# Is This Code Written in English? A Study of the Natural Language of Comments and Identifiers in Practice

Timo Pawelka
Technische Universität München
pawelka@in.tum.de

Elmar Jürgens
CQSE GmbH, Garching b. München, Germany
juergens@cqse.eu

***Abstract*—Comments and identifiers are the main source of documentation of source-code and are therefore an integral part of the development and the maintenance of a program. As English is the world language, most comments and identifiers are written in English. However, if they are in any other language, a developer without knowledge of this language will almost perceive the code to be undocumented or even obfuscated. In absence of industrial data, academia is not aware of the extent of the problem of non-English comments and identifiers in practice. In this paper, we propose an approach for the language identification of source-code comments and identifiers. With the approach, a large-scale study has been conducted of the natural language of source-code comments and identifiers, analyzing multiple open-source and industry systems. The results show that a significant amount of the industry projects contain comments and identifiers in more than one language, whereas none of the analyzed open-source systems has this problem.**

## I. Introduction

Source-code comments and identifiers are very important in the development and maintenance of a program. Comments are, after the code itself, the main source of documentation [1] and several experiments showed that commented code is easier to understand than code without comments [2], [3]. Identifiers, too, are of paramount importance for the readability of source-code, as approximately 70% of it consists of identifiers [4].
The lingua franca of programming is English, which is especially important given that the development and the maintenance of programs is often done by developers from many different countries. If a significant amount of comments or identifiers is not in English, it can lead to complications. For example, a person who is not in command of the language used, will find the code near to being obfuscated or similar to having no comments at all or incomprehensible abbreviations for identifiers. This impedes code comprehension and consequently the programming progress.
To date, however, it is not clear how common or widespread the problem of non-English comments and identifiers actually is. Regarding that, it is interesting whether there are notable differences between open-source and industry projects. For open-source in particular, it is important that all contributing parties find a common ground regarding the used natural language. Projects developed by a company may not have as many languages barriers between programmers, but it can quickly become a problem if the development or maintenance is, for example, outsourced to another country.
To shed light on that matter, we conduct a study that analyzes the comments and identifiers of multiple open-source and industry projects. In order to identify the languages of both the comments and the identifiers, we implement and evaluate an approach which uses a combination of existing methods in the field of language identification (LI). The results of the identification process are then compared and evaluated, leading to the conclusion that the problem of non-English comments and identifiers is not rare, but in this study actually exclusively occurring in the industry projects.

**Research Problem** Comments and identifiers in different languages are problematic in a program, especially if the development is international, e.g because of outsourcing. No studies have yet been conducted to determine how often it practically occurs that one project contains comments or identifiers from multiple natural languages and if there are differences between open-source and industry code.

**Contribution** This paper makes two main contributions. First, we developed and evaluated an approach for identifying the natural language of source-code comments and identifiers — called **Lisanc**. Second, we conducted a study that analyzes 13 open-source projects and 10 industry projects. Our study shows that non-English comments and identifiers do occur in practice, but for the analyzed systems actually exclusively in the industry projects.

**Outline** The paper follows a common structure. Please note, however, that the evaluation consists of two different parts. In IV, the language identifier we propose gets evaluated w.r.t. precision and recall. In V, we analyze real world software for comments and identifiers language distributions.

## II. Related Work

To the best of our knowledge, this is the first paper to look at the occurrence of non-English comments and identifiers.

This work was partially funded by the German Federal Ministry of Education and Research (BMBF), grant Q-Effekt, 01IS15003A. The responsibility for this article lies with the authors.

### A. Quality of Comments and Identifiers

Other papers have investigated the quality of comments and identifiers; for example, the work of Steidl et al. [5], Khamis et al. [6] and Tan et al. [7], which are focusing on the quality of comments or the work of Deissenböck and Pizka [4] and Anquetil and Lethbridge [8], where both emphasize the importance of well chosen names for identifiers. However, no studies have explicitly examined the extent of non-English comments and identifiers.

### B. Language Identification for Comments and Identifiers

In order to identify the language of the comments and identifiers, we have used several approaches from the well-researched field of language identification. The general strategy for automatic language identification is to create language models during a training phase and compare the input text against them. There are multiple different approaches to this task. The most common utilize the n-gram approach proposed by Cavnar et al. [9]. This method splits the input text into its n-grams and uses their frequency models to find the nearest, and therefore most probable, language by comparing it to the n-gram models of training data. Variants of this approach include Bayesian Decision Rules [10] or Markov Models [10].
Even though there are several proposals regarding the task of language identification, many problems, like the general short length of comments and identifiers, remain. Other studies have pointed out that general LI approaches produce good results for long texts (a couple of hundred characters), but the accuracy quickly deteriorates with shorter lengths [11], [12]. There are approaches that can yield better results with short text than others [13], but the issue still remains, especially for very short text with a length below 50 characters. This is important to note because source-code identifiers are naturally short. The length distribution for comments tends to be more mixed, varying between projects, but most comments generally have fewer than 100 characters.
In the case of non-English language in code, comments or identifiers are often multilingual. This is because there are terms, definitions, external sources, etc., that have been defined in English. Many LI approaches assume that the given text is monolingual. Earlier research has tried to deal with this problem [14], but in cases where the proportion of language A to language B is approximately 1:1, it is impossible to decide which language to pick.

## III. Approach

### A. Overview

This section presents our approach for the language identification of comments and identifiers, called **Lisanc**. It comprises three sequential steps. The first is the *extraction* of all unique comments and identifiers from the project, followed by the *normalization* where the comments and identifiers are modified, if they contain certain features that have a negative influence on the language identification. The third and final step is the actual *language identification* of every single comment and identifier. The general idea of the proposed LI approach and the main contribution is to aggregate and compare the LI results of multiple different tools and subsequently choose one.

```
/**
 * Licensed to the Apache Software Foundation
 * (ASF) under one or more contributor license
 * agreements. See the NOTICE file
 * distributed with this work for additional
 * information regarding copyright ownership.
 * ...
 */

//clase que guarda el estado de cada elemento
//en el servidor
public class Elemento {
   int id;
   float tamaño;
   Vector3f posición;
   Vector3f velocidad;

   /**
    * el constructor para el clase Elemento
    * @author garcia
    */
   public Elemento(int i, float x, float y,
       float z, float t) {
      id = i;
      tamaño = t;
      posición = new Vector3f(x, y, z);
      velocidad = new Vector3f();
   }

   //float tr(float f) {
   //   float a = Math.round(f * 100);
   //   return a / 100;
   //}

   //Función que forma un mensaje con todos
   // los parámetros del elemento
   String getUpdateAsMessage() {
      String mes = (tr(posición.x) +
          tr(posición.y));
      return mes;
   }
}
```

Fig. 1. Code example

### B. Description

Below is a more detailed description with the help of the running example in Figure 1.

*1) Extraction:* In this step, all comments and identifiers of the project under study are extracted by iterating through every file individually. ConQAT[1] is used to comb through the files and extract all comments and identifiers. Afterwards, copyright comments and commented-out code are filtered from the list of the extracted comments. An implementation of the classification methods for comments described in the work of

[1]https://www.conqat.org/

TABLE I
EXTRACTED AND NORMALIZED COMMENTS AND IDENTIFIERS

| *Type* | *Extracted* | *Normalized ("–" if nothing changed)* |
|---|---|---|
| Comments | `clase que guarda el estado de cada elemento en el servidor` | – |
| | `* el constructor para el clase Elemento`<br>`* @author garcia` | `el constructor para el clase Elemento garcia` |
| | `Función que forma un mensaje con todos los parámetros del elemento` | – |
| Identifiers | `Elemento` | – |
| | `Vector3f` | `Vector f` |
| | `posición` | – |
| | `velocidad` | – |
| | `getUpdateAsMessage` | `Update As Message` |

Steidl et al. [5] is used to achieve this. Copyright comments are identified by checking whether the comment contains either of the keywords "copyright" or "license". Commented-out code is detected by applying regular expressions for common code structures, such as loops; if a comment contains more than a certain amount of commented-out code, further described in Steidl et al. [5], it will be excluded. Additionally, comments and identifiers are only considered for extraction if they have a predefined minimum number of characters. The minimum length thresholds differ between comments and identifiers and are called $l_{comment}$ for comments and $l_{identifier}$ for identifiers. Note that, for comments, whitespace-characters count towards the total character count.
The column "Extracted" in Table I contains the result of the extraction of the comments and identifiers from the code of the running example in Figure 1. The copyright header and the commented-out code were ignored. For the identifiers the minimum length was set to eight and therefore only the identifiers which have more or equal than eight characters have been extracted. The length threshold here was set to a value where the effects are clearly visible.

*2) Normalization:* In the normalization process, comments and identifiers are modified to improve the results of the language identification. This involves removing certain characters or adding whitespace characters. Many of the following normalization follow closely parts of the works of Dit et. al. [15].
The following modifications are applied to every single identifier, in the following order:

- **Remove set or get**: The naming of setters and getters follows the pattern *setVariable* or *getVariable*. If the variable name is, for example, in German, the prefix set/get is still in English. In order to avoid a mixture of fixed expression and text, which can impede the LI, set and get are removed, if they are at the very beginning of the identifier.
- **Split CamelCase and Underscore**: Underscores are replaced with a space and words following the CamelCase pattern are split according to the following examples:
  - "CamelCase" → "Camel Case"
  - "UMLexample" → "UML example"
  - "Matrix123Test" → "Matrix 123 Test"
- **Remove non-Letter chars**: Characters that are not part of a word in any language are removed.

The modifications for the comments are as follows:

- **Remove DocTags**: e.g. @param, @author etc.
- **Split CamelCase and Underscore**
- **Remove non-Letter chars**

The results of the normalization of comments and identifiers from the running example are shown in the column "Normalized" in Table I. For example, "getUpdateAsMessage" was normalized to "Update as Message".

*3) Language Identification:* The proposed language identification process is based on a majority vote of multiple LI tools, with different or differently implemented LI methods. The basic idea is that the LI tools identify the language of every extracted comment or identifier of the project by itself first; every tool has then a two lists of results. One list containing the suggested languages for the comments and one for the identifiers. All of the results of the tools are then combined into two lists of suggested languages; again one for comments and one for identifiers.
Note that this is done separately for comments and identifiers. This means that the language results for comments will be combined only with other comment results. The same applies to identifiers. Also, the set of LI tools used for the comments differs from the set of tools for the identifiers. This is because of the differences between comments and identifiers (such as length), different sets of LI tools yield better results for comments than for identifiers and vice versa. More details, such as the list of the used tools and the their evaluations are provided in Section IV.
Taking the normalized comments and identifiers from Table I, the proposed language identification approach works as follows:

TABLE II
LANGUAGE COUNTERS: THE NUMBER OF LI TOOLS SUGGESTING A GIVEN LANGUAGE

| *Type* | *Id* | *es* | *en* | *de* | *un* | *pt* | *gl* |
|---|---|---|---|---|---|---|---|
| Comments | 1 | 6 | | | 1 | 1 | |
| | 2 | 7 | | | | 1 | |
| | 3 | 8 | | | | | |
| Identifiers | 4 | 3 | | | 1 | 1 | |
| | 5 | | 2 | 3 | | | |
| | 6 | 2 | | | 1 | | 2 |
| | 7 | 4 | | | 1 | | |
| | 8 | | 4 | 1 | | | |

TABLE III
LANGUAGE IDENTIFICATION RESULT

| *Type* | *Id* | *Preliminary Language* | *Final Language* |
|---|---|---|---|
| Comments | 1 | es | es |
| | 2 | es | es |
| | 3 | es | es |
| Identifiers | 4 | es | es |
| | 5 | de | un |
| | 6 | un | un |
| | 7 | es | es |
| | 8 | en | un |

The LI tools identify the language of every comment and identifier individually and the results are combined in such a way that every comment and identifier is tagged with a map of all proposed languages of the LI tools and their occurrences, called the **language counter**. Table II contains the resulting language counters for the running example. For example, for the comment with Id 2 in Table II, seven of eight tools propose that it is in Spanish, while one says that it is in Portuguese.
Next, every comment and identifier of the language counters is analyzed individually in order to choose one result. The method used is a constrained majority vote in which the language with the most votes is selected, but only if it fulfills the following conditions. Otherwise, it is labeled as **un**, which is short for **unknown**:

- **Minimum Agreement[%]**($x_{agree}$): A minimum percentage of the used tools must agree on the language with the most votes. For example for the comment with Id 1 in Table II, Spanish (es) is the language with the most votes for Id 1 and in total $\frac{6}{8} = 75\%$ of all LI tools agree on it.
- **Confidence:** This condition dictates that there can only be one language with the highest number of votes. The language counter for Id 6 contains two languages with the most votes, Spanish and Galician, thereby violating this condition.

TABLE IV
LANGUAGE DISTRIBUTIONS

| *Type* | *Language[%]* *es* | *en* | *de* | *un* |
|---|---|---|---|---|
| Comments | 100 | | | |
| Identifiers | 40 | 20 | 20 | 20 |

The column "Preliminary Language" in Table III is a possible outcome for the filtering step with a minimum agreement percentage and the confidence condition.
The next step applies only if the list containing all comments or identifiers has more than one entry. A new language counter is created that contains all previously chosen languages over the whole project (language or unknown) and their number of occurrences, called the **language distribution**; again, this is done separately for comments and identifiers. Table IV is the language distribution for the column "Preliminary Language" in Table III. The lists with the comments and the identifiers are then iterated again. Every entry that is not tagged as unknown is tested to determine whether its identified language fulfills the following condition; otherwise, the entry is re-labeled as unknown.

**Minimum Occurrence[%]** ($x_{occurr}$): This condition describes the minimum percentage amount a language must have in the language distribution mentioned above in order to be considered a viable result. Taking the language distribution from Table IV, setting $x_{occurr} = 25\%$ and applying it, results in the languages for the comments and identifiers in the column "Final Language" of Table III. The English and German entries have both been relabeled to unknown because the occurrence percentages of both languages (20%) do not exceed $x_{occurr}$ (25%).
This constraint aims to filter out those rare entries where a great majority or even all used LI tools agree on a wrong language. The case, where for example $2\%$ of all comments or identifiers are in fact in a different language is neglected with the described parameters, because the goal of this approach is to identify whether a project contains substantial amounts of non-English comments or identifiers.
This measure proved very useful in removing persistent false entries. Note that this measure is especially important for identifiers, as the probability of identifiers which are incomprehensible for developers is greater than for comments.
The parameters that differ between the language identification for comments and identifiers are:

- Different combination of tools.
- Different minimum Length: $l_{comment}$ or $l_{identifier}$.
- Different values for $x_{agree}$, $x_{occurr}$.

## IV. EVALUATION

In this section, we evaluate the quality of the proposed approach. In order to achieve this, the best individual values for the parameters are identified for both the comments and the identifiers LI.

TABLE V
USED LI-TOOLS

| *Name* | *Ref.* | *Source* |
|---|---|---|
| CelLang | | https://code.google.com/p/language-identification/ |
| CLD | | https://code.google.com/p/chromium-compact-language-detector/ |
| DictIdentify | [16] | http://mlcomp.org/programs/633 |
| GuessLanguage | | https://bitbucket.org/spirit/guess_language |
| JLad | | https://github.com/VivienBarousse/jlad |
| LangDetect | | https://code.google.com/p/language-detection/ |
| LangDetect2 | | https://github.com/feedbackmine/language_detector |
| LangId | [17] | https://github.com/carrotsearch/langid-java |
| LangId2 | | https://code.google.com/p/lang-id/ |
| LDig | [18] | https://github.com/shuyo/ldig |
| LingPipe | [19] | http://alias-i.com/lingpipe/index.html |
| JavaTextCat | | http://www.findbestopensource.com/product/javatextcat |
| TikaIdentifier | | https://tika.apache.org/ |

TABLE VI
BENCHMARK PROJECTS

| *Source (https://github.com/)* | *Natural languages* |
|---|---|
| schakko/zabos<br>crunkchilla/TPE<br>Andi17/Fallstudie | de,en |
| enrico200165/ev_tools<br>mrandisi/adi<br>filippomortari/VRMS_Project | it,en |
| dtFraca/Javamatik<br>Uvuros/iutvalence-java-mp-g1p5-2012-2013<br>AydenSekey/caesar | fr,en |
| Kayne/JavaZadania<br>piotr45/Projekt2<br>kryptoala/modul_3 | pl,en |
| IsraelCgutierrez/3D-Multiuser-Environment-Server<br>dennistu1994/ANGLE<br>frieser/formation | es,en |

The case study analyzing real world software is described in section V.

As described in Section III, the proposed LI approach utilizes multiple LI tools. Table V contains all of the tools that have been tested and evaluated.

### A. Study Objects

In order to determine the best parameters for the approach, it needs to be tested and evaluated against a benchmark, which is similar to the field it will be applied to. For that reason two benchmarks were created, one with comments and one with identifiers, by extracting and labeling all comments and identifiers of the projects shown in Table VI.

The projects for the benchmark were chosen because they all contain text written in two languages. They were found by using the GitHub code-search for terms like "variable" or "loop" in different languages.

The extraction and normalization described in Section III-B were used to get the comments and identifiers, with a minimum length of 5. This was set because words with fewer than five characters are prone to ambiguity; an example is the word *color*, which exists in multiple languages.

The languages of the comments and the identifiers were then identified by going through their lists and manually deciding in which language they were written in. If in doubt the comment or identifier was translated word by word with an online dictionary. If we were unable to determine the language in which an entry was written, the entry was excluded. The end result were two benchmarks one with 10354 comments and one with 7201 identifiers, with both containing entries in six different natural languages — English, German, French, Spanish, Polish and Italian.

### B. Evaluation Questions

**EQ 1** *Is there a combination of LI tools that is better than any single one?*

The first question is whether the proposed approach is indeed better suited for the problems stated in this paper than any of the presented tools.

**EQ 2** *What are the best combinations of LI tools and parameters for comments and identifiers?*

Having established that there are indeed combinations of tools that are better than any single one, it is important to determine which combinations are best. This is done separately for comments and identifiers, as it is possible that the best combination for one of them is not the best for the other.

*1) Study Design:* We answered the research questions with the following study design. Every tool presented in Table V is tested against the benchmarks introduced in Section IV-A. The $F_\beta$-measure [20] is used to compare the results. The $F_\beta$-measure describes a performance classification that weighs the recall $\beta$-times more than the precision. In this work, the precision is valued more highly than the recall. This is because it is more important to know with a high level of certainty whether a project contains a specific language than it is to be able to label every comment or identifier with a language. Because of that, $\beta$ is set to $0.5$, which causes the precision to be weighed twice as many times as the recall, but does not neglect the recall completely.

With the result over the benchmarks, the precision, recall and the $F_{0.5}$ can be calculated using the following formula:

$$F_\beta = \frac{(\beta^2+1)Precision_\mu Recall_\mu}{\beta^2 Precision_\mu + Recall_\mu}$$

Fig. 2. F-Measure over multi-class classification with micro-averaging [20].

In order to determine the best parameters and consequently the best combination of tools, a Java program was created that automatically checked every possible composition of

- $x_{agree}$, ranging from $10\% - 80\%$ with a step size of $10\%$, and
- $x_{occurr}$, ranging from $0\% - 5\%$ with a step size of $1\%$[2],

and finding the best LI tool combination for this tuple ($x_{agree}$, $x_{occurr}$).

The tuple ($x_{agree}$, $x_{occurr}$) is called **composition**, whereas a set of LI tools is called **combination**.

For every single composition out of the $8 * 6 = 48$, every possible tool combination is created and evaluated against the benchmark. This means that, at first, every combination of two LI tools — which can be described with the binomial coefficient $\binom{14}{2}$, with 14 LI tools in Table V — is created and evaluated and then every combination of three, and so on. This sums up to $\sum_{i=2}^{14} \binom{14}{i} = 16,369$ combinations. Then, the one combination of the $16,369$ with the highest $F_{0.5}$-measure is selected.

Altogether, this resulted in the evaluation of $16,369 * 48 = 785,712$ different combinations and parameters, which took about 33 minutes for the identifier-benchmark and 47 minutes for the comments benchmark.

Subsequently, all $48$ $F_{0.5}$-measures (one for every composition) were compared in order to determine the best overall combination of tools and parameters.

The minimum length restrictions for the identifiers and the comments were chosen with the following goal in mind. An average of at least $90\%$ of all comments and identifiers of a project should be extracted and their language identified. $90\%$ ensures that the language of a significant amount of comments and identifiers is identified and the errors caused by a small amount of characters are limited.

$l_{comment}$ and $l_{identifiers}$ are calculated by creating the comments and identifiers length distributions over all projects (Table XI and Table XII). These distributions contain the number of comments or identifiers with a length of one up to 100 characters. If, for example, the minimum length for identifiers is five, all identifiers with a length smaller than five are excluded. The minimum lengths are then calculated by testing which length is the smallest, such that the remaining number of all comments or identifiers is not smaller than the $90\%$ defined above. Here, five is the absolute minimum for the length because the probability of identifying the language correctly or even being able to do so with such few characters is low.

### C. Results

The results of all the single tools on both benchmarks can be seen in Table IX and Table X. The corresponding precision and recall are also listed to show the possible discrepancies between them.

Using the results of the single tools to determine the best composition of parameters and combination of tools for the comments or the identifiers results in Table VII for the parameters and Table VIII for the tool combinations; Lisanc$_{co}$ is the version of Lisanc used to identify the natural language of comments and Lisanc$_{id}$ for identifiers. The minimum length restrictions were calculated using Figure 3 and Figure 4. These figures represent the number of comments or identifiers for each length, ranging from one to 100. In each figure, the length with the maximum number of comments or identifiers is marked, as is the section that is excluded by the minimum length — the red area.

[2]Note that every value outside of the ranges is considered impractical or futile by testing and experience.

TABLE VII
BEST COMBINATION OF PARAMETERS FOR COMMENTS AND IDENTIFIER

| *Type* | $x_{agree}$*[%]* | $x_{occur}$*[%]* | *Minimum Length* |
|---|---|---|---|
| Comments (Lisanc$_{co}$) | 30 | 3 | 13 |
| Identifier (Lisanc$_{id}$) | 30 | 4 | 8 |

TABLE VIII
BEST COMBINATION OF TOOLS FOR COMMENTS AND IDENTIFIERS

| *Type* | *Tool combination* |
|---|---|
| Comments (Lisanc$_{co}$) | CelLangBayes, CelLangAdhoc, CLD, LangID, Jlad, LingPipe, JLad, DictionaryIdentify |
| Identifier (Lisanc$_{id}$) | CelLangAdhoc, CLD, LangID, LingPipe, DictionaryIdentify |

The results of Lisanc$_{co}$ and Lisanc$_{id}$ on their corresponding benchmarks are shown in Table IX and Table X, where all results are presented in descending order according to their $F_{0.5}$-measure.

**EQ 1** asks whether there is a combination of language identification tools for our proposed approach that scores a better $F_{0.5}$-measure than any single tool alone. This can be answered directly by looking at the Table IX and X. In both cases, the proposed approach yields significantly better results, not only for the precision, but also for the recall. Consequently, it has the best $F_{0.5}$-measures. Of course, this is an expected outcome, because Lisanc was specifically trained on this data set. Mitigation for this is discussed in section V-E. The differences between the precision and recall of our approach in comparison with the best values of the single tools, which are marked bold in Tables IX and X, are very notable. For the comments, the difference between the recall of Lisanc$_{co}$ and the best single tool (LingPipe) is only about 0.15 percentage points. For the precision, however, the difference is, with 99.25% for Lisanc$_{co}$, more than one percentage point better than for the best precision of a single tool; CelLangBayes with 98.15%. This is much, considering that the value of CelLangBayes is almost perfect. For the language identification of the identifiers it is the other way around. The LI tool CLD almost has the same precision as Lisanc$_{id}$, with 97.15% versus 97.35%, but there is a wide gap between the recall, of LangID, with 79.55% as the best single tool value, and 83.1% for Lisanc$_{id}$. Whereas the single tools have also good results for the language identification of

TABLE IX
RESULTS OF ALL TOOLS AND THE BEST LISANC COMBINATION ON THE COMMENTS BENCHMARK.

| *LI tool* | *Precision[%]* | *Recall[%]* | $F_{0.5}$ |
|---|---|---|---|
| Lisanc$_{co}$ | **99.25** | **96.15** | **0.9861** |
| CelLangBayes | **98.15** | 94.3 | **0.9736** |
| LingPipe | 96.4 | **96.0** | 0.9630 |
| LDig | 96.55 | 92.65 | 0.9574 |
| LangID | 96.15 | 93.95 | 0.9571 |
| CelLangAdhoc | 97.05 | 89.3 | 0.9539 |
| JLad | 95.75 | 92.0 | 0.9500 |
| LangID2 | 95.9 | 89.85 | 0.9464 |
| LangDetect | 94.5 | 91.15 | 0.9379 |
| CLD | 95.65 | 80.65 | 0.9224 |
| GuessLanguage | 96.25 | 78.15 | 0.9198 |
| DictionaryIdentify | 89.75 | 89.75 | 0.8975 |
| JavaTextCat | 94.05 | 72.8 | 0.8886 |
| LangDetect2 | 88.7 | 77.9 | 0.8630 |
| Tika | 89.45 | 65.05 | 0.8321 |

TABLE X
RESULTS OF ALL TOOLS AND THE BEST LISANC COMBINATION ON THE IDENTIFIER BENCHMARK.

| *LI tool* | *Precision[%]* | *Recall[%]* | $F_{0.5}$ |
|---|---|---|---|
| Lisanc$_{id}$ | **97.35** | **83.1** | **0.9413** |
| CelLangAdhoc | 88.65 | 68.35 | **0.8368** |
| LangID | 84.45 | **79.55** | 0.8343 |
| CLD | **97.15** | 53.3 | 0.8342 |
| CelLangBayes | 86.8 | 68.05 | 0.8226 |
| JLad | 83.8 | 68.15 | 0.8012 |
| LingPipe | 79.3 | 77.1 | 0.7886 |
| LDig | 74.5 | 57.2 | 0.7025 |
| LangID2 | 74.25 | 52.85 | 0.6868 |
| LangDetect | 67.3 | 56.6 | 0.6485 |
| DictionaryIdentify | 59.55 | 59.55 | 0.5955 |
| LangDetect2 | 61.5 | 39.45 | 0.5530 |
| GuessLanguage | 71.3 | 15.6 | 0.4163 |
| JavaTextCat | 49.8 | 19.6 | 0.3806 |
| Tika | 43.75 | 15.85 | 0.3237 |

comments, the main advantage of our approach is the language identification of identifiers. The values for both the recall and the precision are high enough to reliably identify the language of a significant amount of identifiers.

**EQ 2** is answered by the Tables VII and VIII. They represent the best composition of parameters and best combination of LI tools for the used benchmarks. Because the benchmarks consist of comments or identifiers those parameters and tool

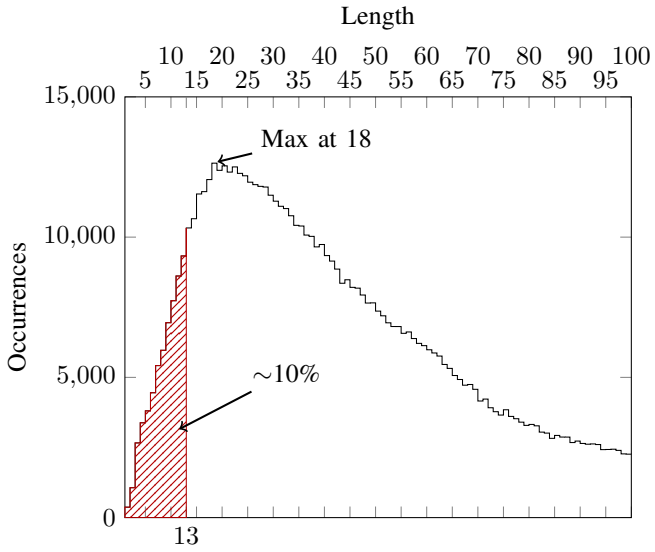



Fig. 3. Comment Length Distribution over all projects

combinations are calibrated for the best identification of source-code comments and identifiers.

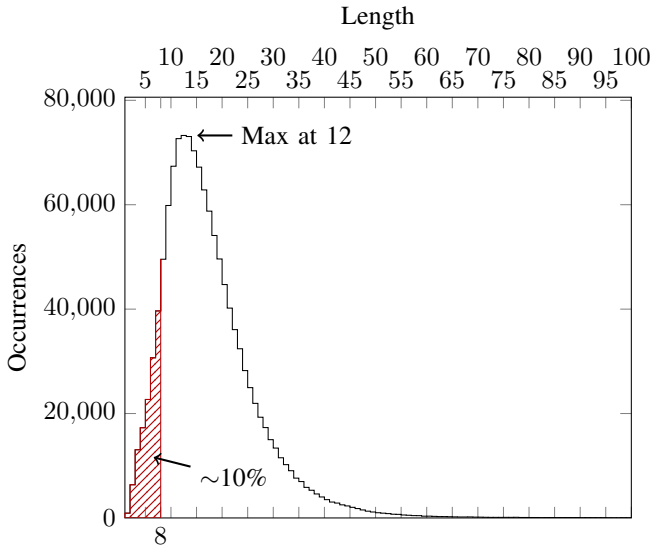



Fig. 4. Identifier Length Distribution over all projects

## V. CASE STUDY

### A. Research Questions

After describing and evaluating the approach for the language identification of source-code comments and identifiers, this section presents the application of this approach in practice.

The issues tackled are summarized with the following research questions.

**RQ 1** *To which extent do non-English comments and identifiers occur?*

The primary question is whether non-English comments and identifiers appear in projects regularly enough to matter in practice. This does not mean whether any non-English comments or identifiers can be found at all, but whether they constitute a significant part of the project.

**RQ 2** *Are there differences between the comments or identifiers of open-source and industry projects?*

Having established the fact that non-English comments and identifiers occur, it is interesting to compare projects developed with different backgrounds. As mentioned in Section I, there are differences between open-source and industry projects. Companies have a closed team of developers from at most few countries, whereas the the open-source development possibly involves people from all around the world.

**RQ 3** *Is there a difference between the distributions of non-English comments and identifiers?*

With this RQ we investigate whether the threshold for using non-English languages is higher for identifiers than for comments. This assumption is based on the fact that the main part of identifiers often consists of fixed expressions.

### B. Study Objects

The study objects used for answering the research questions can be divided into two groups. One consists of 13 open-source projects and the other consists of 10 industry projects. Table XI is a list of the analyzed open-source projects, with the numbers of unique comments and identifiers for each project. The industry projects are listed in Table XII.

TABLE XI
OPEN-SOURCE PROJECTS WITH NUMBER OF UNIQUE COMMENTS AND IDENTIFIERS

| *Name* | *Comments* | *Identifiers* | *LOC* |
|---|---|---|---|
| | **Java** | | |
| ConQAT | 18,409 | 15,627 | 326,197 |
| OpenJDK | 118,724 | 83,853 | 2,822,281 |
| JUnit | 886 | 3,358 | 38,540 |
| libGDX | 9,746 | 22,855 | 343,923 |
| Apache Storm | 1,435 | 6,706 | 129,964 |
| Apache Ant | 14,551 | 11,811 | 265,830 |
| | **C#** | | |
| CoreFX | 37,631 | 54,924 | 1,824,018 |
| SignalR | 1,681 | 5,090 | 68,498 |
| | **C/C++** | | |
| MongoDB | 88,524 | 87,078 | 2,634,062 |
| PhantomJS | 148,542 | 241,560 | 4,685,330 |
| VLC | 24,915 | 47,131 | 647,328 |
| XAMPP | 22,390 | 24,060 | 352,130 |
| FileZilla | 4,726 | 10,429 | 163,469 |
| | **492,160** | **614,482** | **14,301,570** |

TABLE XII
INDUSTRY PROJECTS WITH NUMBER OF UNIQUE COMMENTS AND IDENTIFIERS

| *Name* | *Comments* | *Identifiers* | *LOC* |
|---|---|---|---|
| | **Java** | | |
| System 1 | 6,350 | 22,397 | 404,550 |
| System 2 | 18,074 | 21,756 | 596,825 |
| System 3 | 6,121 | 7,547 | 107,613 |
| System 4 | 8,048 | 27,454 | 425,119 |
| System 5 | 10,278 | 12,609 | 441,181 |
| System 6 | 4,608 | 8,195 | 353,244 |
| System 7 | 9,261 | 12,868 | 256,904 |
| | **C#** | | |
| System 8 | 1,921 | 4,313 | 81,745 |
| System 9 | 20,069 | 29,752 | 541,522 |
| System 10 | 31,174 | 29,663 | 690,177 |
| | **115,904** | **176,554** | **3,641,976** |

### C. Study Design

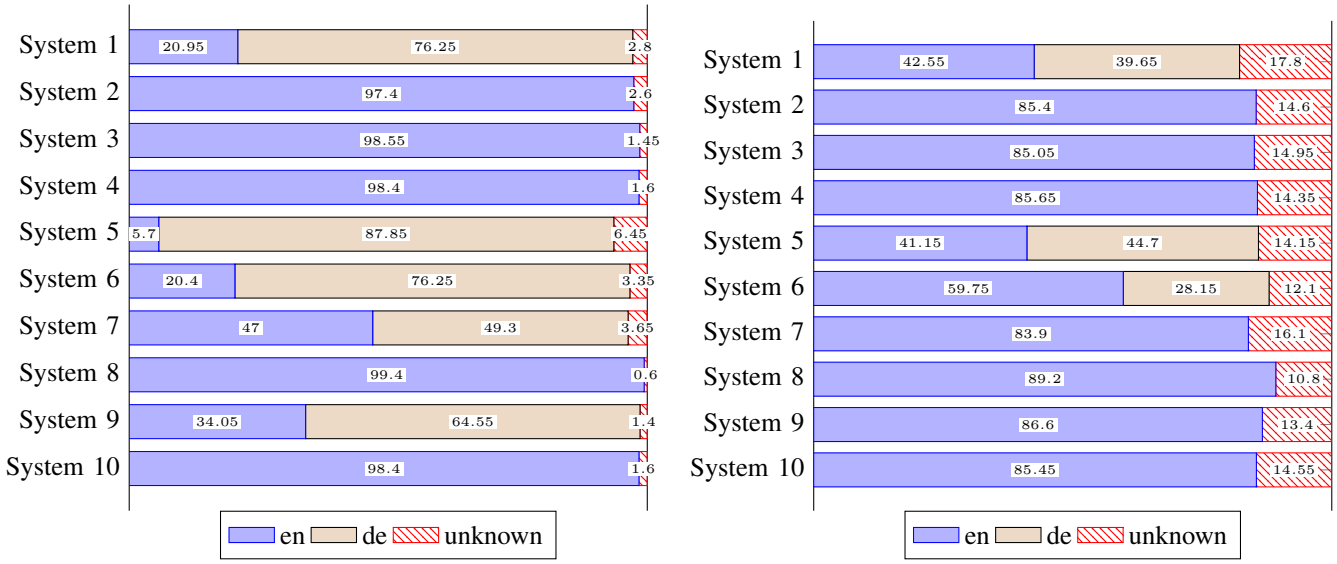
The research questions are answered with the help of the following study design. The approach described in Section III-B is used to create the results for every project listed in Table XI and Table XII. From these results, the language distributions for the comments and the identifiers of open-source or industry projects can be created. A language distribution consists of the relative frequencies of all occurring languages over all comments or identifiers.

### D. Results and Interpretation

The language distributions of the results for the open-source projects can be seen in Figure 6 and for the industry projects in Figure 5.

**RQ 1**. Figure 5a and Figure 5b both show projects that contain non-English comments and identifiers. There are five industry systems which contain a percentage of non-English comments from about 50% up to almost 90% and three industry systems with about 28% up to 45% non-English identifiers. Therefore RQ 1 can be positively answered: There are comments and identifiers written in a non-English language. In fact the percentages of non-English comments and identifiers are significant. There are cases where non-English comments and identifiers make sense; for example, if some expressions are unique to that language or that country, such as insurance values. However, this does not apply to all cases, which leads to the conclusion that the analysis of natural language of comments and identifiers should be included in the quality assessment of a software.

Regarding **RQ 2**, the differences between the analyzed open-source and industry systems, non-English comments or identifiers exclusively occur in the industry projects. In fact




(a) Comments language distribution

(b) Identifier language distribution

Fig. 5. Results of the industry projects

half of the industry systems contain non-English comments or identifiers, whereas none of the analyzed open-source systems contain any other natural language than English. Of course, there is a possibility that some actually do contain other natural languages, but with the used approach it is very improbable. Such other languages would only constitute a tiny part of the whole system (minimum occurrence percentage) or are constrained to a very small number of characters (minimum length). Therefore the analysis of the language of comments and identifiers is especially important for industry projects.

**RQ 3** discusses the differences between the natural language of source-code comments and identifiers. There are two industry projects, which contain only non-English comments and three which contain both non-English comments and identifiers. However, there are no systems, which only contain non-English identifiers. Also, if a project contains both non-English comments and identifiers, there are significantly fewer non-English identifiers, percentage-wise, than non-English comments.

### *E. Threats to Validity*

**Internal Validity:** The chosen parameters and LI tool combinations have a strong influence on the LI results. The LI tool combination delivers the biggest part for the language identification, while the task of the parameters is to filter out a big chunk of false positives without removing too many true positives.

The parameters and the combination of tools were determined by testing on manually created benchmarks that share the same source-code context. This is important, because for example the English used in the documentation of source-code can differ from, say, the English in a novel, and therefore can influence the outcome.

To reduce the bias of the training and the validation of the approach on the same data, we also computed the results, where the data was split into two random chunks with similar size, where one chunk was used for the model training and one for the validation. This splitting was done multiple times. The results of the splitting were more or less similar to the ones depicted in section IV, with Lisanc always yielding better results.

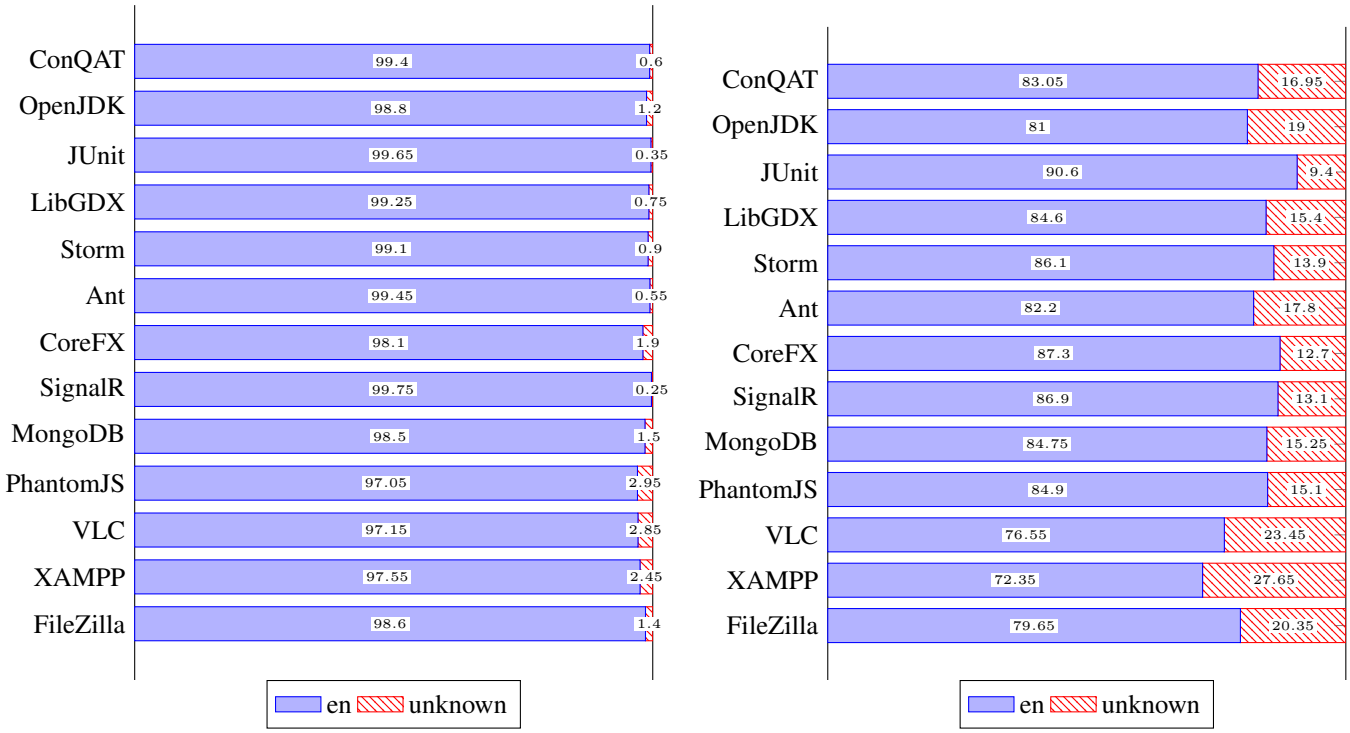
**External Validity:** The choices for the open-source system were made in order to achieve variety for the criteria *size*, *popularity*, *programming-language* and *application*

The selection of the industry projects was more limited. However, they are still from six different companies, written in two programming languages and have considerably different applications. Also, it is important to note that the occurrence of non-English comments and identifiers is not constrained to one company, but is instead distributed over almost all of them.

As both selections can obviously not cover all possible varieties of OS- and industry-systems, all statements in this work about them should not be taken as facts, but rather as indications or trends.

## VI. Conclusion and Future Work

This work presents an approach adapted to the needs of the extraction and language identification of source-code comments and identifiers. An exemplary application is the quantification of non-English comments and identifiers. Multiple conclusion can be drawn on that basis, for example, if it is possible to outsource the development or the maintenance of the system in the current state. Additionally the proposed method can be used for the localization and subsequently




(a) Comments language distribution

(b) Identifier language distribution

Fig. 6. Results of the open-source projects

the fixing (i.e. translating) of non-English comments and identifiers.

The case study demonstrates that the problem of different natural languages for comments and identifiers in a project is not a rare case for industry projects. In fact, half of the examined projects contain non-English comments or identifiers. Therefore we conclude that the natural language of comments and identifiers deserves more attention in research and practice, especially regarding industry code.

## References


[1] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd annual international conference on Design of communication: documenting & designing for pervasive information*. ACM, 2005, pp. 68–75.

[2] S. N. Woodfield, H. E. Dunsmore, and V. Y. Shen, "The effect of modularization and comments on program comprehension," in *Proceedings of the 5th international conference on Software engineering*. IEEE Press, 1981, pp. 215–223.

[3] T. Tenny, "Program readability: Procedures versus comments," *Software Engineering, IEEE Transactions on*, vol. 14, no. 9, pp. 1271–1279, 1988.

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

[5] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," in *Program Comprehension (ICPC), 2013 IEEE 21st International Conference on*. IEEE, 2013, pp. 83–92.

[6] N. Khamis, R. Witte, and J. Rilling, "Automatic quality assessment of source code comments: the javadocminer," in *Natural language processing and information systems*. Springer, 2010, pp. 68–79.

[7] L. Tan, D. Yuan, and Y. Zhou, "Hotcomments: how to make program comments more useful?" in *HotOS*, 2007.

[8] N. Anquetil and T. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 1998, p. 4.

[9] W. B. Cavnar, J. M. Trenkle *et al.*, "N-gram-based text categorization," *Ann Arbor MI*, vol. 48113, no. 2, pp. 161–175, 1994.

[10] T. Dunning, *Statistical identification of language*. Computing Research Laboratory, New Mexico State University, 1994.

[11] T. Vatanen, J. J. Väyrynen, and S. Virpioja, "Language identification of short text segments with n-gram models." in *LREC*. Citeseer, 2010.

[12] T. Baldwin and M. Lui, "Language identification: The long and the short of the matter," in *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2010, pp. 229–237.

[13] E. Tromp and M. Pechenizkiy, "Graph-based n-gram language identification on short texts," in *Proc. 20th Machine Learning conference of Belgium and The Netherlands*, 2011, pp. 27–34.

[14] M. Lui, J. H. Lau, and T. Baldwin, "Automatic detection and language identification of multilingual documents," *Transactions of the Association for Computational Linguistics*, vol. 2, pp. 27–40, 2014.

[15] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE, 2011, pp. 11–20.

[16] R. Řehůřek and M. Kolkus, "Language identification on the web: Extending the dictionary method," in *Computational Linguistics and Intelligent Text Processing*. Springer, 2009, pp. 357–368.

[17] M. Lui and T. Baldwin, "langid. py: An off-the-shelf language identification tool," in *Proceedings of the ACL 2012 system demonstrations*. Association for Computational Linguistics, 2012, pp. 25–30.

[18] D. Okanohara and J. Tsujii, "Text categorization with all substring features." in *SDM*, 2009, pp. 838–846.

[19] A. i. 2008., "Lingpipe 4.1.0." [Online]. Available: http://alias-i.com/lingpipe

[20] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427–437, 2009.