Two different facets of architectural smells criticality: an empirical study

Ilaria Pigazzini[1], Davide Foppiani[1] and Francesca Arcelli Fontana[1]

[1]University of Milano - Bicocca, Milan, Italy

Abstract

Architectural smells (AS) represent symptoms of problems at architectural level that have an impact on architectural debt. It is important to identify among them the most critical ones, so that developers can prioritize them for their removal. In order to evaluate the criticality of AS, in this paper we consider two facets: the PageRank metric, to assess the centrality of a smell in a project, and Severity, a metric to estimate the cost-solving of smells. We have proposed these two metrics in a previous work and here we perform an empirical analysis of the evolution and correlation of these metrics in the version history of 10 projects (at least 22 versions each, 264 projects in total). The analysis of the evolution is useful in order to identify which architectural smells types tend to become more critical. The analysis of the correlation is useful to study whether the criticality of a smell has an influence on how much it costs to remove it, and vice-versa.

Keywords

Architectural Smells, Architectural Debt, Architectural Smells criticality, Architectural Smells evolution, Empirical study

1. Introduction

Architectural debt can be monitored through different issues, such as through the presence of architectural smells in a project. Architectural smells (AS) are design decision that negatively impact internal software qualities and are symptoms of architectural debt [1], [2]. Software systems affected by AS are difficult to maintain and evolve, hence it is important to study them and identify solutions to support developers in their removal, in particular the removal of the most critical ones (*AS prioritization*).

In such terms, *criticality* of an AS models the degree of removal urgency associated to the AS, i.e., the smell should be removed as soon as possible because it affects a part of the project which is important for the developers (e.g., frequently changed or highly referenced) or has a strong impact on the maintainability of the project.

However, it is not trivial to model and evaluate the importance and urgency of the removal of an AS. In the literature, the identification of the best metrics to be used for the evaluation of criticality is considered a complex task [3], mainly because it is tightly connected to how smells are perceived by developers [4] and such perception is subjected to many variables, such as the developer experience, code ownership [5], whether the smell is located in a central part of the project and other facets. Moreover, while criticality gives us information about the removal urgency, there is another aspect connected to the removal of smells which can be considered and quantified. AS have a *cost-solving* (cost of fixing, cost of refactoring), which is the effort needed to remove a smell from the system [6]. This variable depends less from the perception of the developers but more from the specific characteristics of the interested AS.

To resume, during AS management, developers can take into consideration two distinct aspects concerning smells: their criticality, i.e., how much is important to remove them as soon as possible (urgency), and their cost-solving, i.e., how much it cost to remove them.

Both criticality and cost-solving are particularly relevant for developers when making decisions about AS management: for instance, to choose which smell to refactor first [1][5]. A developer may prefer to refactor first the smells which require less time to be solved (low cost solving) to quickly enhance the quality level of the project, instead of fixing the most critical ones. On the other hand, the developer may decide to remove the most difficult/critical ones, but to make this decision, different factors must be considered: it can be too expensive and risky; too many changes could compromise other parts. Perhaps, the most difficult AS was created by design choice and no better solution is available, as in the case of cycles created by callbacks for event listeners in GUI components [1][7]. Finally, the most critical AS could appear in a not-central part of the project, such as a deprecated, unessential package, and could be not interesting for the developers.

In this paper, we consider two metrics, PageRank and Severity, and we propose to use them to model the criticality (PageRank) and the cost-solving (Severity) of three

MSR4SA'21: 1st International Workshop on Mining Software Repositories for Software Architecture, September 15–17, 2021, Virtual

EMAIL: i.pigazzini@campus.unimib.it (I. Pigazzini); d.foppiani@campus.unimib.it (D. Foppiani); arcelli@disco.unimib.it (F. A. Fontana)

ORCID: 0000-0003-2629-6762 (I. Pigazzini); 0000-0002-1195-530X (F. A. Fontana)

AS based on dependency issues, namely: Cyclic Dependency, Unstable Dependency, and Hub-Like Dependency (see Section 3.2). *PageRank*, inspired by the well-known metric from Brin and Page [8] is a measure that estimates whether an AS is located in an important part of the project [9], where the importance is evaluated according to how many parts of a project depend on the ones involved in the AS (as a sort of centrality measure of the AS). We want to use PageRank as a proxy of AS criticality, i.e., the higher the PageRank, the higher the criticality of the AS. *Severity*, defined by us, is a measure associated to each specific type of AS and is computed through the metrics used to detect each smell. Our idea is that the AS characteristics, such as the number of system dependencies it affects, are useful to estimate how much effort is required to refactor the smell (cost-solving), e.g., a smell which involves many dependencies will require a deep analysis and a lot of time to be solved.

We have considered these two metrics in a previous study [10], where PageRank and Severity have been evaluated on only 6 single-version projects. We have now extended the study by conducting an empirical evaluation on a total of 264 versions of 10 projects with the aim to empirically study criticality and cost-solving during the evolution of the projects, and investigate whether there is a correlation between the trends of the two metrics, to answer the following Research Questions (RQ):

RQ1: How PageRank and Severity of the smells evolve in the version history of a project?

RQ2: Can we find some correlation between PageRank and Severity by considering each type of smell?

The answer to *RQ1* aims to analyze if the values of the two metrics tend to increase or decrease in the version history of the projects. Moreover, we are interested in understanding which AS type(s) tend to become more critical and/or difficult to remove in the version history of a project, where the criticality is evaluated through the PageRank and the cost solving is estimated with the Severity metric. In this way a developer can decide to focus the attention on these types of smells first.

The answer to *RQ2* allows to evaluate the correlation between the criticality and the cost solving of a smell. If for example the values tend to go together, highly correlated, for a specific type of AS, it means that as long as the smell is critical, it is also hard to remove and vice-versa: in this case, the two metrics would produce the same ranking of smells, i.e., the prioritization of the smells would be equal by considering one of the two metrics interchangeably. In case of positive correlation, it could be also in any case interesting to analyze possible outliers with different values of the metrics (high/low) and better capture the relevance of the metrics (see examples in Section 4.2). We could find that the two metrics have a strong positive correlation for a specific type of smell, and not for other smells. This scenario can outline the relevance of the metrics for each type of smell. Otherwise, no correlation, we could infer that there is no link between the urgency of removing a smell and the cost of removing a smell, as computed by the proposed metrics. In this case a developer can decide to not remove an AS with low PageRank and high cost solving, and to remove first an AS with high PageRank and low cost solving, since this AS could become more critical since it appears in a central part of the project.

We aim with our study to provide developers insights on the evaluation of criticality and cost solving of AS through the PageRank and Severity metrics. Severity metric is focused on evaluating the cost solving in terms of the number of project dependencies affected by the smells, while PageRank is more focused on the importance (criticality) of the affected components (classes/-packages). Hence, both metrics could be useful to determine the prioritization of AS, i.e., help the developer in choosing which smell to refactor first depending on the developer's needs, i.e., the need to address the most critical ones first or the most expensive ones.

We have considered the two metrics in the computation of an Architectural Debt Index [11] based on the number of the AS found in a project and their criticality measured in terms of both PageRank and Severity metrics. The results of this study can be useful also to evaluate whether the two metrics truly capture different aspects of a smell or not. In the latter case, one of the two metrics could be left out.

The paper is organized through the following sections: in Section 2 we introduce some related work, in Section 3 we describe the study design, in Section 4 we provide the results we obtained to answer the RQs. Section 5 presents the discussion of the results and Section 6 outline some threats to the validity of the work. Finally in Section 7 we conclude our work by outlining some threats to validity and future developments.

2. Related Work

We first briefly describe some empirical studies on architectural smells.

Le et al. [12] investigated the nature and impact of architectural smells through a large empirical study, by exploiting the projects' issue trackers to analyze the impact of smells on software development; Arcelli et al. [13] studied the relationship between code smells and architectural smells and found that architectural smells are independent from code smells; Sharma et al. [14] conducted an empirical study to investigate the relationship between design and architectural smells in C# projects. Finally Herold [15] performed a preliminary empirical study to investigate the relationship between architectural smells and architectural degradation, the latter mea-

sured through the number of architectural violations.

With respect to these previous papers, we performed an empirical study focused on the evaluation of different facets of architectural smells criticality, not previously studied in the literature according to our knowledge.

We now outline some related works done in the literature on the evaluation of criticality and prioritization of code or architectural smells. What distinguishes the following works is the kind of information used to estimate the priority of a smell. *For instance, concerning code smells*, Vidal et al. [16] presented an approach to identify the most critical smells based on a combination of three criteria, namely: past component modifications, important modifiability scenarios for the system and relevance of the kind of smell. Also Rani et al. [17] proposed a methodology for code smell prioritization. First, it detects smelly classes using structural information of source code, then mines change history, as done by Vidal et al., to prioritize the smells. Always according to code smells studies, Sae Lim et al. [18] exploited the *developers' context* (a list of issues extracted from an issue tracking system) to define priority. Instead, Arcelli et al.[19] proposed a severity index of the smells based on how the metric thresholds used for the smells detection are exceeded. Similarly, Guggulothu et al. [20] proposed a prioritisation approach for four code smells (Long Method, Feature Envy, God Class and Data Class), depending on their impact on design quality, where the impact is measured depending on the overcome of a set of metrics such as coupling, size, complexity and cohesion. Moreover recently, Pecorelli [5] proposed a machine learning approach to prioritise the application of refactoring on code smells. They generated a rank of code smells according to the perceived criticality that developers assign to them.

According to architectural smells, there are fewer studies about prioritization. Martini et al. [1], performed a study on the analysis of the most critical AS through the feedback of the developers of two industrial projects. The smells having top refactoring priority in the opinion of practitioners are the ones with the highest negative impact on the maintainability and evolvability of the project. On the same line, Oliveira et al. [21] investigated criteria that developers use in practice to prioritize design-relevant smelly elements with the aim to develop a set of prioritization heuristics. From their results, two out of nine heuristics reached an average precision higher than 75%. Finally, Vidal et al. [3] presented and evaluated a set of five criteria for ranking groups of code smells as indicators of architectural problems in evolving systems.

According to our knowledge no extensive work has been previously done on the analysis of the evolution and correlation between criticality and cost-solving, evaluated in terms of PageRank of AS and Severity metrics. In a previous study [10] we only manually analyzed the two metrics by considering only 6 projects. Here we extend the previous work on a large number of projects (10 projects, 22 versions each, for a total of 264 versions), and we analyze the correlation existing between the two metrics through Spearman and Kendall correlation tests. Moreover, we study the evolution of the metrics in the project history. Finally, in this paper we propose to exploit PageRank as a proxy for criticality, and Severity as a metric to estimate cost-solving.

3. Case Study Design

We describe below the analyzed projects, the data we collected on AS, their Severity and PageRank and the data preparation and analysis.

3.1. Analyzed projects

We analyzed several versions of 10 projects, for a total of 264 versions (see Table 1). Most of the chosen projects were picked from the Qualitas Corpus [22]. We selected these projects since they have already been the subject of several studies, they are publicly available and enable the replication of this study. These data were also combined with data from the MavenRepository[1], also publicly available. We considered several releases for each project. To easily compare the different projects, we chose roughly the same amount of versions and preferred different releases, major or minor, over patches when possible. In general, in this paper we use the term version to refer both minors and majors. The chosen systems also vary in size and number of smells (see Table 1). In the column group *last version* we report the projects' size (in terms of classes/packages) and number of AS of the *last version* of the project in the development history.

3.2. Data collection

Architectural smells we performed this study by considering the AS detected with the Arcan tool[2] [23] described below, but other AS can be considered in the future [24]. We limited the analysis on the following three smells since they are the only ones for which we developed a Severity metric, contextually to the definition of our Architectural Debt Index (ADI) [11].

- *Unstable Dependency (UD)* describes a component (package) dependent on other components that are less stable than itself; This may cause a ripple effect of changes in the system. Instability of a component is measured with the metric proposed by Martin [25] as the ratio of outgoing dependencies to the total number of dependencies of

[1] https://mvnrepository.com/

[2] Download: https://drive.google.com/file/d/1WNx7FHRykbyOIxz92cDQpSL2rl_gEJ4P/view?usp=sharing

Table 1
Summary of the dataset

Project	#V	#Cl.	#Pkg	#AS	#CD-Cl	#CD-Pkg	#HL-Cl	#HL-Pkg	#UD	#AS
		last version			all versions					
Ant	24	1157	62	413	8131	2064	15	92	243	10545
Azureus	24	8148	480	7722	97172	29801	41	70	3478	130562
FreeCol	24	1310	35	1395	30488	1652	86	54	356	32636
Hibernate	24	2980	170	1172	12910	9026	18	129	1267	23350
JMeter	26	681	55	307	3930	2681	79	54	574	7318
JGraph	24	188	20	118	2602	79	79	1	51	2812
Jstock	24	865	19	665	13585	619	64	8	247	14523
Jung	22	705	40	133	894	658	31	27	270	1880
Lucene	31	1425	22	408	6241	407	9	59	187	6903
Weka	44	2423	80	1090	25241	5200	102	41	1042	31626

Acronyms. V: version, CL: classes, Pkg: packages, AS: Architectural Smells, CD: Cyclic Dependency, HL: Hub-Like Dependency, UD: Unstable Dependency

the component. *Consequences:* The components with an high instability are more prone to change with respect to the more stable ones, this means that the component which depends on less stable components is forced to change along with them.

- *Hub-Like Dependency (HL)* arises when a component (class or package) has outgoing and incoming dependencies with a large number of other components [26]; The affected component represents a unique point of failure for the system and also a dependency bottleneck. *Consequences:* The component in the middle of the hub is a unique point of failure and a dependency bottleneck. Moreover the logic inside a Hub-Like Dependency is hard to understand, and the smell causes change ripple effect.
- *Cyclic Dependency (CD)* refers to a component (class or package) that is involved in a chain of relations that break the desirable acyclic nature of a component's dependency structure. Components involved in a CD cannot be reused in isolation and a change on one component propagates to the other ones. *Consequences:* The components involved in a dependency cycle can be hardly released, maintained or reused in isolation. Moreover, a change on one affected component will propagate towards all the other ones involved in the cycle.

We considered these three AS because they are some of the most studied smells [27][13][11][15] and they are also perceived as important and detrimental for the quality of the software systems by practitioners[1][24]. In particular, these smells are based on dependency issues. Dependencies are of great importance in software architecture: components that are highly coupled and with a high number of dependencies are considered more critical, since they have higher maintenance costs. In particular, Cyclic Dependency is one of the most common smell and is considered the most critical smell by developers [1].

We used our Arcan tool for the AS detection, since it is publicly available, allows to easily detect the considered AS and has been previously validated [28]. We computed [3] the PageRank and Severity metrics related to the three types of smells and we reported the "granularity level" of the considered smells, either class or package. Our distinction between AS at class and package level can be mapped to another nomenclature adopted in the literature [14] which calls "design smells" our class AS and "architectural smells" our package AS.

We now report the definition of the two metrics under analysis.

Severity is a metric that we defined for each type of AS to estimate the AS cost solving. In particular, it evaluates different features of the smells which have an impact on the effort needed for its removal. For example, for the estimation of Hub Like Dependency cost-solving, we consider the number of dependencies affected by the smell, because this metric gives us information about how many parts of code a developer investigate/change/remove to refactor the HL.

Severity is computed differently for each type of AS: for **UD** it is evaluated through the number of *bad* dependencies which cause the Unstable Dependency smell, where for bad dependency we mean a reference from the affected package to the less stable packages i.e. if package B has high instability and package A has low instability, the dependency A $\rightarrow$ B is a bad dependency; for **HL** the Severity corresponds to the total number of dependencies which cause the HL smell (dependencies from a class/package directed to the hub and vice-versa);

[3] https://figshare.com/articles/dataset/_/13636472

for **CD** it is computed through the number of components involved in the cycle multiplied with the minimum number of times a cycle repeats itself. A dependency between two components can occur multiple times because we count the number of references from a class/package to the others. For instance, if there is a cycle between package A and B, caused by 5 classes belonging to A calling B, and B's classes calling A 3 times, the Severity value is equal to 3. This means that the cycle is repeated *at least* 3 times.

PageRank of an AS evaluate the criticality (urgency) associated to an AS. The PageRank value of a smell instance is computed as the mean value of the PageRank of the components (class or package) affected by the smell. The intuition is that components with high PageRank are important inside the project, where the *importance* [9] corresponds to how many parts of the project depend on the component. PageRank of a component is computed through the PageRank formula implemented by Brin and Page [8], executed on the dependency graph of the project:

$$PR(v) = \frac{1-d}{N} + d\left(\sum_{k=1}^{n} \frac{PR(p_k)}{C(p_k)}\right) \quad (1)$$

where, the vertex v is a node of the dependency graph associated to a project; $PR(v)$ is the value of PageRank of the vertex v; N is the total number of AS in the project; P_k is a vertex with at least a link directed to v; n is the number of the p_k vertexes; $C(p_k)$ is the number of links of vertex p_k; d (damping factor) is a custom factor fixed at 0.85, a default value defined by Brin and Page.

The range of the metric spans from 0 to infinite and higher values correspond to higher criticality. To associate a unique value of PageRank to a single smell instance, we compute the mean value of the PageRank scores of all the components involved in the smell. In this way, smells of any type can be ordered by this metric, from the most critical to the less critical.

Both Severity and PageRank are based on the project dependencies, however they are computed in different ways and aim to evaluate two distinct aspects: importance/criticality (for PageRank) and dependencies structure/cost-solving (for Severity). Hence, we performed a correlation analysis to investigate the possible relationship between the two metrics.

3.3. Data preparation and analysis

We ran Arcan and we pre-processed the output data in order to produce the dataset for our analysis. Other than Arcan, we exploited the Knime platform[4] and R programming language[5] for the processing and statistical analysis of the data. The resulting dataset is a collection of 262155 smells categorized by project, version, type, granularity level, Severity and PageRank. Table 1 shows the summary of our dataset, where we report the project size and the number of smell instances, divided by type: for each project (considering *all versions* in history) we show the number of detected CD at class and package level (*CD-Cl* and *CD-Pkg*), of detected HL at class and package level (*HL-C* and *HL-P*), of detected UD (*UD*) and the sum of all project's AS (*AS*). A *smell instance* corresponds to one occurrence of the smell in the project, thus the reported numbers are the counts of all the occurrences.

We studied two different aspects: 1) Severity and PageRank evolution, in order to answer RQ1; 2) Severity and PageRank correlation to answer RQ2.

Concerning evolution, we analyzed the evolution of the two metrics for each type of smell in order to study their different behaviours. We summarised the data for each version by averaging the values of both metrics with respect to the total number of smells detected in the version. We conducted trend analysis to understand how the average values of PageRank and the different types of Severity evolve overtime. We exploited the *Mann-Kendall test*, which is a non-parametric test able to assess if there is a monotonic upward or downward trend of the variable of interest over time. The null hypothesis for this test is that there is no monotonic trend in the series. The alternate hypothesis is that a trend exists. This trend can be positive, negative, or non-null. We also analyzed the two metrics' evolution respect to the evolution of the size, where size corresponds to the number of classes and packages of the projects under analysis, to check whether the two things are correlated. We ran Spearman and Kendall correlation tests to investigate this aspect.

Concerning the correlation analysis of PageRank and Severity, we first tested the normality of our data. Given the large size of our dataset, we used Q-Q plots [29] to evaluate if the measures do not follow a normal distribution. A Q-Q plot is a graphical method for comparing two probability distributions by plotting their quantiles against each other. These plots are often used when the dataset is large enough to introduce bias in the Shapiro-Wilk test [30], which is a commonly used normality test. The Q-Q plots of all the projects showed a non-normal behaviour. Then, we tested the correlation between Severity and PageRank for each version of the projects. We computed the correlation on the metrics data of all smell type together and also separately for each smell type. We also computed the correlation separately for each granularity level, to contextualize the results at package or class level. Given the non-normal distribution of our data, we chose the Spearman's [31] and Kendall's [32] coefficients to calculate the correlation.

[4] https://www.knime.com/knime-analytics-platform
[5] https://www.r-project.org/

4. Results

We report the results both for PageRank and Severity evolution and their correlation. At the end of each section, we also report the answer to the relative RQs. All the results and plots can be found in the replication package[6].

4.1. Evolution results

In order to answer RQ1, we checked the trend of PageRank and Severity values throughout the versions of the projects. For every project and for both PageRank and Severity, we run the Mann-Kendall test. Table 2 and 3 show the outcome of the test, namely reporting the *Trend* (increasing + or decreasing -), the *P-value* and the *Reference AS* (the type of smell which the PageRank refers to) for PageRank, while *Granularity* (class or package) for Severity. The tables report only results where $p - value < 0.05$, i.e., there is a trend. We outline from Table 2 and 3 the following remarks:

- PageRank and Severity show a trend during time in few projects. We found PageRank trend in four over ten projects, while Severity showed a trend in five projects. The tables only show the projects with a positive or negative trend.
- Concerning the Severity of CDs, we observed both positive and negative trend at class level, in 4 projects, and a negative trend at package level, in one project.
- Concerning the Severity of HLs, we had examples at both class and package level of positive trends.
- The Severity metric of Unstable Dependency smell does not show a trend in any project, and we could notice only one project (Hibernate) where the PageRank of UD smells had a trend.

We extended our analysis to see if the project size (measured by number of classes and packages) is correlated with the values of PageRank and Severity. We tested it for each project over its development evolution. We then analyzed the distribution of the correlation on the data of all projects. The first thing we noticed is that the number of classes and packages increases overtime. However, this does not happen for Severity and PageRank values: we do not find a significant correlation between size and the metrics except for the correlation between PageRank computed on AS on packages and the number of packages in the system. The correlation values, computed for all the projects, have range in $[0.34, 0.89]$, with median equals to 0.74. We hypothesise that the correlation is high for PageRank because of how it is computed: the more the number of packages, the more the dependencies and higher the PageRank values are. For this reason, one may say that this should be true also for PageRank computed on classes correlated with the number of classes: instead, their correlation values range in $[-0.87, 0.9]$ with median equals to 0.45. This result may be due to the high variance in the number of classes among the projects (variance which is smaller for what concerns packages).

Table 2
Mann-Kendall results - PageRank

Project	Trend	P-value	Reference AS
Ant	+	0.009867	CD-package
Azureus	+	2.77E-05	CD-class
Azureus	+	0	CD-package
Azureus	+	3.81E-06	HL-class
Azureus	+	0	HL-package
Azureus	+	0	UD-package
Hibernate	+	0.030929	CD-class
Hibernate	+	0	CD-package
Hibernate	+	0.000677	HL-class
Hibernate	+	0	HL-package
Hibernate	+	2.38E-07	UD-package
Jgraph	+	0.001375	HL-class

Table 3
Mann-Kendall results - Severity

Project	Trend	P-value	Granularity
Severity - CD			
Azureus	+	0.024848	class
Hibernate	+	0.000291	class
Jstock	-	0.025486	package
Jung	-	0.039728	class
Lucene	-	3.25E-06	class
Severity - HL			
Jstock	+	0.002832	package
Lucene	+	0.000422	class
Weka	+	0.002132	class
Weka	+	0.005923	package

RQ1 Answer *How PageRank and Severity of the smells evolve in the version history of a project?*: in general we found that the average values of PageRank and Severity do not have a trend (neither positive or negative) over time. Concerning the comparison with projects' size evolution, we found out that PageRank computed on packages show a positive correlation with the evolution of the number of packages: this is reasonable, since the increase/decrease in the number of packages has an impact also on the creation/deletion of package dependencies, thus on PageRank.

[6] https://figshare.com/articles/dataset/_/13636472

Table 4
Severity and PageRank correlation (last version only)

Project	Version	Spearman	P-value	Kendall	P-value
Ant	1.10.7	0.582	< 0.001	0.46	< 0.001
Azureus	4.8.1.2	0.871	< 0.001	0.704	< 0.001
FreeCol	0.10.7	0.809	< 0.001	0.64	< 0.001
Hibernate	4.2.2	0.719	< 0.001	0.573	< 0.001
JMeter	5.2.1	0.575	< 0.001	0.455	< 0.001
JGraph	5.13.0.0	0.664	< 0.001	0.581	< 0.001
Jstock	1.0.6w	0.621	< 0.001	0.494	< 0.001
Jung	1.7.6	0.643	< 0.001	0.506	< 0.001
Lucene	4.3.0	0.411	< 0.001	0.33	< 0.001
Weka	3.7.9	0.53	< 0.001	0.428	< 0.001

4.2. Correlation results

In order to answer RQ2, we report in Table 4 the results of the correlation between Severity and PageRank, evaluated on all AS, not considering their type. As can be seen, the majority of the projects presented a strong positive correlation ($\rho > 0.6$).

Following, we discuss the correlation results, but by considering the different types of AS. The coefficient values are bounded between:

- (**CDs**) 0.427 and 0.942 with Spearman's and between 0.214 and 0.812 with Kendall's;
- (**UDs**) 0.253 and 1 with Spearman's and between 0 and 1 with Kendall's;
- (**HLs**) -1 and 1 for both coefficients.

Due to their low occurrences, the metrics of HL and UD usually present a strong correlation. However, there are cases in some projects versions where the scarce number of detected smells makes this calculation misleading: in some cases correlations are very high, in other ones are very low (fluctuate).

On the other hand, CD is the most common smell in the dataset and this has an effect on the correlation values: they largely vary in the dataset, making CD the smell type with some of the highest correlation values and at the same time the smell with some of the lowest correlation values. However, a clear result is that for all projects the correlation at package level between PageRank and Severity of CD is strong, with the exception of JGraph (see the following paragraph).

Observations on weak and negative correlations From Table 4 we can observe that some projects, such as JMeter, Lucene, Weka and Ant show a weak correlation between the two metrics. We aim to investigate these behaviours and we start by analyzing two projects: JMeter, having a weak correlation, and JGraph, showing non-positive correlation values for CDs at package level. We focus on the last version of both projects because it is associated to the most updated codebase, hence we assume it is the most exemplary for them.

By analyzing the correlation coefficients of *JMeter*'s AS, we noticed that when they are calculated separately for each AS type, they present higher values than the ones reported in Table 4. Using Spearman's as an example: 0.575 is the ρ value by not considering the AS type and 0.638, 0.9, 0.881 are the values for CDs, HLs and UDs respectively. The values seem to imply that actually, while the correlation in general is weak for this project, when we look at the specific smell types, the two metrics tend to be positively correlated. However, the number of HLs and UDs in JMeter is very small compared to the number of CDs. Since correlations computed on few observations are not significant, we can conclude that only the correlation value computed on CDs is relevant for JMeter, and it explains why the overall correlation value is weak for this project.

If we closely analyze *JGraph* evolution, initially it shows a negative correlation for CDs at package level, which progressively increases (0.2 in version 5.10.0.1) and becomes strongly positive (0.73) in version 5.12.1.0. We further investigated what caused these changes in the correlation values. In the first versions with negative correlation we observed 3 CDs at package level, two of them with similar Severity and PageRank values and one with a strongly higher PageRank value, probably the cause of the negative correlation. After version 5.10.0.1 we noticed the presence of a 4th one. Its Severity was in line with the others and also its PagerRank: this likely balanced the PageRank values and subsequently caused the increase of the positive correlation.

Hence we can conclude that the variations in the correlations values from negative to positive were due to the introduction of a new smell instance, whose metrics values strongly impacted the correlation values due to, as for JMeter, the general small amount of smell instances. However, this specific case does not represent a common behaviour in our dataset.

RQ2 Answer *Can we find some correlation between PageRank and Severity by considering each type of smell?*, we found out that the smell type showing the highest PageRank and Severity correlation is CD at package level. However, also the other types, HL and UD, showed strong correlations, but given the lower amount of HL and UD instances, we consider the result regarding CDs more meaningful. We also investigated specific cases of projects with weak correlation and negative correlation but we did not find further insights.

5. Discussion

We found a strong correlation between PageRank and Severity. This means that, concerning the analysed data and the considered smells, the criticality and the cost-solving of smells go hand in hand: in the case of this study, if a smell affects an important (unimportant) part of the system, then it will also have a high (low) cost solving. We can outline two different interpretations of the results. The positive correlation could be due to the nature of the two metrics, both bounded to the dependencies of the system. In this case, the conclusion would be that PageRank and Severity capture the same characteristic of the smells, and one of the two is redundant. As consequence, in the ADI computation [11], only one of the two metrics should be used to evaluate AS criticality.

However, given how the metrics are defined, they differ one from the other. Severity takes into account the dependencies which are directly affected by the smell, while PageRank considers also dependencies outside the smell which converge towards the components affected by the smell. Take for instance the Severity of CD, which is based on the dependencies forming the cycle and their weight. If the components involved in the cycle have a high PageRank, it means that they are involved in many dependencies with many other parts of the system, which is unliked from the fact that those components are part of the cycle. With such premise, the two metrics would capture different aspects of the smells, and their positive correlation could mean that critical parts of the system attract AS which are more expensive to solve.

Moreover, one could ask where is the difference in using PageRank when we could use simple coupling metrics such as FanIn and FanOut [25]. However, when evaluating the coupling of a component, such metrics take into account only the incoming or outgoing dependencies of the component itself. On the contrary, the PageRank value of a component takes into account the PageRank of all the components belonging to the dependency graph. In particular, the PageRank of a component is defined recursively and depends on the number of dependencies and the PageRank metric of all the components that reference it (incoming dependencies). In this way, a component having many incoming dependencies but referenced by components with few incoming dependencies, is less important with respect to another component with many incoming dependencies and referenced by other components with many incoming dependencies. That is why PageRank is said to evaluate the importance of a component with respect to the entire graph.

From our analysis it results that the positive correlation is particularly evident in the case of CD. The reasons behind the CD Severity high correlation can be multiple: a part of code with high PageRank is interested by more changes [33] with respect to other parts of code, and thus more open to the introduction of (structurally complex) CDs. This is interesting because in the literature we find studies which confirm the correlation in the other direction [12], i.e., the presence of AS makes the components more prone to change: if our hypothesis can be further corroborated, the conclusion would be that the relationship between PageRank and CD Severity is like a dog chasing its tail, one triggers the other. Another reason could be that components with high PageRank are involved in a high number of dependencies, thus still making easier for a developer to wrongly introduce new entangled dependencies and create cycles very difficult to remove.

To conclude, there is a positive correlation between AS Severity and PageRank, however at the moment we cannot draw a definitive conclusion about how to interpret this finding. We plan to conduct a validation of our results with developers from industry, who could evaluate the ability of the two metrics to capture criticality and cost-solving, and also manually check the specific cases where smells have high PageRank and high Severity.

6. Threats to validity

Our study presents some *threats to validity* which we address by following the structure suggested by Yin [34]. Concerning the **construct validity**, the two metrics, PageRank and Severity, may not measure what we claim they do, i.e., the criticality of the AS. However, this is a preliminary study and the next step is to validate the current definition of the metrics with developers, by letting them check whether the prioritization produced by the metrics is significant or not. Other threats regarding the **internal validity** could be related to the choice of the statistical methods used for the correlation analysis and their implementation in the used tools, but we exploited very well known and used tools (R language). Moreover, we did not validate the two metrics by investigating the perception of developers of PageRank and Severity. However, PageRank was adopted in other studies as software ranking metric [35][33][36], and we plan for the future

to validate Severity in industrial setting. Threats to **external validity** could be caused by the fact that we only analyzed projects written in Java and publicly available. However, we partially mitigate such issues by analyzing 10 projects with more than 22 versions each. Moreover, the high number of CDs could have reduced the effect of the other types of detected AS in the results. We could have mitigated this aspect by sampling the CD instances and thus balancing the dataset. However, this would additionally reduce the size of the dataset, mining the validity of the CD results too. In the future, we aim to extend the study with additional data for the smells and further remediate to this threat. Finally, concerning threats to the **reliability** of the study, Arcan could be subjected to a systematic bias in the detection, partially mitigated by the provided replication package and the fact that the tool has been validated on open source and industrial projects [23] [28] [1] [24]. Moreover, some threats could occur due to errors in the data extraction and preparation phases, resulting in errors in the construction of the dataset. However, we carefully checked every stage of the data preparation and relied on the support of Knime[7].

7. Conclusion

We performed an empirical analysis on 22 versions of 10 projects of two software metrics, Severity and PageRank, in order to evaluate the cost-solving and criticality of AS. We also performed this evaluation with the perspective to better understand if in the ADI computation both the two metrics have to be used or not, if they provide hints on the criticality evaluation of the AS that have to be both taken in consideration. To conclude, from the analysis of the evolution and correlation of PageRank and Severity we found out that the two metrics tend to be correlated, except for some extreme cases. It could be useful for developers to analyze the specific cases where AS have high PageRank and low Severity (and vice-versa), since they could indicate smell instances which require a tailored prioritization rationale: developers may be interested in identifying cases where the smell is easy to solve (low Severity) but in an important part of the system (high PageRank), and choose to refactor this case first; on the contrary, s/he could decide not to refactor a smell difficult to solve (high Severity) and in an unimportant (low PageRank) part of the system. We can assert that such smells are a signal that both PageRank and Severity could be useful to define different refactoring priorities, from different points of view. In particular, PageRank can be used to identify parts of code which need a continuous inspection, while Severity can be used to evaluate the cost-solving for the AS removal.

[7] https://www.knime.com/knime-analytics-platform

The smell type presenting the strongest correlation is CD, suggesting that highly critical components (with high PageRank) attract CDs hard to solve (with high Severity). Thus, developers should pay a lot of attention to CD smell, also because CD is the most common AS and in particular those at package level tend to become more critical in terms of PageRank in the history of the project development. However, we do not exclude the possibility that the two metrics have strong correlation because they capture the same aspects of smells. In that case, we could exploit this information to refine the computation of our ADI and leave out one of the two.

In any case, we need to conduct a validation of both metrics and on the correlation results, with expert developers or by comparing the ranking provided by the metrics with information coming from issue trackers [12]. The intuition behind is that a component affected by a critical smell (with high PageRank and high Severity) should be also interested by many issues. In addition to the validation, in future developments we aim to extend this work by analyzing more projects, also coming from industry, and verify if the same results can be confirmed.

In this paper, we addressed the criticality evaluation of three AS, but the study can be extended also to other kinds of AS, e.g., Scattered Functionality and Feature Concentration, two smells which violates the separation of concerns principle. Given that such smells are not based on dependency issues, we shall define additional criticality metrics for them.

References

[1] A. Martini, F. Arcelli Fontana, A. Biaggi, R. Roveda, Identifying and prioritizing architectural debt through architectural smells: a case study in a large software company, in: Proc. of the European Conf. on Software Architecture (ECSA), Springer, 2018.

[2] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, I. Gorton, Measure it? manage it? ignore it? software practitioners and technical debt, in: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, 2015.

[3] S. Vidal, W. Oizumi, A. Garcia, A. Díaz Pace, C. Marcos, Ranking architecturally critical agglomerations of code smells, Science of Computer Programming 182 (2019) 64–85.

[4] D. Taibi, A. Janes, V. Lenarduzzi, How developers perceive smells in source code: A replicated study, Information and Software Technology 92 (2017) 223–235.

[5] F. Pecorelli, F. Palomba, F. Khomh, A. De Lucia, Developer-driven code smell prioritization, in: Proceedings of the 17th International Conference on Mining Software Repositories, MSR '20, ACM, 2020.

[6] L. Rizzi, F. A. Fontana, R. Roveda, Support for architectural smell refactoring, in: Proceedings of the 2nd International Workshop on Refactoring, IWoR@ASE, 2018, pp. 7–10.

[7] I. Pigazzini, F. A. Fontana, B. Walter, A study on correlations between architectural smells and design patterns, J. Syst. Softw. (2021).

[8] S. Brin, L. Page, The anatomy of a large-scale hypertextual web search engine, in: Seventh International World-Wide Web Conference, 1998.

[9] I. Şora, A pagerank based recommender system for identifying key classes in software systems, in: 10th Jubilee International Symposium on Applied Computational Intelligence and Informatics, 2015.

[10] F. A. Fontana, I. Pigazzini, C. Raibulet, S. Basciano, R. Roveda, Pagerank and criticality of architectural smells, in: Proceedings of the 13th European Conference on Software Architecture, ECSA 2019, 2019.

[11] F. A. Fontana, P. Avgeriou, I. Pigazzini, R. Roveda, A study on architectural smells prediction, in: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2019.

[12] D. M. Le, D. Link, A. Shahbazian, N. Medvidovic, An empirical study of architectural decay in open-source software, in: 2018 IEEE International Conference on Software Architecture (ICSA), 2018.

[13] F. A. Fontana, V. Lenarduzzi, R. Roveda, D. Taibi, Are architectural smells independent from code smells? an empirical study, Journal of Systems and Software 154 (2019) 139 – 156.

[14] T. Sharma, P. Singh, D. Spinellis, An empirical investigation on the relationship between design and architecture smells, Empirical Software Engineering (2020).

[15] S. Herold, An initial study on the association between architectural smells and degradation, in: Software Architecture, Springer International Publishing, Cham, 2020, pp. 193–201.

[16] J. A. D. P. Santiago A. Vidal, Claudia Marcos, An approach to prioritize code smells for refactoring, Autom. Softw. Eng. 23 (2016) 501–532.

[17] A. Rani, J. K. Chhabra, Prioritization of smelly classes: A two phase approach (reducing refactoring efforts), in: 2017 3rd International Conference on Computational Intelligence Communication Technology (CICT), 2017.

[18] N. Sae-Lim, S. Hayashi, M. Saeki, Context-based approach to prioritize code smells for refactoring, Journal of Software: Evolution and Process (2017).

[19] F. A. Fontana, M. Zanoni, Code smell severity classification using machine learning techniques, Knowl. Based Syst. 128 (2017).

[20] T. Guggulothu, S. A. Moiz, An approach to suggest code smell order for refactoring, in: International Conference on Emerging Technologies in Computer Engineering, Springer, 2019, pp. 250–260.

[21] A. Oliveira, L. Sousa, W. Oizumi, A. Garcia, On the prioritization of design-relevant smelly elements: A mixed-method, multi-project study, in: Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse, SBCARS '19, Association for Computing Machinery, 2019.

[22] R. Terra, L. F. Miranda, M. T. Valente, R. S. Bigonha, Qualitas.class Corpus: A compiled version of the Qualitas Corpus, Software Engineering Notes 38 (2013).

[23] F. A. Fontana, I. Pigazzini, R. Roveda, M. Zanoni, Automatic detection of instability architectural smells, in: 2016 IEEE International Conference on Software Maintenance and Evolution,ICSME 2016, 2016.

[24] F. A. Fontana, F. Locatelli, I. Pigazzini, P. Mereghetti, An architectural smell evaluation in an industrial context, ICSEA 2020 (2020) 78.

[25] R. C. Martin, Object oriented design quality metrics: An analysis of dependencies, ROAD 2 (1995).

[26] G. Suryanarayana, G. Samarthyam, T. Sharma, Refactoring for Software Design Smells, 1 ed., Morgan Kaufmann, 2015.

[27] D. Sas, P. Avgeriou, F. A. Fontana, Investigating instability architectural smells evolution: An exploratory case study, in: Int. Conference on Software Maintenance and Evolution, ICSME, 2019.

[28] F. Arcelli Fontana, I. Pigazzini, R. Roveda, D. A. Tamburri, M. Zanoni, E. D. Nitto, Arcan: A tool for architectural smells detection, in: Int'l Conf. Software Architecture (ICSA 2017) Workshops, 2017.

[29] M. B. Wilk, R. Gnanadesikan, Probability plotting methods for the analysis of data, Biometrika 55 (1968) 1–17.

[30] S. S. Shapiro, M. B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (1965) 591–611.

[31] C. Spearman, The proof and measurement of association between two things, The American Journal of Psychology 15 (1904) 72–101.

[32] M. Kendall, J. Gibbons, Rank Correlation Methods, Charles Griffin Book, E. Arnold, 1990.

[33] R. Wang, R. Huang, B. Qu, Network-based analysis of software change propagation, The Scientific World Journal 2014 (2014).

[34] R. Yin, Case Study Research: Design and Methods, Applied Social Research Methods, SAGE Publications, 2009.

[35] F. Perin, L. Renggli, J. Ressia, Ranking software artifacts, in: 4th Workshop on FAMIX and Moose in Reengineering (FAMOOSr 2010), volume 120, Citeseer, 2010.

[36] W.-f. PAN, B. LI, Y.-t. MA, B. JIANG, Identifying the key packages using weighted pagerank algorithm, ACTA ELECTONICA SINICA 42 (2014) 2174.