Detecting Java Software Similarities by using Different Clustering Techniques

Andrea Capiluppi[a,*], Davide Di Ruscio, Juri Di Rocco, Phuong T. Nguyen[b], Nemitari Ajienka[c]

[a]*Dept. of Computer Science, Brunel University London, UK*
[b]*Dept. of Information Engineering, Computer Science and Mathematics, University of L'Aquila, Italy*
[c]*Dept. of Computer Science, Edge Hill University, UK*

Abstract

Research on empirical software engineering has increasingly been conducted by analysing and measuring vast amounts of software systems. Hundreds, thousands and even millions of systems have been (and are) considered by researchers, and often within the same study, in order to test theories, demonstrate approaches or run prediction models. A much less investigated aspect is whether the collected metrics might be context-specific, or whether systems should be better analysed in clusters.

The objectives of this study are (i) to define a set of clustering techniques that might be used to group similar software systems, and (ii) to evaluate whether a suite of well-known object-oriented metrics is context-specific, and its values differ along the defined clusters.

We group software systems based on three different clustering techniques, and we collect the values of the metrics suite in each cluster. We then test whether clusters are statistically different between each other, using the Kolgomorov-Smirnov (KS) hypothesis testing.

Our results show that, for two of the used techniques, the KS null hypothesis (e.g., the clusters come from the same population) is rejected for most of the

*Corresponding author
Email addresses: `andrea.capiluppi@brunel.ac.uk` (Andrea Capiluppi), `{davide.diruscio,juri.dirocco,phuong.nguyen}@univaq.it` (Davide Di Ruscio, Juri Di Rocco, Phuong T. Nguyen), `nemitari.ajienka@edgehill.ac.uk` (Nemitari Ajienka)

metrics chosen: the clusters that we extracted, based on application domains, show statistically different structural properties.

The implications for researchers can be profound: metrics and their interpretation might be more sensitive to context than acknowledged so far, and application domains represent a promising filter to cluster similar systems.

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

1. Introduction

Research on empirical software engineering has increasingly used data made available in online repositories or collective efforts. The latest trends for researchers is to gather "as much data as possible" to (i) prevent bias in the representation of a small sample, (ii) work with a sample as close as the population itself, and (iii) showcase the performance of existing or new tools in treating vast amount of data.

Considering the MSR[1] series of events as an example, its researchers have constantly grown the number of systems analysed in their papers. During its 2017 edition, for instance, the joint set of papers of the main track (i.e., 64 papers overall) collected and analysed altogether over 3 million software systems. A 10-year trend with the number of software systems jointly analysed by the MSR papers is shown in Figure 1. One of the papers alone amassed some 900K systems as its case studies [1].

These always larger samples of systems have mostly overlooked their primary distinctive characteristics, their diversity, context, uniqueness and application domain. Very few works have clearly stated the similarity (or differences) between systems in the interpretation of the results, either by explicitly proposing explanations based on application domains [2, 3, 4], or by sampling the projects to be analysed from a specific, restricted topic [5].

[1]`http://www.msrconf.org/`

This present paper is based on the assumptions that a specific software system might be *similar* to others to some degree, and that there are different approaches to defining their similarity. By applying one of those approaches, a sample of software systems might get divided into subsets (or *clusters*), each containing similar systems, and showing differences with other clusters.

Understanding the similarities among software projects allows for reusing of source code and prototyping, or choosing alternative implementations [6, 7], thereby improving software quality [8]. Having access to similar software projects helps developers speed up their development process. By looking at similar Open Source Software (OSS) projects, for example, developers are able to learn how relevant classes are implemented, and in some certain extent, to reuse useful source code [6, 7, 9].

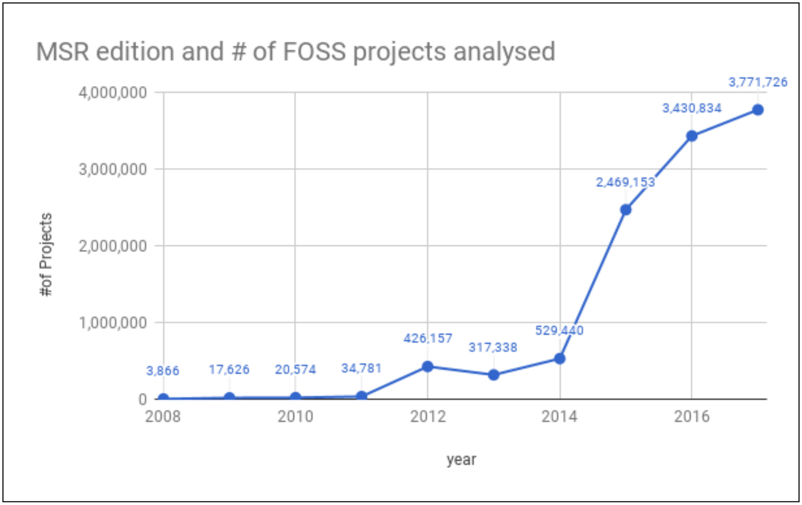


Figure 1: Cumulative number of FOSS projects per year

In the past, two software projects have been considered to be similar if they implement some features being described by the same abstraction, even though they may contain various functionalities for different domains [10]. In this paper, we group similar systems into clusters based on different approaches: first, we group them based on the similarities detected by the CrossSim algorithm [11], which has been developed as part of the EU H2020 CROSSMINER project[2]. Systems are similar, or connected, if they have a limited *distance* [11]. Second, we use the clusters as manually extracted by [12, 13], that have grouped 5,000 software systems into 6 clusters. Third, we use a Python implementation of the Latent Dirichlet Allocation (LDA) approach to automatically extract the descriptions of a project, and we group similar systems based on that extraction.

For all the clusters identified in this paper, we evaluated the metrics of an object-oriented suite, based on the work by Chidamber and Kemerer [14]. The metrics extracted for the projects are well-known structural OO attributes (NOC, DIT, CBO, RFC, WMC, LCOM, NIM, IFANIN, NIV) [15]. The aim of this paper is to answer the following research question (RQ):

RQ: are OO metrics sensitive to the context of their clusters?

In other words, do different software clusters exhibit different OO metrics? To answer that research question, we articulate this paper in the following parts: Section 2 proposes a meta-model of the reasoning behind the need for clustering software systems. Section 3 deals with the related work. Section 4 describes the three approaches used to define an ecosystem that we use throughout this paper. Section 5 illustrates the datasets used. Section 6 provides the results of the statistical tests, that aim to reject, for every OO metric m, the null hypothesis $H_{0,m}$: *the samples are drawn from the same population.*

Section 7 discusses the findings, while Section 8 describes the threats to validity. Finally, Section 9 summarises and concludes the paper.

2. Reasons for clustering

Clustering is deemed to be among the fundamental techniques in knowledge mining and information retrieval [16, 17]. In the context of software engineering, clustering has been used in the past, for reverse engineering and software

[2]`https://www.crossminer.org`

maintenance tasks, with the aim to categorise software artifacts [18, 19]. The concept of *similarity* is a fundamental building block for any clustering technique, as well as a key issue in various contexts, such as detecting cloned code [20, 21, 22, 23], software plagiarism [24], or reducing test suite size in model-based testing [25, 26]. According to *Walenstein et al.* [27], a workable common understanding for software similarity is as follows: *"the degree to which two distinct programs are similar is related to how precisely they are alike."* Nevertheless, a globally exact and shared definition of similarity has not been agreed upon yet: depending on the method used to compare items, various types of similarity may be identified.

Clustering techniques have been exploited in other fields including biology to classify plants or animals according to their properties [28], and geology to classify observed earthquake epicenters and thus to identify dangerous zones [29]. A clustering algorithm attempts to distribute objects into groups of similar objects so as the similarity between one pair of objects in a cluster is higher than that between one of the objects to any objects in a different cluster [30, 31]. In recent years, several clustering methods have been developed to solve a wide range of issues [32]. Among others, there are hierarchical and partitional clustering algorithms [31]. The former use a criterion function to identify partitions while the latter try to group similar partitions. Among partitioning-based algorithms, there are K-Means, K-Medoids, CLARA, CLARANS [33, 34, 35]. Among hierarchical-based algorithms, there are BIRCH [36], CURE [37], ROCK [38], Chameleon [39], to name a few.

Several existing clustering techniques share the property that they can be applied when it is possible to specify a *proximity (or distance) measure* that allows one to assess if elements to be clustered are mutually similar or dissimilar. The basic idea is that the *similarity level of two elements is inversely proportional to their distance.* The definition of the proximity measure is a key issue in almost all clustering techniques and it depends on many factors including the considered application domain, available data, and goals. Once the proximity measure has been defined, it is possible to produce a proximity matrix for the related objects. Given that there are n objects to be clustered, an $n \times n$ proximity matrix needs to be generated containing all the pairwise similarities or dissimilarities between the considered objects.

Recommender systems rely heavily on similarity metrics to suggest suitable and meaningful items for a given item [6, 40, 41, 42]. For example, for third-party library recommendation, it is important to find similar projects to a given project, and mine libraries from the most similar projects [43]. Similarities are used as a base by both content-based and collaborative-filtering recommender systems to choose the most suitable and meaningful items for a given user [6]. In this sense, failing to compute similarities means concurrently adding a decline in the overall performance of these systems.

Nevertheless, measuring similarities between software systems has been considered as a daunting task [10, 44]. Furthermore, considering the heterogeneous nature of artifacts in open source software repositories, similarity computation becomes more complicated as many artifacts and several cross relationships prevail. As a result, similarity computation among software and projects has attracted considerable interest from many research groups. In recent years, several approaches have been proposed to solve the problem of software similarity computation. Many of them deal with similarity for software systems, others are designed for computing similarities among open source software projects. Such approaches are domain specific and they can be classified according to the set of mined features. In particular, there are two main types of software similarity computation techniques as follows [44]. The first is called *low-level similarity* and it is calculated by considering low-level data, e.g., source code, byte code, function calls, API reference, etc. Meanwhile, *high-level similarity* is based on the metadata of the analysed projects e.g., similarities in readme files, textual descriptions, star events, to name a few.

3. Related Work

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 [45], 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.

So far, several tools that capture the topics of software systems have been proposed. Among others, CLAN [46], CrossSim [11], MUDAblue [47] and RepoPal [48] 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.

Wermelinger and Yu [49] posit that presenting two datasets from the same domain 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.* [50] have also highlighted the need to investigate in particular systems belonging to different domains.

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 [51, 52]. In view of this and to reduce the effort required in manual categorisation of software projects, Tian *et al.* [53] proposed a new technique based on text mining to categorise software projects irrespective of the programming language used in their development.

Callau *et al.* [54] 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 that rely heavily on 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 serialisation and de-serialisation of objects and (iii) low-level system *support code* that uses object field reads and writes to implement copy operations, saving the state of the system to disk, and converting numbers and strings from objects to compact bit representation.

In a different study [55], software coupling metrics were studied based on software categories to identify any impact of software categories on coupling metrics (CBO, DAC[3], ATFD[4] and AC[5]). 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.* [56] 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. In [54], authors made a study on the usage of dynamic programming

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

features in terms of the significant energy and memory usage of software in the Databases category wherein database-based software made heavy usage of serialisation and de-serialisation of objects.

In a related study, the focus is on energy management in Android applications [57] 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 [58, 59, 60] 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.* [2] 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 categorised 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.

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

4. Techniques of clustering

In this section we present the three approaches that we used to cluster the systems in our samples: the one illustrated in Section 4.1 clusters projects based on how they are linked to external libraries and components.

The technique described in Section 4.2 is based on 6 pre-determined categories, and the subjective attribution of each project to a cluster. The approach is based on the work proposed by the authors of [12, 13].

The last technique, illustrated in Section 4.3, uses the LDA algorithm to first extract the topics of the software systems, and then it assigns each project to one cluster.

We provide the replication packages, and the datasets with our results, in three collections available online[6].

4.1. Pairwise similarity via CrossSim

Linked Data is a representation method that allows for the interlinking and semantic querying of data [61]. The core of Linked Data is an RDF[7] graph that is made up of several nodes and oriented links to represent the semantic relationships among various artifacts. Thanks to this feature, the representation paves the way for various computations. One of the main applications of RDF is similarity computation for supporting recommender systems [41]. We designed and implemented CrossSim [11] (Cross Project Relationships for Computing Open Source Software Similarity), a tool for computing similarity among OSS projects by considering the analogy of typical applications of RDF graphs and the problem of detecting the similarity of open source projects. CrossSim exploits graphs for representing different types of relationships in the OSS ecosystem. In particular, with the adoption of the graph representation, CrossSim transforms the relationships among non-human artifacts, e.g., API utilisations, source code, interactions, and humans (e.g., developers) into a mathematically computable format, i.e., one that facilitates various types of computation techniques.

[6]CrossSim collection: `https://figshare.com/s/0cd6925bf1601e2f0b74`,
Categorised collection: `https://figshare.com/s/bd506aff8bff2b6ce29f`,
LDA-based collection: `https://figshare.com/s/93f5d997484705a937de`

[7]`https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`

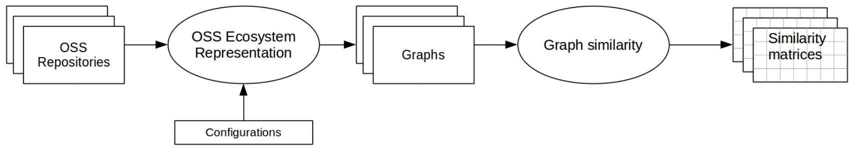


Figure 2: Overview of the CrossSim approach.

The architecture of CrossSim is depicted in Figure 2: the rectangles represent artifacts, whereas the ovals represent activities that are automatically performed by the developed CrossSim tooling. In particular, the approach imports project data from existing OSS repositories and represents them in a graph-based representation by means of the *OSS Ecosystem Representation* module. Depending on the considered repository (and thus to the information that is available for each project) the graph structure to be generated has to be properly configured. For instance, in case of GitHub, specific configurations have to be specified in order to enable the representation in the target graphs of the stars assigned to each project. Such a configuration is repository specific, e.g., SourceForge does not provide the star based system available in GitHub.

The *Graph similarity* module implements the similarity algorithm that applied on the source graph-based representation of the input ecosystems generates matrices representing the similarity value for each pair of the input projects.

To demonstrate the utilisation of graphs in an OSS ecosystem, we consider an excerpt of the dependencies for a pair of OSS projects, namely `project#1` and `project#2` in Figure 3. Using dependency information extracted from source code and the corresponding metadata, this graph can be properly built to represent the two projects as a whole. In this figure, `project#1` contains code file `HttpSocket.java` and `project#2` contains `FtpSocket.java` with the corresponding edges being marked with the semantic predicate `hasSourceCode`. Both source code files implement `interface#1` being marked by the semantic predicate `implements`. `Project#1` and `project#2` are also connected via other semantic paths, such as API `isUsedBy`.

Based on the graph structure, one can exploit nodes, links and the mutual relationships to compute similarity using existing graph similarity algorithms. To the best of our knowledge, there exist several metrics for computing similarity in graphs [62, 40, 41]. Considering Figure 3, we can compute the similarity between `project#1` and `project#2` with regards to the semantic paths between them, e.g., the two-hop path using `hasSourceCode` and `implements` (Figure 4), or the one-hop path using API `isUsedBy`. For example, concerning `isUsedBy`, the two projects are considered to be similar since with the predicate both projects originate from `API#1`. The hypothesis is based on the fact that the projects are aiming at creating common functionalities by using common libraries [10, 63].

CrossSim adopts SimRank [64] as the mechanism for computing similarities among OSS graph nodes. SimRank has been developed to calculate similarities based on mutual relationships between graph nodes. Considering two nodes, the more similar nodes point to them, the more similar the two nodes are. For future work, other similarity algorithms can also be flexibly integrated into CrossSim, as long as they are designed for graphs.

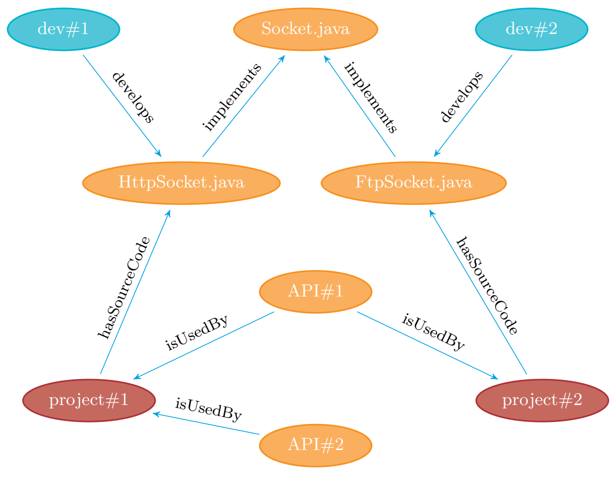


Figure 3: Sample graph-based representation of OSS ecosystems.

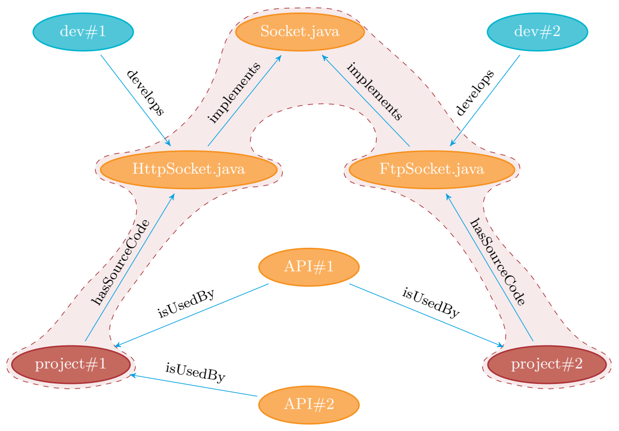


Figure 4: Similarity between OSS projects with respect to source implementation.

4.2. Clustering based on projects descriptions

As the second approach to clustering, we consider the work proposed in [12, 13]. The authors state that "*we manually classified the domain of each system in our dataset. Initially, the first author of this paper inspected the description of the top-200 repositories to provide a first list of application domains, distributed over six domain types.*"

As the clustering unit, this approach uses the collection of software systems sharing the same application domain. The relevance of application domains as a driving factor for specific development approaches has been long recognised: in Glass and Vessey's seminal paper it is mentioned that "*it has become clear that application-independent techniques and tools must be supplemented with an application-specific approach*" [65].

The projects here are the Java subset of the data dump available at `https://zenodo.org/record/804474#.XDi1S9_njCK`. The dataset contains 5,000 project names hosted on GitHub, and the authors pre-determined the following six categories:

- Application Software (AS)
- Documentation (D)

- Non Web Libraries And Frameworks (NW)
- Software Tools (ST)
- System Software (SS)
- Web Libraries And Frameworks (WL)

In the original paper, each project was assigned, by one of the authors, to one of those six categories, based on its characteristics. The assignment was later validated by a second author. Albeit the process was triaged, it is difficult to exactly reproduce the original classification.

The work done in [12, 13] is a unique case of third-party assignment of software projects to categories. Therefore, we make use of the sample, and the classification, and treat it as a 'manual' clustering technique. On the other hand, we cannot fully replicate their clustering approach in the other two samples: the knowledge to assign the projects to the six categories is mostly informal, and not fully reproducible.

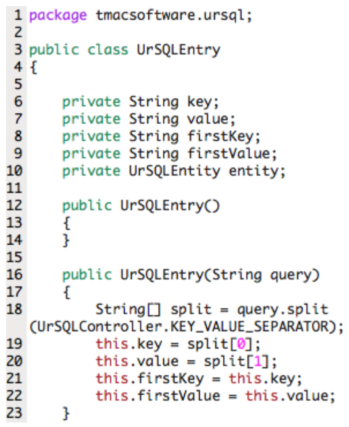
4.3. LDA-informed clustering

This third clustering technique is similar to what presented in Section 4.2, but it adds an automated step to extract the topics contained in a software system. This should be helpful for the reproducibility of the approach.

For this purpose, we extracted the lexical content (e.g., its *corpus*) of each Java class in two ways: *(i)* by considering their class names; and *(ii)* by parsing their code and considering the variable names, comments and keywords.

The code of each 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 [66]. The corpus includes the class name, variable and method names and comments for each class.

Pre-processing of the system corpus is needed to eliminate common keywords, stop words, split and to stem class names [67]. Also, we do not consider



```
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;
    }
```

Figure 5: UrSQLEntry.java Source Code Snippet.

as a term any of the Java-specific keywords (e.g., *if*, *then*, *switch*, etc.)[8]. 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*). Second, each term goes through a phase of *stemming* and *lemmatisation*, to extract its root.

As an example of this lexical extraction, for the lines of code shown in Figure 5 (the UrSQLEntry.java class from the UrSQL project), we derive the following corpus: {*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*}.

For each system, all the Java classes were reduced to a corpus of terms. All these terms were then considered to create a model implementing the Latent Dirichlet Allocation (LDA) algorithm. Python is the programming language used to program the models, and the *gensim* NLP package[9] helped by the machine learning side.

[8]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.

[9]`https://radimrehurek.com/gensim/`

To extract the main topics emerging from the corpus of a software project, we utilize LDA which is a topic-modeling technique [68]. Each document is featured using a Natural Language Processing approach termed the Term-frequency-inverse document frequency (TF-IDF). In NLP, TF-IDF [69] is used to measure the weight of a term within documents (in our case, the source code of a class). With TF-IDF, words are assigned weights, as the product of term frequency and inverse document frequency. We use TF-IDF as a pre-processing step to LDA: the result is a representation of the source code contained in the Java classes, where the same terms can appear multiple times (see Figure 5).

As an example, for the *okhttp* project[10], 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**”

The topics extracted from the projects were finally assigned to a category by two of the authors of this paper: in case of disagreement, a discussion was held to reconcile the views. As the list of categories, we adopted in fact what has been historically used by the `SourceForge.net` repository to

[10]`https://github.com/square/okhttp`

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*}.

5. Datasets used for the Empirical Analysis

This section presents the datasets that have been used for performing the empirical analysis. In particular, the dataset used by CrossSim is presented in Section 5.1. The categorised dataset as presented in [12] is summarized in Section 5.2. The dataset used to apply the LDA-based technique is presented in Section 5.3.

5.1. CrossSim dataset

The overall dataset gathered by the CROSSMINER project to evaluate the CrossSim approach consists of 580 GitHub Java projects. Such a dataset was collected from GitHub by considering the following requirements: *(i)* being GitHub Java projects; *(ii)* providing the specification of their dependencies by means of pom.xml or .gradle files *(iii)* having at least 9 external dependencies; *(iv)* having the README.md file available; *(v)* possessing at least 20 stars[11]. The collected dataset and the CrossSim tool are available online for public usage [70].

For the purpose of our analysis, 6 pairs of software systems were extracted from the CROSSMINER dataset. The criteria of selection were:

- maximum similarity between projects within the pairs;
- minimum similarity between projects of different pairs.

The outcome of the extraction satisfying such criteria is shown in Table 1.

[11]The motivations behind such requirements are explained in [11]
[12]Available at `git://github.com/psaravan/JamsMusicPlayer.git`

Cluster	Projects	Similarity (between pairs)

AB	A = JamsMusicPlayer[12]
	B = ACEMusicPlayer[13]
		1.95E-04

CD	C = sparql-plugin[14]
	D = neo4j-sparql-extension[15]
		0.0027

EF	E = jpmml-model[16]
	F = visitante[17]
		7.32E-04

GH	G = c2d-engine[18]
	H = LeanEngine-Server[19]
		5.47E-04

IJ	I = RestOpenGov[20]
	J = elasticsearch-analysis-lemmagen[21]
		0.00112

KL	K = jcabi-email[22]
	L = jcabi-jdbc[23]
		0.0048

Table 1: Clusters of projects identified by CrossSim.

5.2. Categorised dataset (from [13])

From the original, overall sample of 5,000 projects, we extracted all the Java projects (520 projects overall), while maintaining the information about their assigned category. Considering the Java subset, and using the categories provided, we have the distribution of projects within the ecosystems as shown in Table 2. We extracted the metrics for each project, then clustered those metrics based on the six domains identified above. A statistical test was run between each pair of domains, and per OO metric, to determine if the metrics come from the same population.

[13]Available at `git://github.com/C-Aniruddh/ACEMusicPlayer.git`
[14]Available at `git://github.com/neo4j-contrib/sparql-plugin.git`
[15]Available at `git://github.com/niclashoyer/neo4j-sparql-extension.git`
[16]Available at `git://github.com/jpmml/jpmml-model.git`
[17]Available at `git://github.com/pranab/visitante.git`
[18]Available at `git://github.com/lycying/c2d-engine.git`
[19]Available at `git://github.com/PeterKnego/LeanEngine-Server.git`
[20]Available at `git://github.com/RestOpenGov/RestOpenGov.git`
[21]Available at `git://github.com/vhyza/elasticsearch-analysis-lemmagen.git`
[22]Available at `git://github.com/jcabi/jcabi-email.git`
[23]Available at `git://github.com/jcabi/jcabi-jdbc.git`

Category	**Projects**
Application Software	30
Documentation	48
Non Web Libraries And Frameworks	342
Software Tools	49
System Software	26
Web Libraries And Frameworks	25

Table 2: Number of Java projects in the categories extracted in [13].

5.3. LDA-based dataset

Leveraging GitHub, we collected the project IDs of the 100 most successful Java projects hosted on GitHub as case studies[24]. 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 [71], and potentially have a good intake of new developers [72]. 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. The boxplots in Figure 6 show the basic characteristics of the sample analysed: the distributions of duration of the projects (in days), the number of distinct developers (as authors) and the total number of commits are plotted. In terms of most likely value of each distribution, we observed that the median in the project’s duration is 2002 days; the median number of developers is 87; while the median in the number of commits is 1204.

Figure 7 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.

[24]The list of projects is available at `https://figshare.com/s/c627af8e9e496a9025c4`

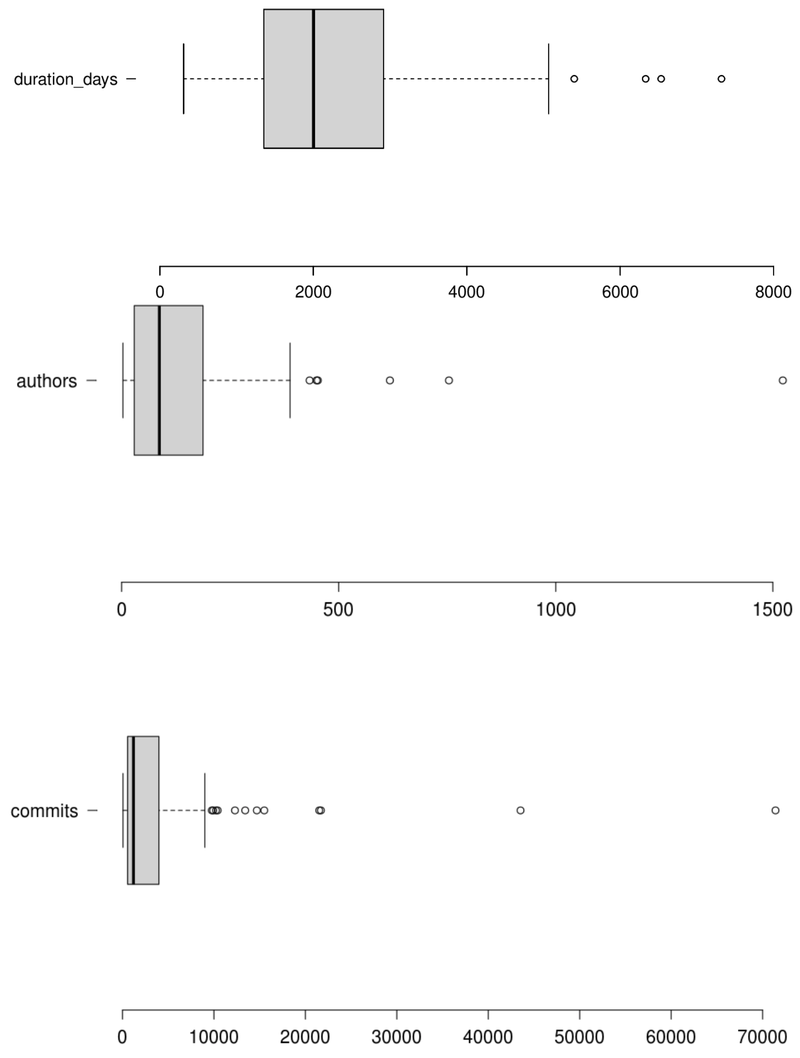


Figure 6: Distribution of duration (in days), number of developers (as authors) and number of commits in the sample analysed with the LDA approach

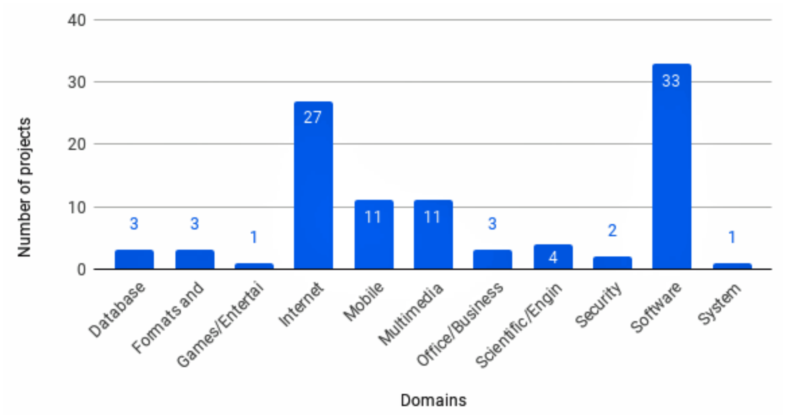


Figure 7: Domains extracted with the LDA approach.

A few of the basic domains, as used by SourceForge, 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.

6. Results

In this section we report the results of the analysis using the three clustering techniques: the one implemented by the CrossSim algorithm (Section 6.1); the mostly manual one (Section 6.2); and the semi-automated one (Section 6.3).

For each dataset and clustering technique, we adopted the Kolgomorov-Smirnov (KS) [73] test to detect whether the distribution of software metrics in the compared domains are from the same population. The appropriate Bonferroni correction [74] was applied, due to the multiple tests being carried out at the same time.

6.1. CrossSim dataset – Results

In order to evaluate the rejection of (or the inability to reject) the null hypothesis, we performed the KS statistical tests for each of the 6 project pairs (AB, CD, EF, GH, IJ and KL) described in Section 5.1. We assigned a standard threshold value $\alpha = 0.05$ for the sensitivity of the test; the Bonferroni correction resulted in $\alpha_B = 0.0034137$.

In Table 3 we summarise the tests by reporting the p-value of each KS test. We must reject the null hypothesis (i.e., "the two samples are drawn from the same distribution") if the p-value of the KS test is lower than the corrected α_B. This is done for each metric, and for each pair of ecosystems. As an example, the KS test for the IFANIN metric, and the pair of ecosystems AB and CD produces a p-value = 1: we reject the null hypothesis that the values of the IFANIN metric come from the same population, when considering the AB and CD ecosystems.

The results of each KS test (in the form of p-values) are summarised in Table 3: as an example, we rejected all the null hypotheses for the EF and IJ clusters: none of the OO attributes can be considered to be sampled from different populations. The most dissimilar pair of clusters appears to be the AB and GH pairs: for most of the structural OO metrics, we could reject the relative null hypothesis, therefore the IFANIN, NIM, NIV, WMC, RFC and DIT values can be considered as drawn from different populations.

In general however, for most of the metrics, and for most of the pairs of ecosystems, the null hypothesis cannot be rejected. When we grouped projects based on the type of dependencies that they have, it is in general difficult to conclude that the clusters are structurally different from each other.

This result is not surprising: the projects paired by CrossSim are associated by what type of external libraries they use, while they could be implementing very different features. They could be based on very different domains, but still use a similar set of external libraries.

	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT	LCOM
ABvCD	1	0.000	0.882	0.000	0.083	2.95E-06	2.51E-11	0.043	0.025
ABvEF	0.874	3.02E-07	0.000	0.006	3.51E-05	1.72E-14	0	3.02E-09	0.894
ABvGH	1.11E-10	0.009	0.051	7.07E-06	3.16E-05	4.41E-07	0	0	0.009
ABvIJ	0.077	0.146	0.339	0.192	0.640	0.533	0.371	1.19E-08	0.497
ABvKL	1.33E-15	1.17E-10	0.497	0.156	0.001	0.116	0.122	0	0.067
CDvEF	1.000	0.030	0.999	0.001	0.014	0.001	0.000	0.285	0.030
CDvGH	0.197	1.95E-06	1	0.001	0.031	1.46E-05	0.013	0.000	0.002
CDvIJ	0.558	0.004	0.967	0.005	0.672	0.000	6.43E-06	0.206	0.789
CDvKL	2.13E-06	0.128	1	8.69E-05	0.126	1.14E-05	9.31E-10	9.31E-10	0.001
EFvGH	0.001	6.13E-05	0.272	0.151	0.995	0.012	0.001	0.000	0.267
EFvIJ	0.337	0.484	0.999	0.363	0.155	0.190	0.014	0.040	0.333
EFvKL	2.17E-11	0.0046722	0.979	0.567	0.002	0.185	0.000	0	0.154
GHvIJ	0.926	0.753	0.984	0.475	0.294	0.539	0.000	0.000	0.095
GHvKL	3.17E-06	5.60E-08	0.999	0.084	0.006	0.093	2.14E-10	0	0.516
IJvKL	0.001	0.003	0.885	0.128	0.303	0.647	0.367	5.67E-05	0.033

Table 3: Results of the pair-wise statistical tests of the OO metrics analysed: AB, CD, EF, GH, IJ and KL refer to the pairs of projects.

6.2. Categorised dataset (from [13]) – Results

The same approach to statistical testing was also applied for the second dataset. Table 4 shows the results of the statistical analysis: as above, each cell contains the p-value of the Kolgomorov-Smirnov (KS) test between two subsets of the dataset. For example, the ASvD row contains the p-values of the KS test between the projects in the *Application Software* domain, and the projects in the *Documentation* domain. The null hypothesis (‘*the two samples come from the same distribution*’) can be rejected if the p-value is larger than a threshold α. For our tests, we selected $\alpha = 0.05$: since multiple tests were run at the same time, a Bonferroni correction was needed, obtaining a corrected threshold of $\alpha_B = 0.0033$ (i.e., 0.05/15 where 15 is the number of multiple tests performed).

We have highlighted in Table 4 the specific OO attribute where we reject the null hypothesis. As it can be seen, the NOC, DIT and LCOM attributes can be considered from the *same* distribution, but only considering the subsets of projects in the *Application Software domain* and those in the *System Software domain*. Similarly, for the NOC attribute, and considering the *Non Web Libraries And Frameworks* and *Web Libraries And Frameworks* clusters.

Considering all the other tests, the resulting p-values allow for a general rejection of the null hypothesis: we can assume that the values of the attributes come from different populations.

This is an interesting result: the clusters generated by a manual inspection of the projects' characteristics result in pools of attributes that are structurally different from each other. As an example, the *Software Tools* (ST) category rejects all the null hypotheses, for all the selected OO attributes, when compared to any other category. This places the Software Tools cluster as a standalone category, with specific (and unique) characteristics.

6.3. LDA-based dataset – Results

Similarly to what has been done in Section 6.2, we clustered the projects in the categories proposed in SourceForge. The OO data was then extracted, per cluster, and one KS test executed per pair of clusters, and for all the metrics.

Table 5 summarises the tests by reporting the p-value of each test. High-

	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT	LCOM
ASvD	0	0	0	0	0	0	0	0	4.76E-06
ASvNW	0	0	0	3.23E-12	0	2.22E-16	0	0	0
ASvSS	1.37E-06	0	0.0247	0	1.14E-10	0	0	0.476	0.204
ASvST	0	0	0	0	0	0	0	0	0
ASvWL	0	0	0	0	0	0	0	0	0
DvNW	0	0	1.79E-05	0	2.80E-14	0	0	0	0
DvSS	0	0	0	0	0	0	0	0	5.83E-10
DvST	0	0	4.66E-06	0	0	0	0	0	0
DvWL	0	0	4.90E-06	0	0	0	0	0	0
NWvSS	0	0	0	0	0	0	0	0	0
NWvST	0	0	0	0	6.14E-07	0	0	0	0
NWvWL	0	0	0.344	0	0	0	0	0	0
SSvWL	0	0	0	0	0	0	0	0	0
STvSS	0	0	0	0	0	0	0	0	0
STvWL	0	0	0	0	0	0	0	0	0

Table 4: Results of the pair-wise statistical tests of the OO metrics analysed: AS refers to *Application Software*, D to *Documentation*, NW to *Non Web Libraries And Frameworks*, ST to *Software Tools*, SS to *System Software*, and WL to *Web Libraries And Frameworks*.

	IFANIN	CBO	NOC	NIM	NIV	WMC	RFC	DIT	LCOM
IvSD	0	0	0	0	0	0	0	0	0
IvMob	0	0	0	0	0	0	0	0	0
IvMM	0	0	0	0	0	0	0	0	0
SDvMob	3.29E-09	0	0.868	9.24E-4	1.9E-4	3.27E-05	0	5.55E-16	7.12E-4
MMvMob	6.68E-06	0	0.999	1.36E-11	0	1.59E-09	2.62E-13	0.57	4.08E-10
MMvSD	0.015	8.42E-09	4.56E-05	0	0	0	0	0	0

Table 5: Results of the pair-wise statistical tests of the OO metrics analysed: I refers to *Internet*, SD to *Software Development*, Mob to *Mobile* and MM to *Multimedia*

lighted are the metrics for which the p-value does not allow us to reject the null hypothesis (‘*the two samples come from the same distribution*’) based on $\alpha = 0.05$ and a Bonferroni correction of 0.0083333.

As shown in Table 5, and similarly to the results in Table 4, the NOC attribute does not always allow for a clear differentiation between clusters. In two cases, we could not reject the null hypothesis: the values of the NOC attribute can be considered as drawn from the same population, at least in the cases of the SD and Mob comparison, and the MM and Mob comparison. For the other comparisons, we can reject all the other null hypotheses: we can conclude that OO attributes are extracted from different populations, when considering the clusters formed by an LDA-informed approach.

It is worth noting that for both Categorised dataset, and the LDA-informed dataset, the DIT and NOC attributes are less sensitive to identify differences between samples. This is partly due to the fact that these attributes reflect a similar characteristic of the structure of Java software (i.e., inheritance). Shared design principles suggest that the DIT attribute should be kept low. This is true for two main reasons:

- The deeper a class is in the hierarchy, the greater the total number of methods it is likely to inherit [75], making its behaviour less predictable [76].
- Khalid *et al.* state that “DIT is directly proportional to complexity” (i.e., an increased DIT will lead to higher maintenance efforts) [77].

On the other hand, the NOC attribute should also be kept low: a large CBO increases the complexity of the system, and it adversely affects other quality factors, such as maintainability, testability and reusability [78].

7. Discussion

In this section we add further insights as part of our discussion: in light of our evidence, we studied past research works, specifically focused on the OO metrics that we used, and we reflected on the importance of clustering.

In a prior study [79], we collected empirical results showing that projects from the same domain exhibit common structural properties in terms of the C&K metrics. In this work, we have expanded on those results: the results that we have gathered above indicate that the projects clustered around the domains (i.e., 2 of the 3 clustering techniques presented above) show indeed a difference in the structural metrics. For interested stakeholders, this can imply that the structure of a software system (and its soundness) depends on domain-based factors common to projects in the same domain. For example, projects from different domains making use of exception handling differently [60, 58].

The discussion that we present here takes into consideration the correlations between the OO metrics that we utilised above. The work reported by [80] has already shown some correlation between pairs of metrics from the C&K suite, for example, CBO and RFC, and RFC and LCOM. We want to know whether these correlations change sensibly, when considering specific clusters. If indeed there were 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 7.

In the analysis below, we report on the correlation study between pairs of OO metrics, and 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 [81] ($[0-0.1]$ 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 6 shows the correlations among pairs of OO metrics, when considering the projects in the domain-driven clusters identified by the LDA technique. 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), thus being 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.

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

The correlation among OO metrics can be extremely sensitive to application domains

Thus, according to such a finding, evaluating the quality of a sytsem becomes also dependent on what type of domain a project belongs to. For instance, the metrics one should consider to analyse gaming software should be different from those used to assess the quality of security software. For instance, the latter is mainly characterized by incremental contributions aiming at fixing already existing functionalities and making them more stable and secure. Such a particular attention to stability and security aspects can be less peculiar for gaming

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 6: 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.

software and consequently OO metrics, e.g., LOCs are less appropriate for assessing the quality of security software or in general of mission critical software systems.

7.1. Clusters and their maturity

The maturity of the considered application domain (as cluster) is another factor that can interfere with the analysis performed by means of OO metrics. Emerging domains are characterized by a plethora of new applications even though while the domain becomes more “stable” and mature, only those applications that managed to create a community (and thus that are actually used and maintained) eventually survive. Thus, the population of the resulting application domain is characterized by OO metrics that might be different from those of the initial population:

> Emerging new application domains might show a larger variability in their structural metrics, in comparison to established domains.

Finally, some domains consist of reusable libraries and frameworks instead of ready-to-use applications. For instance, in the domain of automation testing, we can have JUnit, Selenium, Jasmin, etc. Thus, the domain consists of a population that overall shares only the same goal, but it is likely to be characterized by an insignificant correlation in terms of OO metrics.

This research is part of a wider context, and it requires further multi-disciplinary investigation: the similarity of software systems should follow the same approach of compiling a biological taxonomy, where systems (or parts of) are given a place (or rank) in a hierarchy. Lower levels share (OO-related) attributes with higher levels in the same branch, whereas different branches of the taxonomy show the highest degree of dissimilarity. Such a taxonomy could have a massive impact in how practitioners and researchers work and develop software systems: ad-hoc techniques and tools would be needed and tailored to domain-specific constraints. This has already started to emerge for the software engineering techniques around gaming development [82]: we envisage that a similar approach would be needed when new domains become established, and their boundaries are clearer.

8. Threats to validity

8.1. External validity

Threats to external validity refer to the extent to which the results of our study can be generalized. The projects we have analysed come from publicly available repositories and the used clustering techniques also come from already validated works. However, we cannot claim that the resulting conclusion concerning the considered null hypotheses is generalisable, even though the performed experiments provide us with an acceptable confidence about the general validity of the reached conclusions.

8.2. Internal validity

Threats to internal validity concern any confounding factor that could influence our findings. We attempted to avoid any bias in the building of the ecosystems. To this end, we applied CrossSim and the LDA-based approach by means of the corresponding supporting tools without any manual intervention. However, the tools we have used and implemented could be defective. To counter this threat, we have run several manual assessments and counter-checks. Concerning the approach in [13] we used the categories as they are presented in the original work without any change.

8.3. Construct validity

It pertains to any factor that can compromise the validity of inferences that observations or measurement tools actually represent or measure the construct being investigated. A potential threat to construct validity is related to the size of the analysed data. However, the datasets that have been considered for the experiments come from the original works proposing CrossSim, the LDA-based approach, and the categorization in [13]. Thus, we built on those datasets that were considered to be big enough for experimenting and validating such approaches.

The second threat to construct validity is based on the fact that the three approaches proposed use three different samples of data. We could not use the same sample because the approach described in Section 4.2 above is not fully reproducible. On the other hand, the LDA-based approach was replicated for the CrossSim sample, and we concluded that the clusters grouped by CrossSim represent the type of external libraries they use, while they could be implementing very different features. They could be based on very different domains, but still use a similar set of external libraries.

9. Conclusion

This paper presented three techniques to cluster software projects, and evaluated how the clusters differ from each other, when comparing their internal characteristics. The null hypothesis that we attempted to reject is that all the clusters come from the same population. As metrics of assessment, we adopted a suite of well known OO attributes.

As the first technique we adopted the CrossSim algorithm [11], that draws a similarity between projects based on their usage of external libraries. Interestingly enough, with this technique we could not reject the null hypothesis: projects might still be *internally* similar even if the CrossSim algorithms detect a large distance between them.

The second technique was based on a subjective classification, based on the description of a software project. With this technique we found a strong foundation for rejecting the null hypothesis, although the steps of the clustering process are not easily reproducible.

The third technique was also based on categories, but it automatically extracted topics from a software systems, before assigning them to categories. Also in this case, we could reject the null hypothesis for most of the OO metrics.

The implications of these findings can be profound: software projects can manifest different characteristics, based on the domain or cluster that they belong to. Empirical findings might need readjustment, and tailoring to one or another cluster or domain, to fit with other findings in the same cluster.

Acknowledgements

The research described in this paper has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant 732223.

References

[1] D. Yang, P. Martins, V. Saini, C. Lopes, Stack overflow in github: any snippets there?, in: Proceedings of the 14th International Conference on Mining Software Repositories, IEEE Press, 2017, pp. 280–290.

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

[3] R. Kikas, M. Dumas, D. Pfahl, Using dynamic and contextual features to predict issue lifetime in github projects, in: Proceedings of the 13th International Conference on Mining Software Repositories, ACM, 2016, pp. 291–302.

[4] M. Linares-Vásquez, A. Holtzhauer, C. Bernal-Cárdenas, D. Poshyvanyk, Revisiting android reuse studies in the context of code obfuscation and library usages, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 242–251.

[5] G. Burlet, A. Hindle, An empirical study of end-user programmers in the computer music community, in: Proceedings of the 12th Working Conference on Mining Software Repositories, IEEE Press, 2015, pp. 292–302.

[6] J. B. Schafer, D. Frankowski, J. Herlocker, S. Sen, The adaptive web, Springer-Verlag, Berlin, Heidelberg, 2007, Ch. Collaborative Filtering Recommender Systems, pp. 291–324.
URL http://dl.acm.org/citation.cfm?id=1768197.1768208

[7] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, J. Sun, Detecting similar repositories on github, 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER) 00 (2017) 13–23. doi:doi.ieeecomputersociety.org/10.1109/SANER.2017.7884605.

[8] D. Spinellis, C. Szyperski, How is open source affecting software development?, IEEE Software 21 (1) (2004) 28–33. doi:10.1109/MS.2004.1259204.

[9] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, L. Ochoa, T. Degueule, M. Di Penta, FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns, in: Proceedings of the 41st International Conference on Software Engineering, ICSE '19, IEEE Press, Piscataway, NJ, USA, 2019, pp. 1050–1060. doi:10.1109/ICSE.2019.00109.
URL https://doi.org/10.1109/ICSE.2019.00109

[10] C. McMillan, M. Grechanik, D. Poshyvanyk, Detecting similar software applications, in: Proceedings of the 34th International Conference on Software Engineering, ICSE '12, IEEE Press, Piscataway, NJ, USA, 2012, pp. 364–374.
URL `http://dl.acm.org/citation.cfm?id=2337223.2337267`

[11] P. T. Nguyen, J. Di Rocco, R. Rubei, D. Di Ruscio, CrossSim: Exploiting Mutual Relationships to Detect Similar OSS Projects, in: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2018, pp. 388–395. `doi:10.1109/SEAA.2018.00069`.

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

[13] H. Borges, M. T. Valente, Whats in a github star? understanding repository starring practices in a social coding platform, Journal of Systems and Software 146 (2018) 112–129.

[14] S. R. Chidamber, C. F. Kemerer, Towards a metrics suite for object oriented design, SIGPLAN Not. 26 (11) (1991) 197–211. `doi:10.1145/118014.117970`.
URL `http://doi.acm.org/10.1145/118014.117970`

[15] M. Lorenz, J. Kidd, Object-oriented software metrics, Vol. 131, Prentice Hall Englewood Cliffs, 1994.

[16] C. D. Manning, P. Raghavan, H. Schütze, Introduction to Information Retrieval, Cambridge University Press, New York, NY, USA, 2008.

[17] M. Steinbach, G. Karypis, V. Kumar, A comparison of document clustering techniques, in: 6th ACM SIGKDD, World Text Mining Conference, 2000.

[18] A. Kuhn, S. Ducasse, T. Girba, Enriching reverse engineering with semantic clustering, in: 12th Working Conference on Reverse Engineering (WCRE'05), IEEE, 2005, pp. 10–pp.

[19] O. Maqbool, H. Babri, Hierarchical clustering for software architecture recovery, IEEE Transactions on Software Engineering 33 (11) (2007) 759–780.

[20] W. S. Evans, C. W. Fraser, F. Ma, Clone detection via structural abstraction, Software Quality Journal 17 (4) (2009) 309–330. `doi:10.1007/s11219-009-9074-y`.
URL `https://doi.org/10.1007/s11219-009-9074-y`

[21] C. Ragkhitwetsagul, J. Krinke, D. Clark, A comparison of code similarity analysers, Empirical Software Engineering 23 (4) (2018) 2464–2519. doi:10.1007/s10664-017-9564-7.
URL https://doi.org/10.1007/s10664-017-9564-7

[22] C. Ragkhitwetsagul, J. Krinke, B. Marnette, A picture is worth a thousand words: Code clone detection based on image similarity, in: 2018 IEEE 12th International Workshop on Software Clones (IWSC), 2018, pp. 44–50. doi:10.1109/IWSC.2018.8327318.

[23] R. Tiarks, R. Koschke, R. Falke, An extended assessment of type-3 clones as detected by state-of-the-art tools, Software Quality Journal 19 (2) (2011) 295–331. doi:10.1007/s11219-010-9115-6.
URL https://doi.org/10.1007/s11219-010-9115-6

[24] A. M. Leitão, Detection of redundant code using r2d2, Software Quality Journal 12 (4) (2004) 361–382. doi:10.1023/B:SQJO.0000039793.31052.72.
URL https://doi.org/10.1023/B:SQJO.0000039793.31052.72

[25] A. E. V. B. Coutinho, E. G. Cartaxo, P. D. de Lima Machado, Analysis of distance functions for similarity-based test suite reduction in the context of model-based testing, Software Quality Journal 24 (2014) 407–445.

[26] S. U. R. Khan, S. P. Lee, R. W. Ahmad, A. Akhunzada, V. Chang, A survey on test suite reduction frameworks and tools, Int. J. Inf. Manag. 36 (6) (2016) 963–975. doi:10.1016/j.ijinfomgt.2016.05.025.
URL https://doi.org/10.1016/j.ijinfomgt.2016.05.025

[27] A. Walenstein, M. El-Ramly, J. R. Cordy, W. S. Evans, K. Mahdavi, M. Pizka, G. Ramalingam, J. W. von Gudenberg, Similarity in programs, in: Duplication, Redundancy, and Similarity in Software, 23.07. - 26.07.2006, 2006.
URL http://drops.dagstuhl.de/opus/volltexte/2007/968

[28] U. von Luxburg, R. C. Williamson, I. Guyon, Clustering: Science or art?, in: I. Guyon, G. Dror, V. Lemaire, G. Taylor, D. Silver (Eds.), Proceedings of ICML Workshop on Unsupervised and Transfer Learning, Vol. 27 of Proceedings of Machine Learning Research, PMLR, Bellevue, Washington, USA, 2012, pp. 65–79.
URL http://proceedings.mlr.press/v27/luxburg12a.html

[29] C. Rodolfo, M. Murru, F. Catalli, G. Falcone, Real time forecasts through an earthquake clustering model constrained by the rate-and-state constitutive law: Comparison with a purely stochastic etas model, Seismological Research Letters 78 (2007) 49–56. doi:10.1785/gssrl.78.1.49.

[30] P. Berkhin, A survey of clustering data mining techniques, in: J. Kogan, C. Nicholas, M. Teboulle (Eds.), Grouping Multidimensional Data, Springer, 2006, pp. 25–71. `doi:10.1007/3-540-28349-8_2`.
URL `http://dx.doi.org/10.1007/3-540-28349-8_2`

[31] A. K. Jain, M. N. Murty, P. J. Flynn, Data clustering: a review, ACM computing surveys (CSUR) 31 (3) (1999) 264–323.

[32] D. Xu, Y. Tian, A comprehensive survey of clustering algorithms, Annals of Data Science 2 (2) (2015) 165–193. `doi:10.1007/s40745-015-0040-1`.
URL `https://doi.org/10.1007/s40745-015-0040-1`

[33] L. Kaufman, P. J. Rousseeuw, Finding groups in data : an introduction to cluster analysis.
URL `http://opac.inria.fr/record=b1087461`

[34] R. T. Ng, J. Han, Clarans: A method for clustering objects for spatial data mining, IEEE Trans. on Knowl. and Data Eng. 14 (5) (2002) 1003–1016. `doi:10.1109/TKDE.2002.1033770`.
URL `http://dx.doi.org/10.1109/TKDE.2002.1033770`

[35] O. Maimon, L. Rokach (Eds.), Clustering Methods, Springer US, Boston, MA, 2005, pp. 321–352. `doi:10.1007/0-387-25465-X_15`.
URL `http://dx.doi.org/10.1007/0-387-25465-X_15`

[36] T. Zhang, R. Ramakrishnan, M. Livny, Birch: A new data clustering algorithm and its applications, Data Min. Knowl. Discov. 1 (2) (1997) 141–182. `doi:10.1023/A:1009783824328`.
URL `https://doi.org/10.1023/A:1009783824328`

[37] S. Guha, R. Rastogi, K. Shim, Cure: An efficient clustering algorithm for large databases, SIGMOD Rec. 27 (2) (1998) 73–84. `doi:10.1145/276305.276312`.
URL `http://doi.acm.org/10.1145/276305.276312`

[38] Rock: A robust clustering algorithm for categorical attributes, in: Proceedings of the 15th International Conference on Data Engineering, ICDE '99, IEEE Computer Society, Washington, DC, USA, 1999, pp. 512–.
URL `http://dl.acm.org/citation.cfm?id=846218.847264`

[39] G. Karypis, E.-H. S. Han, V. Kumar, Chameleon: Hierarchical clustering using dynamic modeling, Computer 32 (8) (1999) 68–75. `doi:10.1109/2.781637`.
URL `http://dx.doi.org/10.1109/2.781637`

[40] P. T. Nguyen, P. Tomeo, T. Di Noia, E. Di Sciascio, Content-based recommendations via dbpedia and freebase: A case study in the music domain, in: Proceedings of the 14th International Conference on The Semantic Web - ISWC 2015 - Volume 9366, Springer-Verlag New York, Inc., New York, NY, USA, 2015, pp. 605–621. doi:10.1007/978-3-319-25007-6_35.
URL http://dx.doi.org/10.1007/978-3-319-25007-6_35

[41] P. T. Nguyen, P. Tomeo, T. Di Noia, E. Di Sciascio, An evaluation of simrank and personalized pagerank to build a recommender system for the web of data, in: Proceedings of the 24th International Conference on World Wide Web, WWW '15 Companion, ACM, New York, NY, USA, 2015, pp. 1477–1482. doi:10.1145/2740908.2742141.
URL http://doi.acm.org/10.1145/2740908.2742141

[42] B. Sarwar, G. Karypis, J. Konstan, J. Riedl, Item-based collaborative filtering recommendation algorithms, in: Proceedings of the 10th International Conference on World Wide Web, WWW '01, ACM, New York, NY, USA, 2001, pp. 285–295. doi:10.1145/371920.372071.
URL http://doi.acm.org/10.1145/371920.372071

[43] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, Mining software repositories to support OSS developers: A recommender systems approach, in: Proceedings of the 9th Italian Information Retrieval Workshop, Rome, Italy, May, 28-30, 2018., 2018.
URL http://ceur-ws.org/Vol-2140/paper9.pdf

[44] N. Chen, S. C. Hoi, S. Li, X. Xiao, Simapp: A framework for detecting similar mobile applications by online kernel learning, in: Proceedings of the Eighth ACM International Conference on Web Search and Data Mining, WSDM '15, ACM, New York, NY, USA, 2015, pp. 305–314. doi:10.1145/2684822.2685305.
URL http://doi.acm.org/10.1145/2684822.2685305

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

[46] C. McMillan, N. Hariri, D. Poshyvanyk, J. Cleland-Huang, B. Mobasher, Recommending source code for use in rapid software prototypes, in: Proceedings of the 34th International Conference on Software Engineering, IEEE Press, 2012, pp. 848–858.

[47] S. Kawaguchi, P. K. Garg, M. Matsushita, K. Inoue, Mudablue: An automatic categorization system for open source repositories, Journal of Systems and Software 79 (7) (2006) 939–953.

[48] Y. Zhang, D. Lo, P. S. Kochhar, X. Xia, Q. Li, J. Sun, Detecting similar repositories on github, in: Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on, IEEE, 2017, pp. 13–23.

[49] M. Wermelinger, Y. Yu, An architectural evolution dataset, in: Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on, IEEE, 2015, pp. 502–505.

[50] D. M. German, M. Di Penta, Y.-G. Gueheneuc, G. Antoniol, 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, 2009, pp. 81–90.

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

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

[53] K. Tian, M. Revelle, D. Poshyvanyk, Using Latent Dirichlet Allocation for automatic categorization of software, in: Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on, IEEE, 2009, pp. 163–166.

[54] O. Callaú, R. Robbes, É. Tanter, D. Röthlisberger, How (and why) developers use the dynamic features of programming languages: the case of smalltalk, Empirical Software Engineering 18 (6) (2013) 1156–1194.

[55] L. B. L. De Souza, M. D. A. Maia, Do software categories impact coupling metrics?, in: Proceedings of the 10th working conference on mining software repositories, IEEE Press, 2013, pp. 217–220.

[56] M. Linares-Vásquez, G. Bavota, C. Bernal-Cárdenas, R. Oliveto, M. Di Penta, D. Poshyvanyk, Mining energy-greedy api usage patterns in android apps: an empirical study, in: Proceedings of the 11th Working Conference on Mining Software Repositories, ACM, 2014, pp. 2–11.

[57] L. Bao, D. Lo, X. Xia, X. Wang, C. Tian, 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, 2016, pp. 37–48.

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

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

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

[61] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, Int. J. Semantic Web Inf. Syst. 5 (3) (2009) 1–22.

[62] V. D. Blondel, A. Gajardo, M. Heymans, P. Senellart, P. V. Dooren, A measure of similarity between graph vertices: Applications to synonym extraction and web searching, SIAM Rev. 46 (4) (2004) 647–666. `doi:10.1137/S0036144502415960`.
URL `http://dx.doi.org/10.1137/S0036144502415960`

[63] F. Thung, D. Lo, J. Lawall, Automated library recommendation, in: 2013 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 182–191. `doi:10.1109/WCRE.2013.6671293`.

[64] G. Jeh, J. Widom, Simrank: A measure of structural-context similarity, in: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '02, ACM, New York, NY, USA, 2002, pp. 538–543. `doi:10.1145/775047.775126`.
URL `http://doi.acm.org/10.1145/775047.775126`

[65] R. L. Glass, I. Vessey, Contemporary application-domain taxonomies, Software, IEEE 12 (1995) 63 – 76. `doi:10.1109/52.391837`.

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

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

[68] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent Dirichlet Allocation, Journal of Machine Learning Research 3 (Jan) (2003) 993–1022.

[69] J. Ramos, Using tf-idf to determine word relevance in document queries (1999).

[70] P. T. Nguyen, J. Di Rocco, R. Rubei, D. Di Ruscio, CrossSim tool and evaluation data, `https://doi.org/10.5281/zenodo.1252866` (2018).

[71] K. Crowston, H. Annabi, J. Howison, Defining open source software project success, ICIS 2003 Proceedings (2003) 28.

[72] G. Von Krogh, S. Spaeth, K. R. Lakhani, Community, joining, and specialization in open source software innovation: a case study, Research policy 32 (7) (2003) 1217–1241.

[73] R. H. Lopes, Kolmogorov-smirnov test, International encyclopedia of statistical science (2011) 718–720.

[74] E. W. Weisstein, Bonferroni correction.

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

[76] R. Subramanyam, M. S. Krishnan, Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects, IEEE Transactions on software engineering 29 (4) (2003) 297–310.

[77] S. Khalid, S. Zehra, F. Arif, Analysis of object oriented complexity and testability using object oriented design metrics, in: Proceedings of the 2010 National Software Engineering Conference, ACM, 2010, p. 4.

[78] U. L. Kulkarni, Y. Kalshetty, V. G. Arde, Validation of ck metrics for object oriented design measurement, in: Emerging Trends in Engineering and Technology (ICETET), 2010 3rd International Conference on, IEEE, 2010, pp. 646–651.

[79] A. Capiluppi, N. Ajienka, The relevance of application domains in empirical findings, in: Proceedings of the 2nd International Workshop on Software Health, SoHeal@ICSE 2019, Montreal, Canada, May 30, 2019, 2019.

[80] G. Succi, W. Pedrycz, S. Djokic, P. Zuliani, B. Russo, An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite, Empirical Software Engineering 10 (1) (2005) 81–104.

[81] A. Marcus, D. Poshyvanyk, The conceptual cohesion of classes, in: 21st IEEE International Conference on Software Maintenance (ICSM'05), IEEE, 2005, pp. 133–142.

[82] J. P. Flynt, O. Salem, Software engineering for game developers, Course Technology PTR, 2005.