National Boundaries and Semantics of Artefacts in Open Source Development

Andrea Capiluppi
Department of Computer Science
Brunel University London (UK)
andrea.capiluppi@brunel.ac.uk

Nemitari Ajienka
Department of Computer Science
Edge Hill University (UK)
nemitari.ajienka@edgehill.ac.uk

ABSTRACT

Global software development has long being recognised as a paradigm shift in modern software development. As an immediate effect, co-location of workers in the same building or office is not seen as necessary any longer. Coordination in distributed socio-technical systems is mostly achieved by means of the artifacts that are produced by the developers part of a project's team.

Geographic distance profoundly affects the ability to collaborate. With communication becoming less frequent, the challenge is for it to become more effective. This is especially complex when different nationalities, languages and cultures are part of the same development effort. Open source software is an example of a distributed, multi-lingual development effort. As such, the main resulting artefacts are discussions, and source code. Diverse backgrounds can produce a different semantic corpus if the authors come from the same ethnic and language groups or from different ones.

The purpose of this paper is to evaluate the artifacts in the context of their semantics, and how semantic corpora are affected by development and languages. By using a selection of Open Source projects developed within national boundaries, we compare their semantic richness, and how their class content is reflected in their identifiers. We also compare these national projects to a successful, international project. The aim is to discover how national boundaries influence the semantics of the developed code.

KEYWORDS

semantics, open source, word corpus, software health

ACM Reference Format:
Andrea Capiluppi and Nemitari Ajienka. 2018. National Boundaries and Semantics of Artefacts in Open Source Development. In *SoHeal'18: SoHeal'18:IEEE/ACM 1st International Workshop on Software Health , May 27-June 3, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194124.3194131

1 INTRODUCTION

Software teams are becoming remote, jobs are outsourced and as such developed by developers with different native languages [14]. These developers might also write comments in their native languages for personal note taking or technical debt and forget to delete these after fixing issues. Others might simply choose to write comments in their own language in a team composed of non-English speakers [17].

This complex, distributed collaboration presents many challenges: one such challenge is related to the understandability of the artifacts. For international projects, collaboration is achieved via semantically rich documents. The terminology used in the documentation should be precise and unambiguous [18], in order for non-native speakers to understand the concepts of the underlying artifacts, and to improve collaboration [22]. On the other hand, national boundaries could reinforce local terminology, hence increasing the 'base' vocabulary needed by contributors to be part of a project, and decreasing its understandability [15].

When dealing with international teams, and multi-language development processes, it is possible to strip all the *comments* contained within code and documentation. These snippets can be parsed by readily available tools[1] to automatically detect the language that the documentation and comments are written in. Using such tools, it could be possible to detect the size of the vocabulary used in the artifacts, and detect if national projects have the tendency to increase the number of keywords and concepts. A larger number of such concepts could be related to issues of understandability for other developers, and a hindrance for multi-national, open source projects.

This paper has the objective of studying those open source projects that are clearly delimited by national boundaries, and where their authors are unambiguously linked to one national language. The effect of these national boundaries on the artifacts should be visible in the resulting artifacts: larger dictionary of terms, in both source code and documentation, as compared to international projects, where non-native speakers could find some of the terminology ambiguous or difficult to understand.

We selected 5 open source projects from the GitHub repository, with national teams as the sole contributors, all implemented in the Java language: we evaluated the underlying source code, and isolated the concepts and keywords associated with each class. We considered each class *corpus* throughout a project, and between different projects, for comparison. Furthermore, we considered in a similar way a successful multi-developer, international open source project, CloudStack. We also extracted the keywords and concepts from its classes, with the objective to compare these results with those extracted from the national projects. Shedding light on how

[1] An OSS project to provide a Python port to the Google language detection engine is available at https://pypi.python.org/pypi/langdetect?. A from-scratch implementation in PERL is available at http://search.cpan.org/~ambs/Lingua-Identify-0.56/

semantics play a role in the collaboration between developers could provide additional insights to characterise the health of a long-lived software project.

This paper is articulated as follows: Section 2 provides the related work, Section 3 introduces the methodology of this study, while Section 4 shows the results of the analysis. Section 5 discusses the threats to validity that we could identify in our experimental setup. Section 6 concludes the study with directions for further work.

2 RELATED WORK

Global, international software development shows many promises and benefits, but "anyone engaging in GSD should be aware of the many risks associated with these 'benefits"'[6]. One of such challenges is therelated to culture and language: since English has been traditionally considered the 'lingua franca' of software development, misinterpretations and understandability issues have been flagged [17], and practical solutions proposed. The open source development paradigm has not formally acknowledged the issue in handling international developers, who could be far from native speakers. The mechanisms of such collaboration and what role language plays in facilitating the coordination between developers, have been vastly ignored.

On the contrary, the literature on semantics and software development has attracted substantial attention: some studies have used the term *"semantic"* [2–5, 9, 12, 20, 21], while others have used the term *"conceptual"* [9] to describe the same concept. Poshyvanyk *et al.* [20] state that *conceptual coupling* captures the degree to which the identifiers and comments from different classes relate to each other [2–5, 12, 21]. Gethers *et al.* [9] add a twist to the definition and state that conceptual coupling captures the extent to which domain concepts/features and software artefacts are related to each other. However, both definitions have things in common. They are limited to the underlying meanings of unstructured text in the source code of software entities (e.g., classes) and how these underlying meanings relate to each other. Furthermore, this relationship is derived in the form of semantic similarity metrics (-1 to 1, where 1 = high semantic coupling [19]).

Identifiers used by developers for names of classes, methods, or attributes in source code or other artifacts contain important information and account for approximately half of the source code in software [11]. These names often serve as a starting point during program comprehension. Hence, it is essential that these names clearly reflect the concepts that they are supposed to represent, as self-documenting identifiers decrease the time and effort needed to acquire a basic comprehension level for a programming task [11].

2.1 Background and previous work

In a previous study [1], we compared two sentence similarity techniques (based on N-Gram [2] and Disco Word synonym[3] categories methods) against a corpus or text document cosine similarity based technique (VSM)[4] [5] for computing semantic similarity between Java classes.

The study was conducted using two software projects: by means of Chi-squared independence tests, we identified that measuring the semantic similarity between classes using (only) their identifiers is similar to using the class corpora. This is because using identifiers was more efficient in terms of recall, and computation time [1]. The study also identified that English based word similarity techniques such as WordNet do not perform well on software terminologies (e.g., export ↔ dump). The steps taken to compute the semantic similarity between Java classes using the three techniques is explained in detail in [1].

In addition, the N-Gram technique is based on the edit distance and shared sub-strings of length n between sentences [13]. An example is the semantic similarity between two class identifiers *'Ur S Q L Controller'* and *'Ur S Q L Entry'* which returns a value of 0.6 for shared sub-strings between 0 and 4. Prior information retrieval research [13, 16] that shows that n=4 increases the precision when analyzing words or terms in various languages. However, the n-gram technique returns a number of false positives. For example, when analysing the class identifiers *Abstract Basic Id Bean* and *Lock Cell Renderer* it returns a semantic similarity of 0.9.

The Disco technique although with low precision on non-English words has been compared to other text similarity techniques and proven to perform well when its outputs were manually checked. According to Despotakis *et al.* [7] "although the precision for Disco was low, it did provide additional valuable concepts, which were approved by both experts. We also manually checked the outputs of the semantic similarity measurements to minimize the effects of false positives. Using the Disco tool to compute the similarity between the two identifiers *Buddy Service Test* and *Buddy Servlet* returns a similarity metric of 0.6.

3 EMPIRICAL APPROACH

The work we present here is based on an overarching *goal*: to understand the role of semantics in facilitating distributed development. The *context* we conjugate this aim is within open source projects. The *approach* to measure semantics of a software project is by examining both the corpora of software classes, and their identifiers (e.g., class names).

From this general goal, we derived several research questions (RQs), and testable hypotheses formulated for each question, as summarised below. A rationale is also provided for each RQ.

RQ1 How is the semantics of a project reflected in its artifacts? Is the *overall* size of the corpora dependent on the size of the system, as measured in software classes?
<u>*Rationale*</u>: as long as the number of classes increase, also the number of concepts contained in a system should increase. If the size of a system does increase, but the corpora does not (or *viceversa*), *ad-hoc* explanations should be evaluated.

[2]A Java implementation of the N-Gram distance algorithm is available at https://github.com/tdebatty/java-string-similarity#n-gram

[3]The Disco sentence similarity measures the semantic similarity between sentences according to the synonyms of their words. A Java implementation of the tool is publicly available at https://sourceforge.net/projects/semantics/?source=directory

[4]We have developed a tool that uses the VSM method to automate the corpus based technique. It can be downloaded at: https://github.com/najienka/SemanticSimilarityJava

[5]Two out of the studied projects have also been added to the online repository for replication.

Hypothesis (H_0): the corpora of identifiers increases with the size of a system. National and international projects show a similar correlation between size and corpora.

RQ2 Considering individual classes, what is the size of the corpora contained? Can that be considered quasi-constant for any given class, or across projects?
Rationale: for the principle of class cohesion, classes should contain related content. Although larger classes might exist, their corpora (e.g., concepts and keywords) should be comparable with smaller classes.
Hypothesis (H_0): the size of the corpora per class is similar for different classes in the same project. National projects tend to have large corpora per class, due to language richness.

RQ3 Considering the classes in a software project, what is the ratio of classes that share a similar corpus?
Rationale: classes should be independent in their content and semantics. We posit that their semantics should be independent.
Hypothesis (H_0): only a small ratio of classes share the same bag of words, per project.

RQ4 Is there a correlation between the identifiers of a project (e.g., the class names) and the semantics of the classes?
Rationale: a pilot study demonstrates that computing the semantic similarity between classes based on their identifiers is more efficient in terms of computation time and identifier-based similarity metrics reflect corpora-based metrics. National projects should show less of such behaviour: the semantic richness of class corpora is not fully represented by their identifiers.
Hypothesis (H_0): the identifiers of classes are representative of the terms present in their corpora. National projects drift off this behaviour, due to their superior semantic richness.

3.1 Case studies

For this study, we examined 5 GitHub repositories, all implemented in Java. We selected the projects based on a nationality search of the GitHub respositories. For the analyses presented in this paper, we only focused on those projects where developers were clearly identifiable within one national boundary. We manually checked that the names associated to the contributors of each project were consistent with the nationality of the project considered.

The projects selected are summarised in Table 1, with the nationality of their core contributors, and the number of classes in the snapshot evaluated at the time of writing (February 2018).

Project	Nationality	Contributors	Classes
multiple-dimension-spread	Japanese	4	464
javaee-nosql	Brazilian	2	19
clients	Nigerian	8	631
LaunchEnr	Italian	3	312
workshop-hystrix	French	2	31
CloudStack	various	264	1,075

Table 1: Systems analysed and their characteristics

In addition to these *national* projects, we selected the CloudStack system (last row of Table 1), as analysed in [10]. The paper reports a study of the CloudStack community, and it evaluates the contributions based on the different times of the day when code and discussions were contributed. The paper reports that the CloudStack system has evolved from a small community to a large endeavour, spanning various *timezones* and nationalities. The two parts of Figure 1 are taken from the paper and they show how the CloudStack project has gradually moved from a relatively closed community, to an international effort made of many different nationalities (here depicted by different timezones).

For all the systems considered, we isolated the Java classes and analysed their semantic content in two ways: (i) by considering their identifiers (i.e., their class names); and (ii) parsing their code and considering the variable names, comments and keywords. Section 2.1 describes these two steps with more details.

4 RESULTS

In this section we report the results of the analysis, based on the research questions presented above.

4.1 RQ1: Size of corpora

In order to define what is the size of the corpus per class in each project, the VSM tool was used to reduce the source content of each class to its basic keywords and identifiers in the form of a text (`.txt`) file. Stopwords are eliminated, and key terms saved into a vector: repetitions can be present, if the same keyword appears more than once. We filtered out repetitions for the analysis of this first research question.

Table 2 shows the results of this analysis: the third column shows the median value of the size of the class corpora, per project; whereas the fourth column shows the size of the overall keyword corpus, per project.

Project	Classes	Keywords per class (median)	Keywords (overall)
multiple-dimension-spread	464	23	433
javaee-nosq	19	22	183
clients	631	40	2,464
LaunchEnr	312	57	2,286
workshop-hystrix	31	24	232
CloudStack	1,075	44	3,116

Table 2: Size of the corpora for the analysed projects

We notice that the overall number of classes is generally a good proxy for the size of the corpora in the selected systems. The only exception is the *multiple-dimension-spread* project that has the third largest number of classes in the sample, but the size of its corpus is much lower than expected. Figure 2 shows visually the trend between the number of classes, and the overall keyword corpus, per project.

Figure 2 serves as a means to formulate a tentative assessment of the semantic status of the projects: a growth in number of classes is reflected in a growth in number of keywords and concepts contained

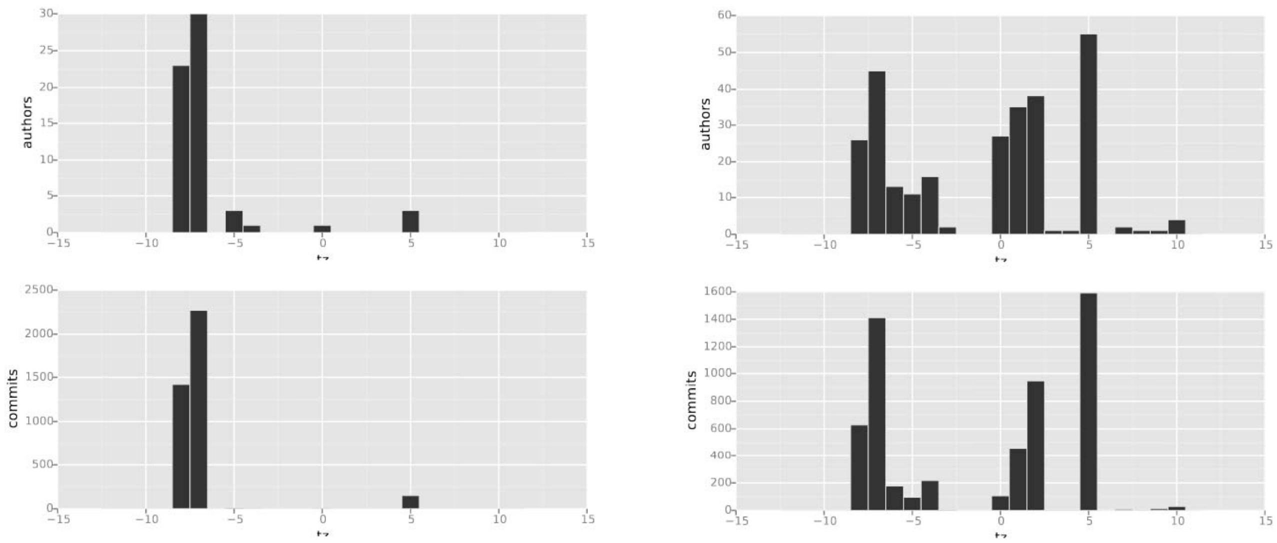

Figure 1: CloudStack in 2010 and 2014 (taken from [10])

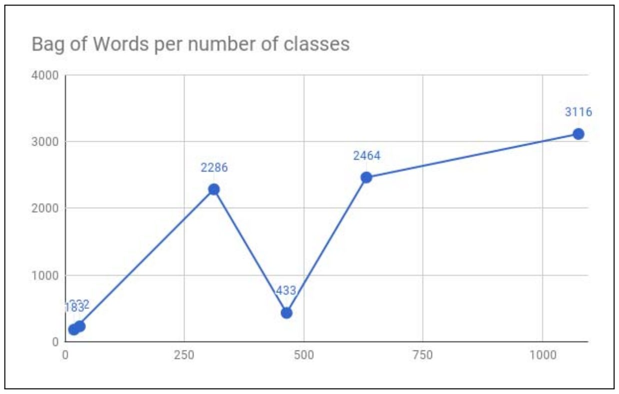

Figure 2: Relationship between number of classes and overall corpus for the analysed systems

in the source code. When the keyword corpus of a system does not evolve with its class number (as in the *multiple-dimension-spread* project), the code does not generate enough new concepts. This could be considered as a powerful metrics of the status of a software project. We will evaluate the status of this project in more details in Section 4.2.

4.2 RQ2: size of class corpus

Similarly to the analysis of the overall keyword corpus, we analysed, per class, the extent of its corpus. We produced a distribution of corpus sizes, per project: the third row of Table 2 summarises the median of these distributions, per project. Larger projects tend to have classes with larger corpora: the *multiple-dimension-spread* project is again anomalous, since it contains many more classes than the *workshop-hystrix* or *javaee-nosql* projects, but it has a similar median.

What is relevant to notice is that the classes in the selected systems converge to a 20-50 interval of bag of words, as a median. Further replication is needed to validate this finding for other systems.

Figure 3 reports the distributions of the corpora per project using boxplots: as labels, we use the nationality of the analysed projects. The French and Brazilian projects (*workshop-hystrix* and *javaee-nosql*, respectively) have less classes, and their class corpora have a relatively smaller distribution. On the other hand, the *Japanese* project has a large number of classes, but a small variability in the class corpora: its boxplot is quite compressed (i.e., both its quartiles coincide with the median).

The role of semantics in the health of a software project becomes clear if we consider the corpora in Figure 2 and the boxplot in Figure 3. The limited corpus of the *Japanese* project signals a stagnating project (also visible in its dip in Figure 2). This is confirmed by its restricted scope: as a further evidence, we observed that more than a third of its classes are indeed test classes.

4.3 RQ3: similarity between class corpora

The third research question focuses on the *class corpora* of a same project, and how similar they are between different classes. This is an important aspect in the development of a software system: if two classes share an excessive amount of keywords and concepts, they should be merged. We posit that smaller projects present a larger amount of similar classes: a reduced set of features get spread in various classes. We also posit that National projects present similar similarity between classes to international projects.

The Vector Space Model (VSM) information retrieval technique was used to retrieve information for class-to-class similarity in semantic space, based on the corpora extracted from the classes. If

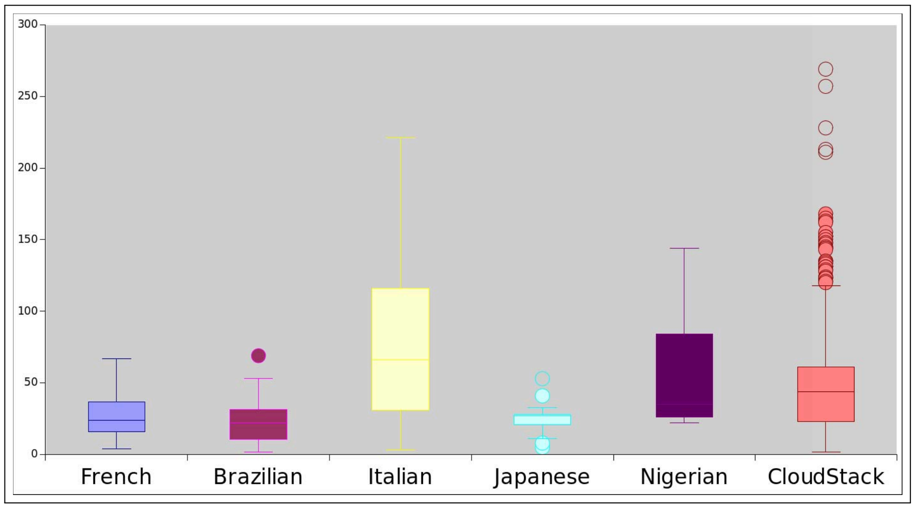

Figure 3: Distributions of class corpora, per project

a class had the same keyword more than once, its cardinality was preserved for comparing similarity with another class.

In most of the semantic coupling studies, the computation of the similarities have been implemented using MATLAB [6]. The tool computes the semantic similarity between two documents (classes) using the *cosine similarity*, as shown in the formula below:

$$CosineSimilarity(d1, d2) = Dotproduct(d1, d2)/||d1|| * ||d2|| \quad (1)$$

The semantic similarity of the classes based on their corpora is computed using our corpora-based semantic similarity measurement tool[7]. The class-to-class similarity metric is symmetric, i.e., similarity metric for the pairs *Filio* ↔ *UrSQLEntity* and *UrSQLEntity* ↔ *Filio* is the same (0.006). We report the results of the semantic similarity between classes, and at different levels, in Table 3.

We notice that, according to the posited hypothesis, the effects of language are more visible when considering the project's size: smaller projects, implementing less features, tend to show more similarities between the lesser amount of classes.

At the same time, and contrary to the posited hypothesis, the cosine similarity detects a relevant percentage of similar classes in all the national-specific projects. On the other hand, the classes of CloudStack are less affected by mutual similarity of their corpora.

4.4 RQ4: correlation between identifiers and corpora

Finally, we investigated whether the semantic similarities of class identifiers in a project are correlated with the semantic similarity of their content: the corpus of a class was considered, together with the class name. We posit that National projects are more effective at linking the class identifiers with their corpora; we also posit that

Similarity (cosine)	0.3	0.4	0.5	0.6
multiple-dimension-spread	7.16%	4.61%	3.11%	2.00%
javaee-nosql	11.11%	7.60%	3.51%	1.75%
clients	3.93%	2.38%	1.34%	0.67%
LaunchEnr	2.74%	1.33%	0.67%	0.32%
workshop-hystrix	4.30%	3.87%	2.37%	1.29%
CloudStack	2.22%	1.22%	0.71%	0.40%

Table 3: Similarity levels - class corpora

the size of a project plays an opposite role: the smaller the project, the more likely for it to show alignment between class identifiers and their corpus.

For each class pair, the analysis around RQ4 was based on three steps: (i) computing the semantic similarity between *class corpora*; (ii) computing the semantic similarity between *class identifiers*; and (iii) establishing a *correlation* between these two measurements.

As shown in Section 4.3, we developed our own tool to compute the semantic similarity between the corpora of two classes.

On the other hand, to compute the semantic similarity between class identifiers or names, we used the DISCO publicly available sentence similarity Java tool,[8] that measures the semantic similarity between sentences according to the synonyms of their words. DISCO is an enhancement of the Vector-Space analysis found within the Classifier4j[9] which has been used in IR research.

The tool is based on a dictionary of the EOWL list of words[10], while the synonyms for each word are calculated using the DISCO's semantics[11] to get groups of the most similar words in the dictionary. For example, **shy** ↔ *timid, quiet, introverted, lonely, cautious,*

[6]MATLAB is a high-level technical computing language and interactive environment for algorithm development, data analysis and visualization http://uk.mathworks.com/products/matlab/

[7]https://github.com/najienka/SemanticSimilarityJava

[8]Available at https://sourceforge.net/projects/semantics/?source=directory

[9]http://classifier4j.sourceforge.net/

[10]http://dreamsteep.com/projects/the-english-open-word-list.html

[11]http://www.linguatools.de/disco/disco_en.html

awkward, clumsy, soft-spoken and *gentle*. An example of using the tool to compute the semantic similarity between two class identifiers *'Ur S Q L Controller'* and *'Ur S Q L Entry'* returns a value of 0.89 with a computation time of 0.003 minutes.

After computing identifier and corpora based similarities between class pairs, the similarity results of each of the analysed projects had three items: (i) the class pair, (ii) their identifier based similarity metric and (iii) their corpora based similarity metric. All this data was clustered per project, and correlation sought: the Spearman coefficient was evaluated, and the p-value, for each project. We report this analysis in Table 4.

Using the `spearman` correlation method in the `R` statistical environment, we could then parse the files per project and compute the correlation between identifier and corpora based semantic similarity metrics. The results are presented in Table 4. The first column shows the project names, the second column shows the correlation value (ranges between -1and 1) and the last column shows the derived *p-value*.

Project	Correlation	p-value
multiple-dimension-spread	0.41	0
javaee-nosql	0.465	9.153e-20
clients	0.331	0
LaunchEnr	0.248	0
workshop-hystrix	0.4642	6.72e-51
CloudStack	0.469	0

Table 4: Spearman's rank correlation metrics and p-values

Considering an $\alpha \leqslant 0.05$, the results show a significant correlation between corpora and identifier of classes for the smaller, national projects. When the number of classes is smaller (e.g., in the *javaee-nosql* and *workshop-hystrix* projects), the Spearman correlation is higher. This is in line with the hypothesis posited. This is a very important result: the semantic information provided by the class names is similar to the corpora of the class itself.

On the other hand, with the increase in size of the national projects selected, this correlation gets lower: contrary to the hypothesis posted, corpora and identifiers become less correlated, which could create more understanding work for future development and maintenance.

The international project selected maintains instead a high level of similarity between identifiers and corpora: if replicated and confirmed, this could represent an important aspect in the evaluation of health of a software system.

5 THREATS TO VALIDITY

Her we present the threats to validity that we found for this study: we discuss them below as *external*, *internal* and *construct* threats.

External validity. This paper presents the results of an empirical analysis that should be applicable to all OSS projects [8]. We cannot generalise our findings on any other sample of OSS projects, or from any other repository. This threat is particular acute in our case study, since the studied projects are few and selected based on convenient sampling. A different, replicable methodology is needed to select projects that are limited to national boundaries. Once that methodology is available, a more random approach to sampling would be possible.

Internal validity. The relationship between successful open source collaboration and understandability of artifacts is undoubtedly much more complex. For instance, the understandability of the source code could be related to various other aspects that we have not considered here: the presence of inheritance, polymorphism, and late binding, as well as the static measures of complexity could provide even more challenges to the understandability, than the number of keywords contained in the artifacts. Nonetheless, the extent of the *dictionary* of an artifact, let alone its structure, is the absolute basis of any structural or dynamic complexity.

Construct validity. Although we analysed national projects, as compared to a large international open source project, we did not look through their code to detect the presence (or absence) of national slang or non-English terminology. We just evaluated the code in those national projects for the purpose of future collaboration by other (non-nation specific) developers. This threat is anyway limited: the corpora contains distinct terms, including slang or national words, that would increase the size of the corpora per class.

6 CONCLUSION

This paper considered the semantics of Java identifiers, and the corpus of keywords contained in Java code, in a sample of Java projects. We selected 5 projects that are currently developed by *national* teams (e.g., all contributors are within the same national boundaries), and we also considered CloudStack, the Apache project for creating and deploying cloud networking services.

We posited 4 main research questions, along the relevance of national boundaries in the semantics of the systems. Below we summarise the main findings:

- depending on the size and the scope of the system, the overall number of keywords in a class increases. A smaller ratio between number of classes and overall corpora signals a limited scope application.
- We found a bracket of [22 – 57] keywords per class (as a median), in the analysed systems. Smaller systems have less keywords in their classes. National projects have more keywords per class, as compared to the international project used in our analysis (i.e., CloudStack).
- The smaller projects in our analysis tend to have a larger similarity between the corpora of different classes. The national projects show more similarity between classes than the international project analysed.
- The smaller systems show a larger correlation between class identifiers and their corpora. The national systems show a lower correlation, whilst the international system analysed has a comparatively higher correlation.

These results are encouraging: they point to the importance of semantics for national and international projects. Also, these results provide a new aspect to consider for software health: international projects have to align the corpora and identifiers of their classes as a means to collaboration between international developers.

REFERENCES

[1] Nemitari Ajienka and Andrea Capiluppi. 2016. Semantic Coupling Between Classes: Corpora or Identifiers?. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 40.

[2] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. 2010. A two-step technique for extract class refactoring. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 151–154.

[3] Gabriele Bavota, Bogdan Dit, Rocco Oliveto, Massimiliano Di Penta, Denys Poshyvanyk, and Andrea De Lucia. 2013. An empirical study on the developers' perception of software coupling. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 692–701.

[4] Gabriele Bavota, Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea de Lucia. 2014. Improving software modularization via automated analysis of latent topics and dependencies. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 1 (2014), 4.

[5] Gabriele Bavota, Rocco Oliveto, Malcom Gethers, Denys Poshyvanyk, and Andrea De Lucia. 2014. Methodbook: Recommending move method refactorings via relational topic models. *Software Engineering, IEEE Transactions on* 40, 7 (2014), 671–694.

[6] Eoin Ó Conchúir, Pär J Ågerfalk, Helena H Olsson, and Brian Fitzgerald. 2009. Global software development: where are the benefits? *Commun. ACM* 52, 8 (2009), 127–131.

[7] Dimoklis Despotakis, Dhavalkumar Thakker, Lydia Lau, and Vania Dimitrova. 2011. Capturing the semantics of individual viewpoints on social signals in interpersonal communication. *Semantic Web Journal, Special Issue on Personal and Social Semantic Web ((Under review))* (2011).

[8] Robert Feldt and Ana Magazinius. 2010. Validity Threats in Empirical Software Engineering Research-An Initial Survey.. In *SEKE*. 374–379.

[9] Malcom Gethers, Amir Aryani, and Denys Poshyvanyk. 2012. Combining conceptual and domain-based couplings to detect database and code dependencies. In *Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on*. IEEE, 144–153.

[10] Jesus M Gonzalez-Barahona, Gregorio Robles, and Daniel Izquierdo-Cortazar. 2016. Determining the Geographical distribution of a Community by means of a Time-zone Analysis. In *Proceedings of the 12th International Symposium on Open Collaboration*. ACM, 3.

[11] Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. 2013. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering* 18, 5 (2013), 933–969.

[12] Huzefa Kagdi, Malcom Gethers, Denys Poshyvanyk, and Michael L Collard. 2010. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*. IEEE, 119–128.

[13] Vlado Kešelj, Fuchun Peng, Nick Cercone, and Calvin Thomas. 2003. N-gram-based author profiles for authorship attribution. In *Proceedings of the conference pacific association for computational linguistics, PACLING*, Vol. 3. 255–264.

[14] Do Ba Khang and Tun Lin Moe. 2008. Success criteria and factors for international development projects: A life-cycle-based framework. *Project Management Journal* 39, 1 (2008), 72–84.

[15] Kari Laitinen. 1996. Estimating understandability of software documents. *ACM SIGSOFT Software Engineering Notes* 21, 4 (1996), 81–92.

[16] Paul Mcnamee and James Mayfield. 2004. Character n-gram tokenization for European language text retrieval. *Information retrieval* 7, 1-2 (2004), 73–97.

[17] John Noll, Sarah Beecham, and Ita Richardson. 2010. Global software development and collaboration: barriers and solutions. *ACM inroads* 1, 3 (2010), 66–78.

[18] David Lorge Parnas. 2011. Precise documentation: The key to better software. In *The Future of Software Engineering*. Springer, 125–148.

[19] Denys Poshyvanyk and Andrian Marcus. 2006. The conceptual coupling metrics for object-oriented systems. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*. IEEE, 469–478.

[20] Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy. 2009. Using information retrieval based coupling measures for impact analysis. *Empirical software engineering* 14, 1 (2009), 5–32.

[21] Abdallah Qusef, Gabriele Bavota, Rocco Oliveto, Andrea De Lucia, and David Binkley. 2011. Scotch: Test-to-code traceability using slicing and conceptual coupling. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*. IEEE, 63–72.

[22] Jim Whitehead. 2007. Collaboration in software engineering: A roadmap. In *2007 Future of Software Engineering*. IEEE Computer Society, 214–225.