ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/334465949

The Relevance of Application Domains in Empirical Findings

Preprint · July 2019

DOI: 10.1145/nnnnnnn.nnnnnn

CITATIONS
0

READS
83

2 authors:

Andrea Capiluppi
Brunel University London
100 PUBLICATIONS **1,194** CITATIONS
SEE PROFILE

Nemitari Ajienka
Nottingham Trent University
9 PUBLICATIONS **25** CITATIONS
SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project Open Source modelling and investigation View project

Project Replication studies View project

All content following this page was uploaded by Nemitari Ajienka on 15 July 2019.

The user has requested enhancement of the downloaded file.

The Relevance of Application Domains in Empirical Findings

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

The term '*software ecosystem*' refers to a collection of software systems that are related in some way. Researchers have been using different levels of aggregation to define an ecosystem: grouping them by a common named project (e.g., the Apache ecosystem); or considering all the projects contained in online repositories (e.g., the GoogleCode ecosystem).

In this paper we propose a definition of ecosystem based on application domains: software systems are in the same ecosystem if they share the same application domain, as described by a similar technological scope, context or objective. As an example, all projects implementing networking capabilities to trade Bitcoin and other virtual currencies can be considered as part of the same "cryptocurrency" ecosystem.

Utilising a sample of 100 Java software systems, we derive their application domains using the Latent Dirichlet Allocation (LDA) approach. We then evaluate a suite of object-oriented metrics per ecosystem, and test a null hypothesis: '*the OO metrics of all ecosystems come from the same population*'.

Our results show that the null hypothesis is rejected for most of the metrics chosen: the ecosystems that we extracted, based on application domains, show different structural properties.

From the point of view of the interested stakeholders, this could mean that the health of a software system depends on domain-dependent factors, that could be common to the projects in the same domain-based ecosystem.

KEYWORDS

FOSS, Application Domains, Latent Dirichlet Allocation, Machine Learning, Expert Opinions, OO (object-oriented)

ACM Reference Format:
Andrea Capiluppi and Nemitari Ajienka. 2019. The Relevance of Application Domains in Empirical Findings. In *Proceedings of ICSE2019 (SoHeal).* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Once the dependencies between different software projects have become documented and clear, it became also evident that a definition of software *ecosystem* was needed, for discussion and to agree upon: the earliest definition that has been proposed states that a software ecosystem is *A set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them* [15].

The operationalisation of this definition has been attempted in different directions in the last few years: the term ecosystem has been applied to collections of software projects under the same macro-project (Apache [3], Eclipse [21], Gnome [38], etc); and to collections of software projects that are hosted under the same online portal (SourceForge [19], GitHub [25, 42], etc). In general, there has been very little convergence on what indeed is a software ecosystem, as highlighted in a systematic literature review in [26].

In this paper we propose a novel definition for software ecosystem based on the context, or application domain, that software systems implement in their requirements. We posit that systems are similar, or connected, if they share the same topic, or domain: as an example, we consider all the Github.com projects implementing a bitcoin wallet[1] as under the same domain-driven ecosystem.

While the primary 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 empirical software engineering research. As in the example reported in [31], 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, and irrespective of their language.

Tools that capture the topics of software systems are and have been proposed in the past: CLAN [29], CrossSim [33], MudaBlue [18] and RepoPal [43] are some of the most cited tools, with various levels of precision and accuracy. Even if such tools are available for researchers and practitioners, their usage to practically inform development has been so far quite limited.

In this paper we extracted the domains of the software systems by directly analysing their source code and comments: by eliminating the common constructs and keywords of the Java language, we pulled the lexical content out of all the classes composing a project's source code and comments. After stemming this content, a Python implementation of the LDA algorithm lists the most likely topics that are contained within the code. These projects were then clustered into groups, based on their domains. Finally, a set of Object-Oriented (OO) structural metrics are extracted for the projects, with the aim to answer the following research question:

> RQ: are OO metrics sensitive to application domains?

[1] https://github.com/search?q=bitcoin+wallet

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
SoHeal, May 2019, Montreal, Canada
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

We articulate this paper in the following parts: section 3 introduces the methodology, describing how the corpus was extracted from the source code, and how the LDA was instrumented to output the topics contained in the software systems. Section 4 reports the results, including the statistical tests of whether different application domains come from the same distribution or not. We also discuss the implications for software health there. Section 2 summarises the related work in the area of empirical research and application domains; section 5 concludes, and shows what we have planned as future work.

2 RELATED WORK

Wermelinger and Yu [40] posit 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 in particular systems belonging to different domains.

This is as Software Ecosystems have been defined as "a collection of software products that have some given degree of symbiotic relationships" [30]. According to Bosch and Bosch-Sijtsema [5] a software ecosystem consists of "a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs". A different definition is provided by Manikas and Hansen "a software ecosystem is the interaction of a set of actors on top of a common technological platform that results in a number of software solutions or services" [26].

These definitions of Ecosystems imply that software belonging to specific application domains will share similarities (e.g., in their structure, development and evolution). Furthermore, Ecosystems concern software in some form (software systems, products, services, or a software platform) and includes some degree of relationship either "common evolution" [35], "business", "symbiotic" or "technical" [26] in domains and this highlights the importance of analysing empirical results in the software engineering field on software domain-by-domain basis. Certain studies have been aimed at defining architectural patterns that form the foundations to define and implement various software ecosystems (e.g., eLearning ecosystem [12], farm software ecosystem [20]).

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 [11, 17]. In view of this and to reduce the effort required in manual categorisation of software projects, Tian *et al.* [37] proposed a new technique based on text mining to categorise software projects irrespective of the programming language used in their development.

Callau *et al.* [7] studied the use of dynamic programming features such as method and class creation and removal at run-time e.g., during testing and how much these features are actually used in practice, whether some are used more than others, and in which kinds of projects. Their results revealed three application domains prominent with the usage of dynamic programming features: (i) *user interface applications*, which make heavy usage of dynamic method invocation as a lightweight form of an event notification system, (ii) *frameworks* that communicate with databases or implement object databases, which make heavy usage of serialization and de-serialization of objects and (iii) low-level system *support code* that uses uses object field reads and writes to implement copy operations, saving the state of the system to disk, and convert numbers and strings from objects to compact bit representation.

In a different study [10] software coupling metrics were studied based on software categories to identify any impact of software categories on coupling metrics (CBO, DAC[2], ATFD[3] and AC[4]). The authors emphasised the need to pay special attention to software categories when comparing systems in distinct categories with predefined thresholds already available in the literature. For example, empirical results from the study revealed that out of ten distinct categories selected (including Audio and Video, Graphics, Security and Games) there is a different level of coupling among the different categories. Games had a higher coupling level while the Development category showed less coupling than others. Statistical tests conducted at a 0.01 significance level supported these results which indicate the importance of analysing software engineering research results by domain/category.

Linares-Vasquez *et al.*[23] analysed the energy usage of API method calls in 55 different Android apps from different categories such as Tools, Music, Media and others. Results revealed GUI and image manipulation apps made use of the highest number of energy-greedy API method calls followed by Database apps. Both categories represented 60% of the energy-greedy APIs in the studied sample. This shows a trend in this study and the previous study on the usage of dynamic programming features [7] in terms of the significant energy and memory usage of software in Databases category wherein database-based software made heavy usage of serialization and de-serialization of objects.

In a related study, the focus is on energy management in Android applications [2] with an analysis of different power management commits (including Power Adaptation, Power Consumption Improvement, Power Usage Monitoring, Optimizing Wake Lock, Adding Wake Lock and Bug Fix and Code Refinement). The studied projects were clustered into 15 categories. The top three categories in terms of the number of power management commits were found to be Connectivity, Development and Games.

Previous studies [1, 6, 32] revealed that projects from different domains use exception handling differently, and that poor practices in writing exception handling code are widespread. In a study on Java projects by Osman *et al.* [34] they 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.

[2] Data Abstraction Coupling
[3] Access to Foreign Data
[4] Afferent Coupling

In general, different results have been observed when more attention is paid to the categorisation of analysed software projects.

3 METHODOLOGY

In this section, we describe the projects selected as a sample for this paper, and what metrics were extracted for each project. Figure 1 summarises the toolchain used in this paper.

3.1 Sampling Java Projects

Leveraging the GitHub.com repository, we collected the project IDs of the 100 most successful Java projects hosted on GitHub.com as case studies[5]. The "success" of the projects is determined by the number of *stars* received by the community of GitHub users and developers, as a sign of appreciation. We used this approach to stratified sampling because the projects obtained by this filter are likely to be used by a large pool of users [9], and potentially have a good intake of new developers [39]. Smaller projects are less likely to be sampled by this stratified approach.

The source code of the selected systems was downloaded for the analysis: only the 'master' branch of the systems was considered.

3.2 Extracting the corpus from Java classes

We extract the lexical content (e.g., its *corpus*) of each Java class in two ways: (i) by considering their class names; and (ii) parsing their code and considering the 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.

Pre-processing of the system corpus is performed to eliminate common keywords, stop words, split and to stem class names [28]. We do not consider as a term any of the Java-specific keywords (e.g., *if*, *then*, *switch*, etc.)[6]. Additionally, the camelCase or PascalCase notations are first decoupled in their components (e.g., the class constuctor *InvalidRequestTest* produces the terms *invalid*, *request* and *test*).

As an example, for the lines of code shown in Figure 2 (the UrSQLEntry.java class from the UrSQL project), we derive the following *complete* corpus using an information retrieval tool developed in Java: {*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*}. The tool is available upon request.

3.3 Domain Modeling with LDA

For each system, all the Java classes were reduced to a corpus of terms. All these terms were 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.

[5]The list of projects is available at https://figshare.com/projects/Domains_and_OO_metrics/57401

[6]The complete list of Java reserved words that we considered is available at https://en.wikipedia.org/wiki/List_of_Java_keywords. The `String` keyword was also considered as a reserved word, and excluded from the text parsing.

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 (see Figure 2).

As an example, for the *okhttp* project[7], the LDA model produces the following topics from the corpus of its Java classes:

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**"

3.4 Assignment to Domains

In order to assign the lexical content of a software project to one category, or domain, there are a few options: either to perform a semi-automated task, or to execute a machine learning approach. The semi-automated task includes a step where the lexical content is extracted by a lexicon parser and summarised in the most prominent *topics*; and a manual step where the topics are assigned to a domain by both authors of the paper.

For this paper, we preferred to conduct a semi-automated analysis, since (i) the project sample is manageable, and, most importantly (2) a knowledge base around application domains and lexicon is not available yet. We will summarise our contribution on the machine learning process in the further work section below.

The second step of our approach can also be achieved in a different way from ours: a software project to an application domain can be achieved 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 reported in [4] follows that approach: the dataset contains 5,000 GitHub project (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, so 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:

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

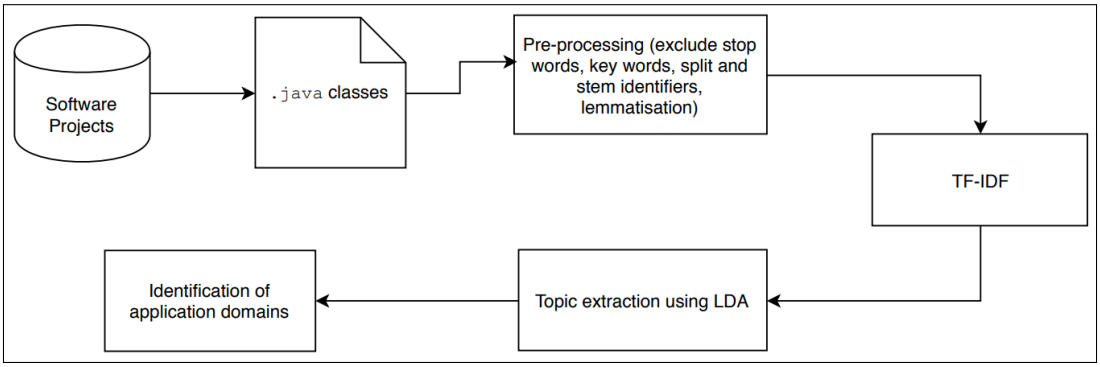

Figure 1: Toolchain adopted from data extraction to domain identification

```
1  package tmacsoftware.ursql;
2
3  public class UrSQLEntry
4  {
5
6      private String key;
7      private String value;
8      private String firstKey;
9      private String firstValue;
10     private UrSQLEntity entity;
11
12     public UrSQLEntry()
13     {
14     }
15
16     public UrSQLEntry(String query)
17     {
18         String[] split = query.split
   (UrSQLController.KEY_VALUE_SEPARATOR);
19         this.key = split[0];
20         this.value = split[1];
21         this.firstKey = this.key;
22         this.firstValue = this.value;
23     }
```

Figure 2: UrSQLEntry.java Source Code Snippet

- 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)

In our methodology, we use an NLP-based automatic approach to extract the topics from the software systems, and a manual approach to assign the projects to pre-existing categories. As the list of categories, we adopted in fact 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*}.

We did not assign domains automatically, as the approaches developed in [29], [18] or [43]. We relied on the topics extracted by the LDA implementation to inform the category of belonging. In case of disagreement in terms of the application domain assigned, we reconciled our views into a consolidated final version of the domain assignments. The process to reconcile the views on application domains is made available in the data repository, for inspection. As an example, from the topics in the box above, the *okhttp* project was assigned to the *Internet* application domain.

3.5 Clustering, Metrics and Statistical Testing

We group all the systems belonging to the same domain in the same cluster, and gather their metrics in a common bucket. We then test the whether these clusters come from the same distribution, by means of a Kolmogorov-Smirnov test (KS) [41]. The null hypothesis H_0 states that '*the clusters are drawn from the same distribution*'.

The metrics extracted for the projects are well-known structural OO attributes (NOC, DIT, CBO, RFC, WMC, LCOM, NIM, IFANIN, NIV) [24]. Metrics were extracted using the Scitools Understand tool[8]: abstract classes, interfaces and inner classes were also considered in the data extraction.

We performed the Kolmogorov-Smirnov tests for each OO attribute measured, and considering the Java classes in each cluster: a threshold value ($\alpha = 0.05$) was selected for the p-values of the statistical test. The Bonferroni correction was applied, since we run multiple tests at the same time, thus bringing the threshold to $\alpha_B = 0.00855$.

4 RESULTS AND IMPACT

Figure 3 shows the distribution of application domains in the sample of 100 Java projects, as extracted by the LDA algorithm, then assigned by the authors of the paper, finally agreed between authors to ensure consistency.

A few of the basic domains, as used by SourceForge.net, are completely absent from our sample: *Desktop Environment*, *Education* or *Text Editor* (and a few others) are not represented when sampling projects based on their success (e.g., usage or further development).

On the other hand, there are 4 domains that are more prominent than others: *Internet* (with 27 projects), *Mobile* (11), *Multimedia* (11) and *Software Development* (33). For the statistical analysis, we use only these 4 domains to find differences between the distributions of metrics: using smaller sized clusters would suffer from a small effect size, and the relative results would be less relevant.

[8] https://scitools.com/

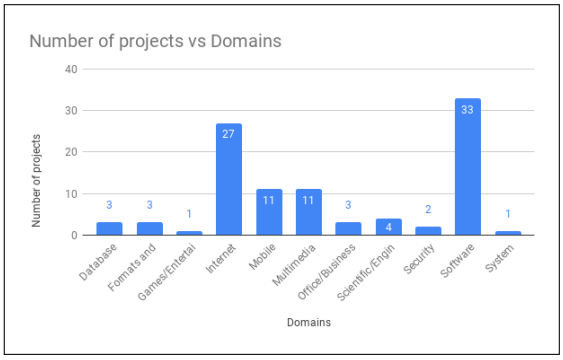

Figure 3: Domains extracted with the LDA approach

The null hypothesis states that *the OO metrics of the Java classes come from the same population*. This implies that the application domains should not induce any variability in the results. As an example, we named as $KS\_CBO-7-8.txt$ the file containing the p-value of the Kolmogorov-Smirnov test, when considering the CBO metric, and comparing the files belonging to the domains 7 and 8 (i.e., *Internet* and *Mobile*)

Table 1 summarises the results for the tests: with a ✓ we report when we reject the null hypothesis for **all** the metrics. Otherwise, we note for which metrics we could not reject the null hypothesis.

When considering the *Internet* domain, all its OO metrics show differences with the other domains. We could generally reject the null hypothesis: its metrics do not come from the same population. Focusing on the *Mobile* domain, we found that projects in this category could be considered different from any other domain, apart from the *NOC* attribute: it is not possible to reject the hypothesis that the projects from the *Mobile*, *Multimedia* and *Software Development* come from the same population, regarding the *NOC* attribute. For all the other attributes, we could reject the null hypothesis.

Finally, the *IFANIN* attribute plays a similar part between the projects belonging to the *Multimedia* and the *Software Development* domains. All the other OO attributes show a significant difference between domains.

	Mobile	Multimedia	Software Devel
Internet	✓	✓	✓
Mobile	x	*NOC*	*NOC*
Multimedia		x	*IFANIN*
Software Devel			x

Table 1: Results of the Kolmogorov-Smirnov test

These results are limited to 4 application domains only, given that other domains are less represented in our sample. Nonetheless, it is possible to summarise our findings as follows:

> OO metrics are generally sensitive to application domains

These findings could have a profound impact on how empirical research has been conducted in the past: researchers would need to pay a closer attention to what application domains are included in their data sampling; and results could be also generally sensitive to domains, when considering other metrics, not only the OO suite that we considered in our paper. The next section illustrates what could be the implications on the definition and measurement of software health.

4.1 Implications for Software Health

The results gathered in the empirical study show that the projects clustered around the domains that we proposed show indeed a difference in the structural metrics. According to Jansen, there exists no framework that can be used to determine the health of open source ecosystems. This is because Health is typically looked at from the scope of projects, not from an ecosystem scope [14]. Such a framework will enable developers select healthiest ecosystems to join while end-users can select robust and long-living ecosystems.

In the study, a comprehensive list of health metrics mentioned in the literature were used to form a framework. Jansen emphasised that the framework can be applied by researchers who aim to reach a goal associated with ecosystem health, such as improve activity in an ecosystem, evaluate the health of one ecosystem over another, or identifying weaknesses in an ecosystem with the aim of making it healthier [14]. The metrics included high-level product level, network level and theory based metrics such as: bug-fix time, usage, number of active projects, active contributors, contributor satisfaction and others. However, structural metrics are not covered in the framework.

In this study, we have analysed source code level metrics across domains and demonstrated the distinction in the metric patterns across domains. Putting these metrics together to add a new layer to the framework (e.g., structural level) will help to inform contributors or open-source developers about which domains to contribute to and which domains are suffering structurally.

The metrics can be mapped or compared to the "standard" as specified in the literature to detect the health of software domains in a repository such as GitHub or SourceForge. For example, the "standard" based on the literature is that the CBO metric has been linked to complexity as well as the RFC [8, 22] metric. As such, if a combination of these metrics for a domain is high then this will give end-users a view of the health of projects in that domain, while giving OSS contributors a view of which domains to support.

The work reported by [36] has already shown some correlation between pairs of metrics from the C&K suite, for example, CBO and RFC, and RFC and LCOM. If indeed there was a correlation in all the clusters analysed above, it would suggest an increased probability of falsely rejecting a null hypothesis within the clusters shown in Figure 3.

In this analysis below, we report on the correlation study between pairs of OO metrics, when clustered by application domain. The value of the correlation coefficient lies in the range $[-1; 1]$, where -1 indicates a strong negative correlation and 1 indicates a strong positive correlation. We adapt the categorisation for correlation coefficients used in [27] ($[0-0.1]$ to be *insignificant*, $[0.1-0.3]$ *low*, $[0.3-0.5]$ *moderate*, $[0.5-0.7]$ *large*, $[0.7-0.9]$ *very large*, and $[0.9-1]$ *almost perfect*) if the rank correlation coefficient proves to be statistically significant at the $\alpha = 0.01$ level.

Table 2 shows the correlations between pairs of OO metrics, when considering the projects in the identified domain-driven clusters. It becomes clear that the metrics show collinearity, but depending on the cluster considered, this collinearity could be stronger or weaker. An example of this is between the RFC and WMC attributes: for the projects in the *Internet* cluster, this association has a medium (M) strength; the association becomes large (L) when considering the projects in the *Mobile* cluster; for the projects in the *MultiMedia* category, the association becomes almost perfect (AP), therefore larger than 0.9; when considering the projects in the *Software Development* cluster, on the other hand, such collinearity becomes small (s), hence isolating these projects, and their characteristics, from the rest of the sample.

Considering the definition of the RFC and WMC attributes, a stronger correlation implies a larger complexity of the code: when the number of methods (i.e., WMC) grows in a Java class, the response for that class (i.e., the RFC) also grows. This is also an indication that more testing will be needed for that class: the projects in the MultiMedia category show a different behaviour to those belonging to the Software Development category. Evaluating the software health therefore becomes also dependent on what type of domain a project belongs to.

Based on these results, it is possible to summarise our correlation findings as follows:

> The correlation between OO metrics can be extremely sensitive to application domains

5 CONCLUSION AND FURTHER WORK

Research studies based on OSS systems have exponentially increased: only in the MSR series of conferences, the growth in number of OSS projects analysed by researchers had a thousandfold increase in the last 10 years. On the flipside, most research papers have overlooked their primary distinctive characteristics: their diversity, context, uniqueness and application domain.

In this paper, we sampled 100 GitHub projects and assigned each to an application domain, based on the key terms appearing in the source code. Using the LDA algorithm, we obtained the emerging topics from these terms, and used these topics to uniquely classify these systems into domains.

We showed that most of the sampled systems belong to a subset of domains: we also showed that the vast majority of the extracted metrics show a cross-domain difference in their behaviour. This means that it is almost always possible to reject the null hypothesis stating that they are drawn from the same population. This makes the application domains as units of analysis: projects belonging to different domains should be analysed separately, and their results considered also separately, at least when considering the OO structural attributes.

As future work, we plan to assign a domain to smaller portions of code, in particular when they are clearly connected by means of coupling. This should take into account the presence of primary and secondary application domains in the same software project. Also, the presence of more and less experienced developers should be investigated, and related to the discrepancies observed on the NOC attribute.

We have also started to populate a large database with the lexicon of thousands of software systems, by application domain. The objective will be to use such content as the oracle for other systems, to help detecting their domains through a machine learning approach.

6 ACKNOWLEDGEMENTS

We would like to thank Dr. G Destefanis for the help in the data extraction, and the extensive comments and discussion.

REFERENCES

[1] Muhammad Asaduzzaman, Muhammad Ahasanuzzaman, Chanchal K Roy, and Kevin A Schneider. 2016. How developers use exception handling in Java?. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 516–519.

[2] Lingfeng Bao, David Lo, Xin Xia, Xinyu Wang, and Cong Tian. 2016. How android app developers manage power consumption?: An empirical study by mining power management commits. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 37–48.

[3] Gabriele Bavota, Gerardo Canfora, Massimiliano Di Penta, Rocco Oliveto, and Sebastiano Panichella. 2013. The evolution of project inter-dependencies in a software ecosystem: The case of apache. In *2013 IEEE international conference on software maintenance*. IEEE, 280–289.

[4] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the factors that impact the popularity of GitHub repositories. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 334–344.

[5] Jan Bosch and Petra Bosch-Sijtsema. 2010. From integration to composition: On the impact of software product lines, global development and ecosystems. *Journal of Systems and Software* 83, 1 (2010), 67–76.

[6] Bruno Cabral and Paulo Marques. 2007. Exception handling: A field study in Java and. Net. In *European Conference on Object-Oriented Programming*. Springer, 151–175.

[7] Oscar Callaú, Romain Robbes, Éric Tanter, and David Röthlisberger. 2013. How (and why) developers use the dynamic features of programming languages: the case of smalltalk. *Empirical Software Engineering* 18, 6 (2013), 1156–1194.

[8] Shyam R Chidamber, David P Darcy, and Chris F Kemerer. 1998. Managerial use of metrics for object-oriented software: An exploratory analysis. *IEEE Transactions on software Engineering* 24, 8 (1998), 629–639.

[9] Kevin Crowston, Hala Annabi, and James Howison. 2003. Defining open source software project success. *ICIS 2003 Proceedings* (2003), 28.

[10] Lucas Batista Leite De Souza and Marcelo De Almeida Maia. 2013. Do software categories impact coupling metrics?. In *Proceedings of the 10th working conference on mining software repositories*. IEEE Press, 217–220.

[11] Amit Deshpande and Dirk Riehle. 2008. The total growth of open source. In *IFIP International Conference on Open Source Systems*. Springer, 197–209.

[12] Alicia García-Holgado and Francisco José García-Peñalvo. 2014. Architectural pattern for the definition of eLearning ecosystems based on Open Source developments. In *2014 International Symposium on Computers in Education (SIIE)*. IEEE, 93–98.

[13] Daniel M German, Massimiliano Di Penta, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2009. Code siblings: Technical and legal implications of copying code between applications. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 81–90.

[14] Slinger Jansen. 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology* 56, 11 (2014), 1508–1519.

[15] Slinger Jansen, Anthony Finkelstein, and Sjaak Brinkkemper. 2009. A sense of community: A research agenda for software ecosystems. In *2009 31st International Conference on Software Engineering-Companion Volume*. IEEE, 187–190.

[16] 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.

[17] Siim Karus and Harald Gall. 2011. A study of language usage evolution in open source software. In *Proceedings of the 8th Working Conference on Mining Software Repositories*. ACM, 13–22.

[18] Shinji Kawaguchi, Pankaj K Garg, Makoto Matsushita, and Katsuro Inoue. 2006. Mudablue: An automatic categorization system for open source repositories. *Journal of Systems and Software* 79, 7 (2006), 939–953.

[19] Raj PM Krishna and KG Srinivasa. 2011. Analysis of projects and volunteer participation in large scale free and open source software ecosystem. *ACM*

Internet								
	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT
CBO	-i							
NOC	-i	i						
NIM	s	M	i					
NIV	i	M	i	M				
WMC	i	M	i	AP	M			
RFC	-i	M	i	M	s	M		
DIT	-s	i	-i	i	-i	i	L	
LCOM	-i	s	i	M	M	M	M	s
Mobile								
	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT
CBO	i							
NOC	-i	i						
NIM	s	M	s					
NIV	i	M	i	XL				
WMC	i	M	s	AP	XL			
RFC	-i	s	s	L	M	L		
DIT	-s	s	s	s	s	s	L	
LCOM	-i	M	s	L	L	L	M	s
Multimedia								
	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT
CBO	i							
NOC	i	i						
NIM	i	s	i					
NIV	i	M	i	M				
WMC	-i	L	i	s	i			
RFC	-i	L	i	s	i	AP		
DIT	-M	s	i	s	i	i	s	
LCOM	i	s	i	M	L	i	i	s
Software Development								
	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT
CBO	i							
NOC	-i	i						
NIM	i	M	i					
NIV	i	i	i	i				
WMC	i	M	i	AP	i			
RFC	-i	s	i	s	i	s		
DIT	-s	i	-i	i	i	i	M	
LCOM	-i	M	i	M	i	M	s	s

Table 2: Correlation between OO pairs, after grouping projects within domain-driven clusters. *i* stands for insignificant correlation; *M* for medium; *L* for large; *XL* for very large; and *AP* for almost perfect

SIGSOFT Software Engineering Notes 36, 2 (2011), 1–5.

[20] Jan Willem Kruize, J Wolfert, Huub Scholten, CN Verdouw, Ayalew Kassahun, and Adrie JM Beulens. 2016. A reference architecture for Farm Software Ecosystems. *Computers and Electronics in Agriculture* 125 (2016), 12–28.

[21] Daniel Le Berre and Pascal Rapicault. 2009. Dependency management for the eclipse ecosystem: eclipse p2, metadata and resolution. In *Proceedings of the 1st international workshop on Open component ecosystems*. ACM, 21–30.

[22] Wei Li and Sallie Henry. 1993. Object-oriented metrics that predict maintainability. *Journal of systems and software* 23, 2 (1993), 111–122.

[23] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining energy-greedy api usage patterns in android apps: an empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2–11.

[24] Mark Lorenz and Jeff Kidd. 1994. *Object-oriented software metrics*. Vol. 131. Prentice Hall Englewood Cliffs.

[25] Pablo Loyola and In-Young Ko. 2014. Population dynamics in open source communities: an ecological approach applied to Github. In *Proceedings of the 23rd International Conference on World Wide Web*. ACM, 993–998.

[26] Konstantinos Manikas and Klaus Marius Hansen. 2013. Software ecosystems–A systematic literature review. *Journal of Systems and Software* 86, 5 (2013), 1294–1306.

[27] Andrian Marcus and Denys Poshyvanyk. 2005. The conceptual cohesion of classes. In *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 133–142.
[28] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, Jonathan Maletic, et al. 2004. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*. IEEE, 214–223.
[29] Collin McMillan, Negar Hariri, Denys Poshyvanyk, Jane Cleland-Huang, and Bamshad Mobasher. 2012. Recommending source code for use in rapid software prototypes. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 848–858.
[30] David G Messerschmitt, Clemens Szyperski, et al. 2005. Software Ecosystem: Understanding an Indispensable Technology and Industry. *MIT Press Books* 1 (2005).
[31] Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. 2013. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 466–476.
[32] Suman Nakshatri, Maithri Hegde, and Sahithi Thandra. 2016. Analysis of exception handling patterns in Java projects: An empirical study. In *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 500–503.
[33] P. T. Nguyen, J. Di Rocco, R. Rubei, and D. Di Ruscio. 2018. CrossSim: Exploiting Mutual Relationships to Detect Similar OSS Projects. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 388–395. https://doi.org/10.1109/SEAA.2018.00069
[34] Haidar Osman, Andrei Chiş, Claudio Corrodi, Mohammad Ghafari, and Oscar Nierstrasz. 2017. Exception evolution in long-lived Java systems. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 302–311.
[35] Walt Scacchi and Thomas A Alspaugh. 2012. Understanding the role of licenses and evolution in open architecture software ecosystems. *Journal of Systems and Software* 85, 7 (2012), 1479–1494.
[36] Giancarlo Succi, Witold Pedrycz, Snezana Djokic, Paolo Zuliani, and Barbara Russo. 2005. An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite. *Empirical Software Engineering* 10, 1 (2005), 81–104.
[37] Kai Tian, Meghan Revelle, and Denys Poshyvanyk. 2009. Using latent dirichlet allocation for automatic categorization of software. In *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE, 163–166.
[38] Bogdan Vasilescu, Alexander Serebrenik, Mathieu Goeminne, and Tom Mens. 2014. On the variation and specialisation of workloadâĂŤa case study of the Gnome ecosystem community. *Empirical Software Engineering* 19, 4 (2014), 955–1008.
[39] Georg Von Krogh, Sebastian Spaeth, and Karim R Lakhani. 2003. Community, joining, and specialization in open source software innovation: a case study. *Research policy* 32, 7 (2003), 1217–1241.
[40] Michel Wermelinger and Yijun Yu. 2015. An architectural evolution dataset. In *Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on*. IEEE, 502–505.
[41] Carolyn Whitnall, Elisabeth Oswald, and Luke Mather. 2011. An exploration of the kolmogorov-smirnov test as a competitor to mutual information analysis. In *International Conference on Smart Card Research and Advanced Applications*. Springer, 234–251.
[42] Yu Wu, Jessica Kropczynski, Patrick C Shih, and John M Carroll. 2014. Exploring the ecosystem of software developers on GitHub and other platforms. In *Proceedings of the companion publication of the 17th ACM conference on Computer supported cooperative work & social computing*. ACM, 265–268.
[43] Yun Zhang, David Lo, Pavneet Singh Kochhar, Xin Xia, Quanlai Li, and Jianling Sun. 2017. Detecting similar repositories on GitHub. In *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE, 13–23.

View publication stats