Using the Lexicon from Source Code to Determine Application Domains

Andrea Capiluppi*, Nemitari Ajienka†, Nour Ali*, Mahir Arzoky*, Steve Counsell*, Giuseppe Destefanis*
Alina Miron*, Bhaveet Nagaria*, Rumyana Neykova*, Martin Shepperd*, Stephen Swift*, Allan Tucker*
*Department of Computer Science, Brunel University London, UK
†Department of Computer Science, Edge Hill University, UK

***Abstract*—*Context:* The vast majority of software engineering research is independent of the application domain: techniques and tools usage is reported without any domain context. This has not always been the case - early in the computing era, there were totally separate application domains (for example, scientific and data processing) and the research focus was often application-specific.**

***Objective:* This paper claims that software systems should be separated and analysed into domain clusters. We propose a code-based approach to identify the application domain of a software system, *via* its lexicon. We compare its precision with the plain textual description attached to the same system.**

***Method:* Using a sample of 50 Java projects, we obtained i) the description of each project (e.g., its ReadMe file), ii) the lexicon extracted from its source code, and iii) a list of its main topics extracted with the LDA information retrieval technique. We assigned a random subset of these data items to different researchers (i.e., 'experts'), and asked them to assign each item to one (or more) application domain. We then evaluated the precision and accuracy of the three techniques.**

***Results:* Using the agreement levels between experts, We observed that the 'baseline' dataset (i.e., the ReadMe files) obtained the highest average in terms of agreement between experts, but we also observed that the three techniques had the same mode and median agreement levels. Additionally, in the cases where no agreement was reached for the baseline dataset, the two other techniques provided sufficient support.**

***Conclusions:* We conclude that using the corpora or the topics from source code can be an adequate substitution to plain description when assigning a software system to an application domain.**

***Index Terms*—Application Domains, Source Code, Java, Latent Dirichlet Allocation, Expert Judgement, Machine Learning**

I. Introduction

Albeit the diversity and context of software systems have received some attention in the past [26], [10], contemporary research in the computing field is almost entirely application-independent. This has not always been the case - early in the computing era, there were totally separate application domains (for example, scientific and data processing) and the research focus was often application-specific [15].

In the context of empirical software engineering research, while the main goal of empirical papers is to achieve the generality of the results, the domain, context and uniqueness of a software system have not been considered very often by researchers. The most common rationale for doing so is to analyze projects having different application domains to decrease threats due to the generalizability of the results.

As in the example reported in [21], the extensive study of all JSON parsers available would find similarities between them or common patterns. That type of study would focus on one particular language (JSON), one specific domain (parsers) and inevitably draw limited conclusions. On the other hand, considering the "parsers" domain (but without focusing on one single language) would show the common characteristics of developing that type of systems, irrespective of their language. The thrust of this paper stems from the work of several prominent researchers who called the empirical software engineering community to 'go deeper, not wider' [24] and 'minding the mine, mining the mind' [18]. As a matter of fact, there is increasing evidence that empirical research on software systems might yield domain-specific results, for instance when clustering the studied systems by the domains that they implement ([11], [27], [23]).

There are two main challenges to domain-based empirical software engineering: first, there is currently no commonly agreed taxonomy of application domains [15]. Several attempts have been performed with varying success: past research on domains has focused on creating a domain taxonomy in a top-down fashion [1], i.e., starting with a seed taxonomy and refining it with various techniques (e.g., *via* expert judgement) [12]. The second challenge is assigning a software system to a domain: again, expert judgements have been applied more often than automatic assignments in past literature [4], but it has become clear that the chosen domains were selected as either too fine-grained or too large, thus defying the purpose of the categorisation.

In this paper, we argue that the extraction of *topics* that emerge from the source code can help the assignment to domains by experts, and instead of (or aside of) reading the documentation that accompanies a software system [25]. Researchers and practitioners can assign a system to a domain by only focusing on a limited number of core topics that emerge more strongly (e.g., with larger weight) from the lexicon of source code.

For this purpose we collected the description of 50 Java software systems, alongside the lexicon of their source code and a list of their most relevant topics. We asked 10 researchers to assign a unique subset of these data files to the most appropriate domain, and triangulated their expert opinions to discuss the following research questions:

1) RQ1: is the lexicon an acceptable substitute for the

description of a software system, for the purpose of assigning it to a domain?
2) RQ2: similarly, are the topics (as extracted by LDA) acceptable substitutes?

This paper is organised as follows: section II deals with related work and illustrates how our paper advances the state of the art. Section III describes the empirical approach that we adopted to extract the data sources and to assign them in subsets to researchers for expert judgement. Section IV summarises the results of the expert judgement task, while section V discusses the findings and their relevance for practitioners. Section VI details the threats to valisity and section VII finally concludes.

II. Related Work

This work is an extension of a previously published paper [6] where we posited that keywords, corpora and topics could significantly help in establishing the provenance of a software system, given a list of pre-defined application domains. Now we plan to evaluate the plain description of a system (e.g., in the form of a ReadMe file) against the corpora and the topics extracted from the source code. The current paper should therefore be considered as the enactment of the future work proposed in [6]. Prior research has shown that the number and size of open-source projects are growing exponentially and open-source projects are becoming more diverse by expanding into different domains [9], [17]. In view of this and to reduce the effort required in manual categorisation of software projects, Tian *et al.* [25] proposed a new technique based on text mining to categorise software projects irrespective of the programming language used in their development. Their contribution is based on the analysis of the documentation that accompanies the software project, rather than the source code, as we propose in our paper.

In general, distinct results have been observed when more attention is paid to the categorisation of analysed software projects. For example, Wermelinger and Yu [27] suggest that presenting two datasets from the same software domain (e.g., Eclipse and NetBeans) allows for future comparative studies and facilitates the reuse of data extraction and processing scripts. On increasing the external validity of empirical result findings, German *et al.* [13] have also highlighted the need to investigate particular systems belonging to different domains. Previous studies [22], [2], [5] revealed that projects from different domains use exception handling differently and that poor practice in writing exception handling code is widespread. In a study on Java projects by Osman *et al.* [23] the authors aimed to answer the following research question: "Is there any difference in the evolution of exception handling between projects belonging to different domains?". The researchers manually categorized 30 projects into 6 domains, namely compilers, content management systems, editors/viewers, web frameworks, testing frameworks, and parser libraries. Their observations showed significant distinctions in the evolution of exception handling between these domains, like the usage of `java.lang.Exception` and custom exceptions in catch blocks. Concretely, content management systems consistently have more exception handling code and throw more custom exceptions, as opposed to editors/viewers, which have less error handling code and mainly use standard exceptions instead.

Fayad and Smidt [11] explored software frameworks and classified frameworks based on related application domains, e.g., operating system and communication frameworks and user interface frameworks. The authors emphasised that in contrast to earlier OO reuse techniques based on class libraries, frameworks are targeted for particular business units (such as data processing or cellular communications) and application domains (such as user interfaces or real-time avionics). They also highlight the fact that the next generation of OO application frameworks target application domains but on the other hand, application developers in more complex domains such as telecommunications and distributed medical imaging have traditionally lacked standard "off-the-shelf" frameworks; as a result these developers in such domains largely implement, test and maintain software systems from scratch. These findings further show the need to treat project by domains for more distinct empirical results or observations. The need for a means of identifying which domain a project belongs to is also highlighted as OSS developers for example can contribute frameworks for projects in the telecommunications domain upon identifying such projects and their required functionality.

In past research, software projects have been assigned to application domains by glancing at the source code, or its general description (e.g., the ReadMe file, or the project documentation); creating categories and finally assigning a project to a category. The research by Borges *et al.,* [4] follows that approach: the dataset contains 5,000 GitHub project (including 520 Java projects). There are two main issues with this approach: the first is that there is hardly any consistency in how a project might get documented by its developers, meaning that the approach in [4] becomes non-reproducible. The second is that the categories are arbitrarily decided by the authors, and become overpopulated with one type of projects. As an example, the following break-down shows the skewness of the dataset in the reported study:

- Application Software (30 projects in the sample)
- Documentation (48)
- Non Web Libraries And Frameworks (342)
- Software Tools (49)
- System Software (26)
- Web Libraries And Frameworks (25)

III. Empirical Approach

In this section, we discuss how we sampled the systems to study and how the three data sources were extracted from the software projects. Section III-A described how the systems were sampled from the GitHub repository and how the ReadMe was extracted from each system. Section III-B shows how the lexicon was extracted from the Java classes in the sample; section III-C shows how the LDA technique was instrumented to extract weighted topics; while section III-E

illustrates how the expert judgement was gathered and triangulated. The data and the scripts used in the analysis are made available online at *https://XXX.xxx* (link removed for double blind review), for inspection and potential further replication.

A. Sampling Software Systems and ReadMe Files

Leveraging the GitHub repository, we collected the project IDs of the 50 most successful[1] Java projects hosted on GitHub as case studies. As such, our data set does not represent a random sample, but a stratified sample based on one attribute (i.e., success) that is related to usage by end users. As a result of the sampling, our selection contains projects that are larger in size than average: we provide the list of the analysed projects in Appendix A.

The repository of each project was downloaded and stored, and all the Java files identified for further parsing. From each project's folder we extracted the main ReadMe file, that is typically assumed to be the first port of information for new users (or developers) of a project.

B. Extracting the Lexical Content from Java classes

We extract the lexical content of a Java class in two ways:

1) by considering their class names; and
2) by parsing their code and considering all identifiers including method and variable names, comments and keywords.

The code of a Java class is converted into a *text corpus* where each line contains elements of the implementation of a class. The corpus in this case ("dictionary" of terms derived from comments, keywords in source code) is built at the *class* level of granularity [16]. The corpus includes the class name, variable and method names and comments for each class. Preprocessing of the system corpus is performed to eliminate Java keywords[2], stop words, split and to stem class names [20]. The list of such terms is available in the replication package for inspection.

The tool can be downloaded from Figshare[3], and it uses the *ninka*[4] library to detect a source file's license, that is not considered relevant for a source file's lexicon.

For the analyses performed in this paper, we extracted both the *complete* and the *unique* corpus of each class. As an example, Figure 1 shows a snippet of Java code, as extracted from the *UrSQL* project[5].

Parsing the lines of code shown in Figure 1 (the UrSQLEntry.java class from the UrSQL project), we derive the following *complete* corpus using an information retrieval tool developed in Perl (also available for inspection):

[1]As a measure of success, we used the number of *stars* that a project received from other users: that implies appreciation for the quality of the project itself.

[2]As shared on https://en.wikipedia.org/wiki/List_of_Java_keywords

[3]https://XXX.xxx, link removed for double blind review

[4]Available at https://github.com/dmgerman/ninka, as presented in [14].

[5]https://github.com/duncangrubbs/urSQL

```java
package tmacsoftware.ursql;

public class UrSQLEntry
{

    private String key;
    private String value;
    private String firstKey;
    private String firstValue;
    private UrSQLEntity entity;

    public UrSQLEntry()
    {
    }

    public UrSQLEntry(String query)
    {
        String[] split = query.split
(UrSQLController.KEY_VALUE_SEPARATOR);
        this.key = split[0];
        this.value = split[1];
        this.firstKey = this.key;
        this.firstValue = this.value;
    }
```

Fig. 1. UrSQLEntry.java source code snippet

Complete corpus (as extracted from Figure 1)

ur sql entri kei valu kei valu ur sql entiti entiti ur sql entri ur sql entri queri split queri split ur sql control kei valu separ kei split valu split kei kei valu valu.

To obtain the unique corpus, the list of keywords is later pruned of duplicated terms, *per* class. Parsing the source code from Figure 1, we derive the *unique* corpus as follows:

Unique corpus (as extracted from Figure 1)

control entiti entri kei queri separ split sql ur valu.

The *complete* and the *unique* corpora are obviously different, the former being of size 35 and the latter of size 10 (in the example above). As a summary, this data extraction produces, for each analysed system, (i) the *complete* list of terms, and (ii) the list of *unique* terms contained in its source code. These terms form the complete and the unique corpus data: the latter was distributed to the experts *as-is*; while the former was used for the extraction of topics through the LDA technique (see section III-C).

The size of the complete (ALL) and unique (UNIQ) corpora of the sample of projects is displayed as two boxplots in Figure 2. The decision of using the unique corpora as unit of category assignment is due to readability and ease of use. Considering a set of 22,000 terms (as a median, see Figure 2 above) would be impractical for the purpose of assigning categories. Therefore, we circulated the unique corpora for

assessment and category assignment, as their size is more manageable (median 1,800 terms, in our sample).

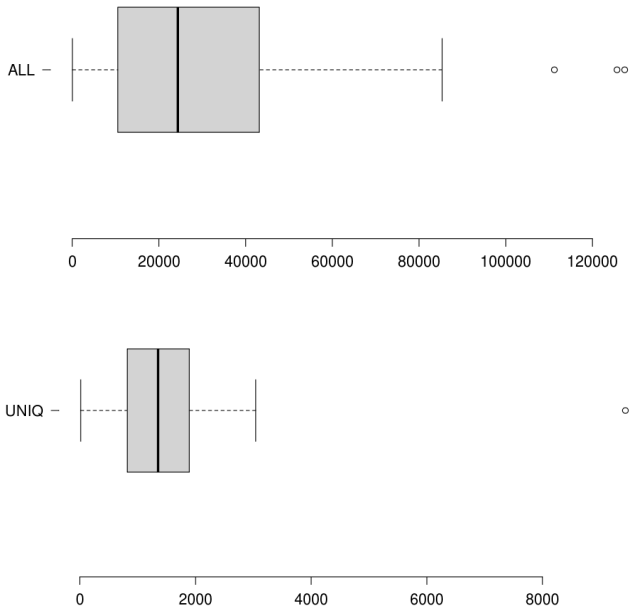

Fig. 2. Boxplots of complete and unique corpora sizes in the sample

C. Domain Modeling with LDA

For each system, all the Java classes were reduced to the complete corpus of terms. All these terms were then considered to create a model implementing the Latent Dirichlet Allocation (LDA) algorithm. Python was the language used to program the models, and the *gensim* NLP package helped in the machine learning side of it. Through the LDA we extracted the main topics emerging from the corpus of a software project, using a Natural Language Processing approach termed the Term-frequency-inverse document frequency (TF-IDF). In NLP, TF-IDF is another way to judge the topic of a text (in our case, the source code of a class) by the words it contains. With TF-IDF, words are given weight, because TF-IDF measures relevance, not frequency. This is a good representation of the source code contained in the Java classes, where the same terms can appear multiple times (as shown in the list of complete corpus of the code snippet above).

As an example, we extracted the topics for the *okhttp* project[6]. The LDA model was instrumented to generate 5 topics, as shown in the box below. The key terms, and their weights, are assigned to each of the extracted topics. In the case of the *okhttp* project, the topics are strongly related to the Networking category.

Topics extracted with the LDA approach

Topic 0: 0.003*"**stream**" + 0.003*"**bodi**" + 0.003*"**header**" + 0.003*"**content**" + 0.003*"**id**" + 0.002*"**benchmark**" + 0.002*"**type**" + 0.002*"**ssl**" + 0.002*"**socket**" + 0.002*"**stori**"

Topic 1: 0.002*"**entiti**" + 0.002*"**url**" + 0.002*"**proxi**" + 0.002*"**slack**" + 0.002*"**event**" + 0.001*"**frame**" + 0.001*"**filter**" + 0.001*"**client**" + 0.001*"**equal**" + 0.001*"**session**"

Topic 2: 0.005*"**cooki**" + 0.004*"**header**" + 0.004*"**interceptor**" + 0.003*"**chain**" + 0.003*"**url**" + 0.002*"**bodi**" + 0.002*"**certif**" + 0.002*"**content**" + 0.002*"**client**" + 0.002*"**timeout**"

Topic 3: 0.005*"**cach**" + 0.004*"**socket**" + 0.004*"**connect**" + 0.004*"**bodi**" + 0.003*"**rout**" + 0.003*"**server**" + 0.003*"**web**" + 0.003*"**header**" + 0.003*"**client**" + 0.003*"**url**"

Topic 4: 0.006*"**event**" + 0.006*"**socket**" + 0.005*"**certif**" + 0.005*"**address**" + 0.005*"**cach**" + 0.004*"**file**" + 0.003*"**deleg**" + 0.003*"**connect**" + 0.003*"**server**" + 0.003*"**inet**"

D. Data Sources and Unique ID

As *per* the methodology above, we extracted three data sets for each of the 50 systems analysed:

- the corpus of *unique* terms;
- the list of topics as extracted by the steps provided in III-C; and
- the ReadMe file that accompanies the project once is distributed onto GitHub.

[6]https://github.com/square/okhttp

Each of these data sources was given an ID as shown in the excerpt of Table I. Overall, we extracted 150 unique data sources from the 50 analysed systems.

E. Assignment of Data Sources to Experts

We assigned these data files for categorisation to 10 academic staff from Brunel University London: 5 lecturers, 2 senior lecturers, 1 reader and 2 professors (all co-authors of this paper) represent the experts whose opinion we mined in this experiment. They all come from the department of

TABLE I
CREATION OF DATA FILES, AND ASSIGNMENT OF A UNIQUE ID

Project name	Corpus	Topics	ReadMe
android-gpuimage	1	51	101
ansj_seg	2	52	102
arrow	3	53	103
atmosphere	4	54	104
...	...	...	...
wire	50	100	150

Computer Science and belong to either the BSEL[7] or IDA[8] research groups.

Each expert was provided with a set of 20 data files containing ReadMe files; 20 data files containing the (unique) corpora of systems, and 20 data files containing the topics. The process of assignment of data files is summarised in Table II. As highlighted in the table, each individual data source was contained in 4 sets: this implies that each data source was analysed by 4 different researchers.

Each researcher was supplied with a unique set of data files and they were required to assign each data file to one (or more) application domain. As the list of domains, we adopted what has been historically used by the SourceForge.net repository to classify the hosted projects:

1) Communications
2) Database
3) Desktop Environment
4) Education
5) Formats and Protocols
6) Games/ Entertainment
7) Internet
8) Mobile
9) Multimedia
10) Office/Business
11) Other/Nonlisted Topic
12) Printing
13) Religion and Philosophy
14) Scientific/ Engineering
15) Security
16) Social sciences
17) Software Development
18) System
19) Terminals
20) Text Editors

In cases where they could not be fitted to any category, an acceptable option was to tick a 'none/unidentified' category.

IV. RESULTS

This section reports the results that we gathered from each experts' assignments. In order to summarise the results, we defined the following four levels of agreement:

- *Perfect*: 4 experts agreed on the application domain of a particular data file;
- *Strong*: the agreement is between 3 experts;
- *Standard*: the agreement is between 2 experts
- *Null*: there is no agreement between experts.

It is important to note that the aim of this exercise is not to precisely identify the correct application domain of a specific system, but to detect whether there is agreement between experts in the assignment of a data file to a pre-defined domain. Table III illustrates the results that we gathered from the experts. As a recap of the research questions:

1) RQ1: is the lexicon an acceptable substitute for the description of a software system, for the purpose of assigning it to a domain?
2) RQ2: similarly, are the topics (as extracted by LDA) acceptable substitutes?

On average, we found that the ReadMe files showed the better agreement between experts: on average, a minimum of two experts agree on the application domain described in the ReadMe files. Being the baseline technique, this was expected. What also emerged is that the plain description of the ReadMe allowed for more variance: for 37 of the selected systems, the experts assigned more than one domain.

The LDA topics and the corpora scored less on average and this is reflected by the number of times no expert agreement was reached (the *Null* row in Table III). In these cases, we observed less variance: for 23 and 24 systems, respectively, the experts noted more more than one application domain. What is interesting to note is that the median (i.e., the central value of the distribution) and the mode (i.e., the most likely value) are the same for all the techniques considered.

A. Intersection of No-Agreements

In Figure 3, we display the projects for which there was no agreement: as mentioned in Table III, for 12 projects there was no agreement using the corpora; for 14 projects, no agreement using the topics; while for 6 projects there was no agreement using the ReadMe files.

Considering the intersection of those sets, 4 projects showed no agreement for either corpora or LDA topics, although there was agreement when examining the ReadMe files. Two projects in our sample there showed no agreement for either of the three data sources. In all the other cases, at least one data type *per* system saw an agreement on application domains.

B. Comparison with a Random Assignment

In order to test how our process differ from a random assignment of application domains, we automated the extraction of 300,000 random domain assignments using the in-built R random and replication features[9].

We plotted the distribution of values obtained for the average agreement on domains (Figure 4): it shows that a random allocation of domains is clearly worse than our approach.

[7]Brunel Software Engineering Lab, http://www.brunel-sweng.org/
[8]Intelligent Data Analysis Brunel, https://ida-research.net/

[9]Using the R *replicate* function as in the following instruction: *data.frame(replicate(4,sample(1:20,50,rep=TRUE)))*

TABLE II
ASSIGNMENT OF DATA FILES TO INDIVIDUAL SETS (ONE SET PER EXPERT). HIGHLIGHTED THE SAME DATA FILE (4) ASSIGNED TO DIFFERENT EXPERTS.

Set_1	Set_2	Set_3	Set_4	Set_5	Set_6	Set_7	Set_8	Set_9	Set_{10}
1	1	1	1	2	2	2	2	3	3
3	3	4	4	4	4	5	5	5	5
6	6	6	6	7	7	7	7	8	8
...	...	...	...	...	...	...	...	...	...
148	148	149	149	149	149	150	150	150	150

TABLE III
RESULTS AND LEVELS OF AGREEMENT, PER TYPE OF DATA SOURCE

Agreement level	Corpora	Topics	ReadMe
Perfect	1	4	6
Strong	3	7	13
Standard	34	25	25
Null	12	14	6
Average	1.62	1.74	2.26
Median	2	2	2
Mode	2	2	2

The vast majority of median and mode values are zero. We conclude that our methodology achieved a better performance than a random assignment process.

V. DISCUSSION

In this section we add further insights as part of our discussion, dividing it in various themes.

A. Domains and structural characteristics

In a prior study [**?**], we collected empirical results showing that projects from the same domain exhibit common structural properties in terms of the C&K metrics [8]. For interested stakeholders, this can imply that the structure of a software system (and its development and future maintenance) depends on domain-based factors common to projects in the same domain. For example, projects from different domains making use of exception handling differently [22], [5].

These findings make the identification and assignment to domains an important step to provide tailored, specific evidence to the evolution of a software system.

B. Precision vs Reliability

The purpose of this study was not to explore the *precision* of the assignments, but the reliability (agreement) of expert opinion: as such, the focus of this paper was not aimed at precisely identifying the application domains of a group of software systems. Instead, we tried to assess whether one technique is more likely to obtain an alignment in expert opinions. This is because a precise description of each categories is still missing from the literature, hence their boundaries are not clearly defined.

C. Technology-specific key terms

It is also important to note that the presence of specific, domain-based keywords was a key factor for the assignment of data sources to categories. For example, the presence of the JNI, SASL or postgreSQL as terms helped formulating a category straight away. These keywords would obviously require a domain expert to evaluate, and to correctly assign to the right category.

D. Foreign language documentation

The presence of foreign languages (e.g., Chinese) is an interesting scenario, and a further case in favour of extracting topics from the source code, since programming standards and syntax are typically based on the English language. As an example, the *ansj_seg* project has a ReadMe file written in Chinese, therefore it was not possible to assign it to any category (as none of the experts analysing that speak or read Chinese). The corpora and the topics extracted from the source code, on the other hand, allowed the experts to formulate an opinion on its category.

The same situation happened with the *java-learning*, *jeecg-boot* and *weixin-java-tools* projects: the experts could not assign the ReadMe files to any category, but corpora and topics allowed a categorisation.

E. Overfull and underfull categories

From the gathered results it is possible to notice that some categories (e.g., *Religion*, *Social Sciences*, *Formats and Protocols*) were never chosen by the experts, whereas other categories (e.g., *Software Development*, *Mobile*) were selected most often. This result demonstrates that top-down taxonomies can over-represent some categories. It also clarifies the need of a proper taxonomy, potentially from the bottom-up, and driven by source code. Such a taxonomy would allow (i) comparisons between projects and (ii) contained-in tests, in order to test whether the corpus of a software system belongs to one or another category.

VI. THREATS TO VALIDITY

In this section we present the threats to validity of this study, dividing them in *external*, *internal* and *construct* threats. Strategies to minimize the effect of each threat are outlined.

A. External validity

This paper presents the results of an empirical analysis that should be applicable to all OSS projects. We cannot generalise our findings on any other sample of OSS projects, or from any other repository. This is especially true given that our sample was obtained through a stratified sampling technique. That had the effect to extract projects whose size is larger than the average GitHub project.

Although we cannot claim the generalizability of our results, we believe that also smaller projects can benefit from our approach: documentation for small-to-medium sized OSS projects can be seriously lacking [7], [19], [3]. Using the corpora (as extracted from the source code) can be beneficial to inform the classification of software systems where documentation is lacking.

B. Internal validity

Our implementation of the LDA algorithm has shown that it is very consistent in helping to identify certain categories (*Software Development*, *System*, *Mobile*, *Internet*). The topics extracted are less sensible to smaller categories (*Religion*, *Social Sciences*, *Printing*), that in general attract less projects. Instead of tuning a stronger version of the LDA algorithm, we believe that there should be a better attempt at taxonomies: this would indicate that some categories (e.g., *Printing*) are typically sub-categories within a larger category (e.g., *System*). We expand this aspect in the Further Work section below.

C. Construct validity

While we asked the experts to provide a category for each data source, we did not query their opinion on two important aspects: (i) the *ease* of assigning each piece of data to one or more categories; and the *confidence* in doing so. From informal conversations with the researchers, we gathered that the topics were a simpler way to interact with the assignment exercise, while the ReadMe files were the ones with more confidence. This is in line with the levels of agreement that we observed when gathering the results of the category selections.

The second threat to construct validity is based on our implementation of the LDA algorithm. We tuned the algorithm in order to get trained in a number of iterations, and extract only a limited number of topics (4). As reported by one of the experts: '*I found the TOPICS part rather tricky. It contained sparse data, hence although I have entered some domains, I feel I based my decision only on intuition, and not data.*'. As a remedy to this threat, it's important to notice that this number could be easily increased, but it should be tailored to the data source. We made the LDA script available for inspection and comments.

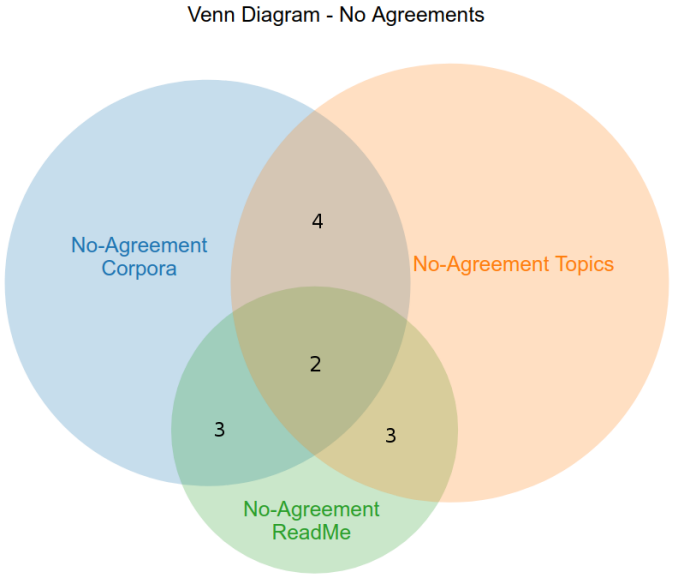

Fig. 3. Intersection of sets where no agreement was reached

VII. Conclusion and Further Work

This paper was built on top of the assumption that the plain-text description of a software system is the better way for researchers and practitioners to assign a software system to a category. We argued that, in case that description is unclear, or unreadable, a machine learning approach could help extracting the keywords, or the topics, from a system and apply to categories. We extracted the plain description of a software systems, alongside the keywords of its source code and the topics emerging from these keywords. We asked 10 experts to assign each of those data sources to an application domain, and collected their agreements. We found that, on average, the plain description has a better agreement level, but a larger variance. We also found that the median and mode values were similar across the three techniques used. These results are encouraging: we showed that the keywords and the topics are valuable substitutes to the plain descriptions, when trying to agree on the application domain of a software system.

We believe that this work opens two important avenues of further research: the first is the creation and the assessment of a bottom-up, source-driven software taxonomy. This would include branches of common sub-categories (e.g., the sub-category *Networking* that applies both to *Software Development* and *System* super-categories); as well as families of categories (e.g., the *Mobile* family) with parallel sub-categories.

The second avenue for further research is based on how software systems differ, and based on their application domains. We started gathering some initial evidence, that has pointed to different structural characteristics, when grouping systems based on their application domains [**?**]. This urges to further consider application domains as units of shared development practices, and similar characteristics, that would further boost the establishment of a software taxonomy.

References

[1] ACM Computing Classification System ToC, available-at = https://www.acm.org/about-acm/class, note = Accessed: 2019-12-15

[2] Asaduzzaman, M., Ahasanuzzaman, M., Roy, C.K., Schneider, K.A.: How developers use exception handling in java? In: Proceedings of the 13th International Conference on Mining Software Repositories. pp. 516–519. ACM (2016)

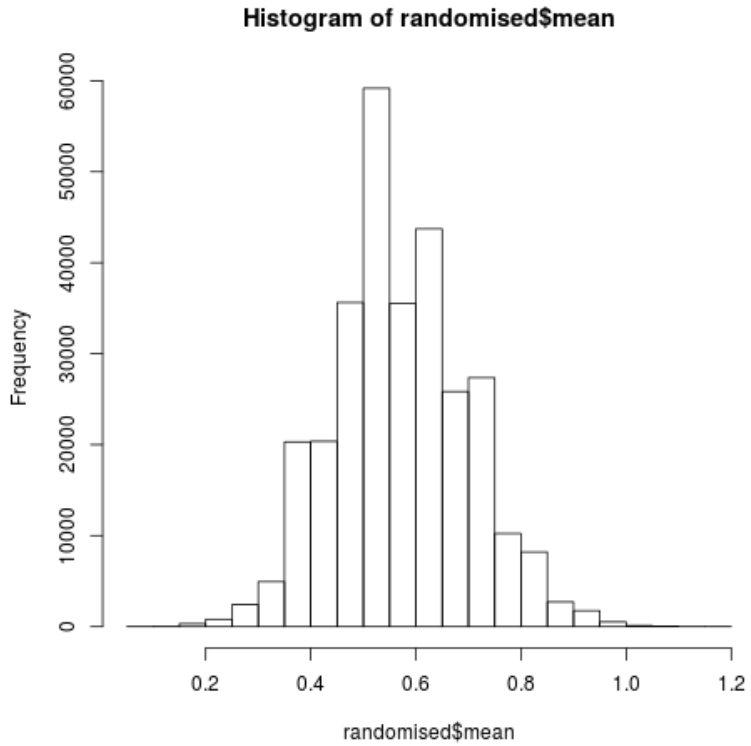

Fig. 4. Histogram of mean number of agreements, as extracted by a random process

[3] Barkmann, H., Lincke, R., Löwe, W.: Quantitative evaluation of software quality metrics in open-source projects. In: 2009 International Conference on Advanced Information Networking and Applications Workshops. pp. 1067–1072. IEEE (2009)

[4] Borges, H., Hora, A., Valente, M.T.: Understanding the factors that impact the popularity of github repositories. In: 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 334–344. IEEE (2016)

[5] Cabral, B., Marques, P.: Exception handling: A field study in java and. net. In: European Conference on Object-Oriented Programming. pp. 151–175. Springer (2007)

[6] Capiluppi, A., Ajienka, N.: The relevance of application domains in empirical findings. In: Proceedings of the 2nd International Workshop on Software Health. pp. 17–24. IEEE Press (2019)

[7] Chen, J.C., Huang, S.J.: An empirical analysis of the impact of software development problem factors on software maintainability. Journal of Systems and Software **82**(6), 981–992 (2009)

[8] Chidamber, S.R., Kemerer, C.F.: A metrics suite for object oriented design. IEEE Transactions on software engineering **20**(6), 476–493 (1994)

[9] Deshpande, A., Riehle, D.: The total growth of open source. In: IFIP International Conference on Open Source Systems. pp. 197–209. Springer (2008)

[10] Easterbrook, S., Singer, J., Storey, M.A., Damian, D.: Selecting empirical methods for software engineering research. In: Guide to advanced empirical software engineering, pp. 285–311. Springer (2008)

[11] Fayad, M., Schmidt, D.C.: Object-oriented application frameworks. Communications of the ACM **40**(10), 32–38 (1997)

[12] Forward, A., Lethbridge, T.C.: A taxonomy of software types to facilitate search and evidence-based software engineering. In: Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds. p. 14. ACM (2008)

[13] German, D.M., Di Penta, M., Gueheneuc, Y.G., Antoniol, G.: Code siblings: Technical and legal implications of copying code between applications. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. pp. 81–90. IEEE (2009)

[14] German, D.M., Manabe, Y., Inoue, K.: A sentence-matching method for automatic license identification of source code files. In: Proceedings of the IEEE/ACM international conference on Automated software engineering. pp. 437–446. ACM (2010)

[15] Glass, R.L., Vessey, I.: Contemporary application-domain taxonomies. IEEE Software **12**(4), 63–76 (1995)

[16] Kagdi, H., Gethers, M., Poshyvanyk, D.: Integrating conceptual and logical couplings for change impact analysis in software. Empirical Software Engineering **18**(5), 933–969 (2013)

[17] Karus, S., Gall, H.: A study of language usage evolution in open source software. In: Proceedings of the 8th Working Conference on Mining Software Repositories. pp. 13–22. ACM (2011)

[18] Ko, A.J.: Mining the mind, minding the mine: grand challenges in comprehension and mining. In: Zaidman, A., Kamei, Y., Hill, E. (eds.) Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018. p. 118. ACM (2018). https://doi.org/10.1145/3196398.3196477, https://doi.org/10.1145/3196398.3196477

[19] Lerner, J., Tirole, J.: The open source movement: Key research questions. European economic review **45**(4-6), 819–826 (2001)

[20] Marcus, A., Sergeyev, A., Rajlich, V., Maletic, J., et al.: An information retrieval approach to concept location in source code. In: Reverse Engineering, 2004. Proceedings. 11th Working Conference on. pp. 214–223. IEEE (2004)

[21] Nagappan, M., Zimmermann, T., Bird, C.: Diversity in software engineering research. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. pp. 466–476. ACM (2013)

[22] Nakshatri, S., Hegde, M., Thandra, S.: Analysis of exception handling patterns in java projects: An empirical study. In: Proceedings of the 13th International Conference on Mining Software Repositories. pp. 500–503. ACM (2016)

[23] Osman, H., Chiş, A., Corrodi, C., Ghafari, M., Nierstrasz, O.: Exception evolution in long-lived java systems. In: Proceedings of the 14th International Conference on Mining Software Repositories. pp. 302–311. IEEE Press (2017)

[24] Spinellis, D.: Half-century of unix: history, preservation, and lessons learned. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). pp. 1–1. IEEE (2017)
[25] Tian, K., Revelle, M., Poshyvanyk, D.: Using latent dirichlet allocation for automatic categorization of software. In: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. pp. 163–166. IEEE (2009)
[26] Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A., Gall, H.C.: Context is king: The developer perspective on the usage of static analysis tools. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 38–49. IEEE (2018)
[27] Wermelinger, M., Yu, Y.: An architectural evolution dataset. In: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on. pp. 502–505. IEEE (2015)

APPENDIX A
LIST OF ANALYSED PROJECTS

Project	**URL**
android-gpuimage	https://github.com/cats-oss/android-gpuimage
ansj_seg	https://github.com/NLPchina/ansj_seg
arrow	https://github.com/apache/arrow
atmosphere	https://github.com/Atmosphere/atmosphere
autorest	https://github.com/Azure/autorest
blurkit-android	https://github.com/CameraKit/blurkit-android
bytecode-viewer	https://github.com/Konloch/bytecode-viewer
cglib	https://github.com/cglib/cglib
dagger	https://github.com/square/dagger
ExpectAnim	https://github.com/florent37/ExpectAnim
graal	https://github.com/oracle/graal
graphql-java	https://github.com/graphql-java/graphql-java
halo	https://github.com/halo-dev/halo
HikariCP	https://github.com/brettwooldridge/HikariCP
http-request	https://github.com/kevinsawicki/http-request
interviews	https://github.com/kdn251/interviews
java-learning	https://github.com/brianway/java-learning
Java-WebSocket	https://github.com/TooTallNate/Java-WebSocket
jeecg-boot	https://github.com/zhangdaiscott/jeecg-boot
jeesite	https://github.com/thinkgem/jeesite
JFoenix	https://github.com/jfoenixadmin/JFoenix
jna	https://github.com/java-native-access/jna
joda-time	https://github.com/JodaOrg/joda-time
jodd	https://github.com/oblac/jodd
JsonPath	https://github.com/json-path/JsonPath
junit4	https://github.com/junit-team/junit4
librec	https://github.com/guoguibing/librec
light-task-scheduler	https://github.com/ltsopensource/light-task-scheduler
mal	https://github.com/kanaka/mal
mall	https://github.com/macrozheng/mall
mosby	https://github.com/sockeqwe/mosby
mybatis-plus	https://github.com/baomidou/mybatis-plus
nanohttpd	https://github.com/NanoHttpd/nanohttpd
NullAway	https://github.com/uber/NullAway
parceler	https://github.com/johncarl81/parceler
PermissionsDispatcher	https://github.com/permissions-dispatcher/
Phoenix	https://github.com/Yalantis/Phoenix
quasar	https://github.com/puniverse/quasar
requery	https://github.com/requery/requery
retrofit	https://github.com/square/retrofit
retrolambda	https://github.com/luontola/retrolambda
Sentinel	https://github.com/alibaba/Sentinel
simplify	https://github.com/CalebFenton/simplify
swagger-core	https://github.com/swagger-api/swagger-core
tcc-transaction	https://github.com/changmingxie/tcc-transaction
symphony	https://github.com/b3log/symphony
testcontainers-java	https://github.com/testcontainers/testcontainers-java
UltimateRecyclerView	https://github.com/cymcsg/UltimateRecyclerView
weixin-java-tools	https://github.com/chanjarster/weixin-java-tools
wire	https://github.com/square/wire