test. ANOVA tests were used for the analysis of co-factors. we dealt with problems related to performing multiple Cliffs delta and Wilcoxon tests using the Holms correction procedure which is less restrictive than Bonferroni adjustment.

Threats to **external validity** concern the possibility of generalizing our results. To make our results as generalizable as possible, we selected our sample of identifiers from a large set of open-source project. We only consider C programs rather than Java program because it has already been proven that contextual identifier splitting/expansion is more significant for C programs than Java ones [NLP+10]. This is due to the fact that Java identifiers mostly adhere to naming conventions and are built using the Camel Case convention and, quite often, using complete English words rather than abbrevi-

ations and/or acronyms. Instead, the usage of a more complex splitting/expansion is particularly useful for programming languages that use short identifiers (*e.g.*, C and C++).

The subjects that performed the experiment form a population that includes a variety of training level (from bachelor to postdoctoral) and can be considered representative of a young developer population hired by a company.

6 Related Work

This section describes related work concerning (i) the role of textual information on program comprehension and software quality and (ii) approaches and tools for the automatic splitting and expansion of identifiers.

6.1 Role of Textual Information on Program Comprehension and Software Quality

A large body of work shows the paramount role of identifiers and comments in program comprehension and software quality. Among these contributions, we state the following:

Takang *et al.* [TGM96] empirically studied the role of identifiers and comments on source code understanding. They compared abbreviated identifiers to full-word identifiers and uncommented code to commented code. The results of their study showed that commented programs are more understandable than non-commented programs and that programs containing full-word identifiers are more understandable than those with abbreviated identifiers.

The main finding of the study on identifiers by Anquetil *et al.* [AL98] was the existence of "hard-terms" that encode core concepts.

Caprile and Tonella [CT99] performed an in-depth analysis of the internal structure of identifiers. They showed that identifiers are an important source of information about system concepts and that the information they convey is often the starting point of program comprehension.

Other researchers [CT00, MMM03] assessed the quality of identifiers, their syntactic structure, plus the information carried by the terms that compose them.

Deißenböck *et al.* [DP05] provided a set of guidelines to produce high-quality identifiers. With such guidelines, identifiers should contain enough information for a software engineer to understand the program concepts.

Lawrie *et al.* [LMFB07] attempted to asses the quality of source code identifiers. They suggested an approach, named QALP (Quality Assessment using Language Processing), relying on the textual similarity between related software artifacts. The QALP tool leverages identifiers and related comments to characterize the quality of a program. The results of their empirical study indicated that full words as well as recognizable abbreviations led to better program understanding. This work suggested that the recognition of words composing identifiers, and, thus, of the domain concepts associated with them could contribute to a better comprehension.

Binkley *et al.* [BDLM09] investigated the use of the identifier separators, namely the Camel Case convention and underscores in program comprehension. They found that the Camel Case convention led to better understanding than underscores and, when subjects are properly trained, that subjects performed faster with identifiers built using the Camel Case convention rather than those with underscores.

Sharif and Maletic [SM10] replicated this study using an eye-tracking system. The results of their study showed that subjects recognized identifiers that used the underscore notation more quickly. They also reported that there is no difference in terms of accuracy between the CamelCase and underscore style.

Overall, prior works reveal that identifiers represent an important source of domain information, and that meaningful identifiers improves software quality and reduce the time and effort to acquire a basic comprehension level for any maintenance task.

6.2 Approaches for Identifier Splitting and Expansion

Stemming from Deißenböck and Pizka's observation on the relevance of identifiers' terms for program comprehension, several approaches have been proposed to split and expand source identifiers. Examples of such works are the CamelCase, Samurai, TIDIER and Normalize. The simplest, widely-adopted CamelCase technique is often sufficient to accomplish software evolution tasks, see for example [DGPA11]. CamelCase is based on the following simple rules:

RuleA: Underscore, structure, and pointer access, as well as special symbols are replaced with the space character.

RuleB: Identifiers are split where terms are separated using the Camel Case naming convention. For example, *bugDescription* is split into *bug* and *Description* while *getBugID* is split into *get*, *Bug* and *ID*.

RuleC: When two or more upper case characters are followed by one or more lower case characters, the identifier is split at the last-but-one upper-case character. For example, *SVNCommit* is split into *SVN* and *Commit*.
Overall, CamelCase splitting algorithm leaves same-case composite identifiers such as *USERID* and *currentversion* unaltered.

CamelCase strategies do not use contextual information, which use is at the basis of Samurai [EHPVS09]. In a sense Samurai can be thought as a clever CamelCase guided by global and local knowledge coded into frequency tables. Indeed, Samurai local table is built by mining terms in the program under analysis while the global table is made by mining the set of terms in a large corpus of programs.

A truly context-sensitive identifier split and expansion approach is TIDIER [GPAG11]. TIDIER uses high-level and domain concepts captured into multiple dictionaries. Dictionaries can be context and application–or domain dependent and can also include acronyms and abbreviations knowledge. TIDIER is inspired by speech recognition techniques (it uses a modified version of Herman Ney dynamic time warping algorithm [Ney84]) to compute a string-edit distance between dictionary terms and a given identifier.

More recently, a machine-translation algorithm based on n-gram language models have been proposed to accurately split and expand identifiers [LBM10, LB11]. The core part of the expansion algorithm is based on a machine translation technique, namely maximum coherence model based on co-occurrence data that capture local context information. The heart of normalization is a similarity metric computed from co-occurrence data, exploited to select the best candidate among several possible expansions.

We share with this previous work the assumption that identifier splitting and expansion is essential for program comprehension; we also agree on the importance of

identifier context. Yet, in this work, we show the extent to which contextual information can support developers when performing program understanding tasks. We assume, a small context (single line or part of a line) will not provide much hints: a context as large as a complete application may be overwhelming and not useful due to information overloading. We thus focused this empirical investigation on function, file, and file plus external documentation context levels. In fact, we believe a not too big file but not too small function are still manageable as most of the time a C function fits into a screen and a file can be reasonably traversed and inspected.

7 Conclusion

This paper reported a controlled experiment that investigated the effect of one of the practical tasks in software maintenance and evolution, that is the splitting and expansion of source code identifiers. More specifically, it revealed the extent to which a source-code context could be helpful when splitting/expanding source code identifiers. The experiment involved human participants where the intellectual level ranges from Bachelor to Post-Doc. The study focused on two types of contexts: internal and external contexts. Internal context: functions and source code files, are the context readily available to any evolution task. External context in our setting involves external documentation. More precisely a thesaurus of acronyms and abbreviations.

We randomly sampled a set of fifty identifiers from a corpus of C programs and asked forty two subjects to split and expand identifiers. The splittings/expansions provided by subjects were compared to TIDIER performance, a recent contextual approach that splits and expands identifiers based on contextual information.

We reported evidence of the usefulness of context for both identifier splitting and expansion. In particular, we found that the file-level context is more helpful than function-level information. Moreover, external information does not significantly impact identifier splitting and expansion. Results also showed that the accuracy of the this comprehension task depends mainly on subject's level of English. C experience does not impact the task a lot. Linux expertise, gender, and program of studies do not intervene when splitting/expanding identifiers. Finally, we introduce a split-expansion entropy to model the difficulty in the split-expansion of identifiers. Our findings supports that a higher entropy does not imply a lower attained split-expansion accuracy as the richer context information balance the wider is the set of possible split terms and expansions. In computing the entropy we used dictionaries extracted from the functions and files (our contexts) as well as other linguistic sources (*e.g.*, Wordnet dictionary).

Correlation value are not very high thus supporting the conjecture that subjects experience, knowledge and linguistic ability is not necessarily captured by the function or file level contexts as developers have a much richer knowledge.

We believe the results that we provide in this paper and the conclusions that we draw provide useful insights to practitioners and researchers alike and confirm our beliefs of the importance of contextual information in program comprehension and the need to promote meaningful identifiers avoiding cryptic abbreviations or user-defined acronyms not known by others. The context is helpful not only to humans but also to automatic tools and Information Retrieval-techniques that use such a process of identifier splitting and expansion. Knowing the levels of interests when performing such a task could reduce the time and effort that a developer will spend trying to investigate many sources of information to map identifiers to their corresponding domain concepts

and hence the time and effort of understanding programs. These later benefits concern also the automatic tools that attempt to explore context. In fact, current or future identifier expansion tools that are dealing with context could use such levels of context to enhance their splitting/expansion accuracy.

Future work will investigate the impact of investigated levels on other comprehension tasks. The ultimate goal is to see if the same levels will be useful for other tasks and thus help not only developers understand programs with less effort and time but also increase the accuracy of automatic tool that deal with the studied program comprehension tasks.

Acknowledgment
This research was partially supported by the Natural Sciences and Engineering Research Council of Canada (Research Chairs in Software Evolution and in Software Patterns and Patterns of Software) and by Antoniol's Individual Discovery Grant.

Data
All artifacts (releases, class diagrams, graph representations) used in this work can be downloaded from the SOCCER laboratory Web server, under the Software Evolution Repository (SER) page, accessible at `http://web.soccerlab.polymtl.ca/SER/IdentifierContext_ProgramComprehension`.

References

[AL98] N. Anquetil and T. Lethbridge. Assessing the relevance of identifier names in a legacy software system. In *Proceedings of CASCON*, pages 213–222, December 1998.

[BCR94] V. Basili, G. Caldiera, and D. H. Rombach. *The Goal Question Metric Paradigm Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

[BDLM09] David Binkley, Marcia Davis, Dawn Lawrie, and Christopher Morrell. To camelcase or under_score. In *The 17th IEEE International Conference on Program Comprehension, ICPC 2009, Vancouver, British Columbia, Canada, May 17-19, 2009*, pages 158–167. IEEE Computer Society, 2009.

[CPM+00] Wohlin C., Runeson P., Host M., Ohlsson M.C., Regnell B., and Wesslen A. *Experimentation in Software Engineering - An Introduction*. Kluwer Academic Publishers, 2000.

[CT99] B. Caprile and P. Tonella. Nomen est omen: Analyzing the language of function identifiers. In *Proc. of the Working Conference on Reverse Engineering (WCRE)*, pages 112–122, Atlanta Georgia USA, October 1999.

[CT00] B. Caprile and P. Tonella. Restructuring program identifier names. In *Proc. of the International Conference on Software Maintenance (ICSM)*, pages 97–107, 2000.

[DGPA11] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol. Can better identifier splitting techniques help feature location? In *Proc. of the International Conference on Program Comprehension (ICPC)*, pages 11–20, Kingston, 2011.

[DP05] F. Deißenböck and M. Pizka. Concise and consistent naming. In *Proc. of the International Workshop on Program Comprehension (IWPC)*, May 2005.

[EHPVS09] Eric Enslen, Emily Hill, Lori L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009, Vancouver, BC, Canada, May 16-17, 2009*, pages 71–80, 2009.

[GK05] Robert J. Grissom and John J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2nd edition edition, 2005.

[GPAG11] L. Guerrouj, M. Di Penta, G. Antoniol, and Y. Gaël Guéhéneuc. Tidier: An identifier splitting approach using speech recognition techniques. *Journal of Software Maintenance - Research and Practice*, page 31, 2011.

[Hol79] Sture Holm. A simple sequentially rejective Bonferroni test procedure. *Scandinavian Journal of Statistics*, 6:65–70, 1979.

[JM01] Maletic J.I. and MarcusA. Supporting program comprehension using semantic and structural information. In *Proc. of 23rd International Conference on Software Engineering*, pages 103–112, Toronto, 2001.

[LB11] Dawn Lawrie and David Binkley. Expanding identifiers to normalize source code vocabulary. In *Proc. of the International Conference on Software Maintenance (ICSM)*, pages 113–122, 2011.

[LBM10] D.J. Lawrie, D. Binkley, and C. Morrell. Normalizing source code vocabulary. In *Proc. of the Working Conference on Reverse Engineering (WCRE)*, pages 112–122, 2010.

[LFB06] Dawn Lawrie, Henry Feild, and David Binkley. Syntactic identifier conciseness and consistency. In *Sixth IEEE International Workshop on Source Code Analysis and Manipulation Philadelphia Pennsylvania USA*, pages 139–148, Sept 27-29 2006.

[LMFB06] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. What's in a name? a study of identifiers. In *Proceedings of 14th IEEE International Conference on Program Comprehension*, pages 3–12, Athens, Greece, 2006. IEEE CS Press.

[LMFB07] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering*, 3(4):303–318, 2007.

[MMM03] Ettore Merlo, Ian McAdam, and Renato De Mori. Feed-forward and recurrent neural networks for source code informal information analysis. *Journal of Software Maintenance*, 15(4):205–244, 2003.

[MV95] A.V. Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *IEEE Computer*, pages 44–55, 1995.

[Ney84] Herman Ney. The use of a one-stage dynamic programming algorithm for connected word recognition. *IEEE Transactions on Acoustics Speech and Signal Processing*, 32(2):263–271, Apr 1984.

[NLP+10] N.Madani, L.Guerrouj, M.Di Penta, Y-G.Guéhéneuc, and G.Antoniol. Recognizing words from source code identifiers using speech recognition techniques. In *Proceedings of the Conference on Software Maintenance and Reengineering*, pages 69–78. IEEE, 2010.

[SBE83] E. Soloway, J. Bonar, and K. Ehrlich. Cognitive strategies and looping constructs: An empirical study. *Commun. ACM*, pages 853–860, 1983.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 625–56, 1948.

[SM10] B. Sharif and J.I. Maletic. An eye tracking study on camelcase and under_score identifier styles. In *Proceedings of the International Conference on Program Comprehension*, pages 196–205, June-July 2010.

[TGM96] Armstrong Takang, Penny A. Grubb, and Robert D. Macredie. The effects of comments and identifier names on program comprehensibility: an experiential study. *Journal of Program Languages*, 4(3):143–167, 1996.