2018 44th Euromicro Conference on Software Engineering and Advanced Applications

# Towards an Architectural Debt Index

Riccardo Roveda
Alten Italia, Milano, Italy
Email: riccardo.roveda@alten.it

Francesca Arcelli Fontana, Ilaria Pigazzini, Marco Zanoni
Department of Informatics, Systems and Communication,
Università of Milano-Bicocca
Email: {arcelli, pigazzini, zanoni}@disco.unimib.it

***Abstract*—Different indexes have been proposed to evaluate software quality and technical debt. Usually these indexes take into account different code level issues and several metrics, well known software metrics or new ones defined ad hoc for a specific purpose. In this paper we propose and define a new index, more oriented to the evaluation of architectural violations. We describe in detail the index, called Architectural Debt Index, that we integrated in a tool developed for architectural smell detection. The index is based on the detection of architectural smells, their criticality and their history. Currently only dependency architectural smells have been considered, but other architectural debt indicators can be considered and integrated in the index computation.**

***Index Terms*—architectural smells, architectural debt index, severity index, technical debt**

## I. Introduction

Many works have been done in the literature on managing technical debt ([1], [2], [3]) which consider different forms of technical debt at different levels, e.g., architecture, design, code, test, social, documentation and technological. As outlined and evaluated by Ernst et al. [4] architectural issues are the greatest source of technical debt. Hence, it is important to understand how to identify and manage architectural problems to avoid and reduce technical debt accumulation.

In this paper, we focus our attention on architectural debt [1], [5], [2], [6] and its possible evaluation. Different tools for software analysis, in particular commercial ones, provide a large number of evaluation measures through the computation of several metrics and technical debt indexes. In a previous paper [7], we briefly described five tools (SonarGraph, Structure101, Cast, InFusion, SonarQube) able to provide a Technical Debt Index, sometimes called in a different way, but with the same or similar purpose. We found that often architectural issues are not taken into account and when they are considered the main focus is on the detection of cyclic dependencies [8]. Many other architectural problems such as architectural smells ([9], [10]) are not considered. Moreover, different architectural smells/problems can be identified only by analyzing the development history of a project( [5]) and Technical Debt (TD) indexes usually do not take into account this kind of information too.

In this work we focus our attention on the debt that can be caused by the presence of architectural smells. Architectural smells are often introduced through the violation of some design principles or decisions [11] and may accumulate high maintenance and evolution costs, representing a source of architectural debt. For this reason, we started working on the definition of a new index, that we call *Architectural Debt Index*, with a focus on architectural smells (AS) and we integrated it in our tool for architectural smell detection, called Arcan [12].

The aim of our index is to evaluate the internal quality of a project in terms of: *1)* the detected architectural smells, *2)* their history, *3)* their severity (the most critical ones) and *4)* architecture design metrics. The index value of a project with a large number of smells and also critical smells (high severity) will be higher with respect to a project with less smells and low severity.

Hence in this work, we aim to answer the following research questions:

RQ1 How should a new index be formulated to more exhaustively evaluate architectural debt?
RQ2 How can we estimate the severity of an architectural smell?
RQ3 Is the new index based on architectural smell detection independent from another existing index based on code level issues?

The answer to research question *RQ1* gives and motivates the definition of a new *Architectural Debt Index (ADI)* based only on the detection of different architectural smells (AS), their severity and history. A high value of the index provides hints on the low architectural quality of a project. Hence, developers/maintainers can get hints on specific problems causing the debt and identify the most critical ones. Moreover, if the Architectural Debt Index during a project evolution tends to increase, this means that developers/maintainers did not focus their attention on the sources of this debt. This could be probably due also to the lack of many available tools able to easily identify architectural smells and/or to the high cost to remove them.

The answer to research question *RQ2* provides a severity estimation of the detected architectural smells. The severity is evaluated according to the violation degree of the metrics values used in the detection of the smell and the relevance of the part of the project affected by the smell, where the relevance is evaluated through a *PageRank* index (see Section IV-A2). The severity estimation allows to identify the most critical smells in terms of negative impact on a project, that can be prioritized to identify those to be removed first.

The answer to research question *RQ3* allows to evaluate if an index such as the ADI based on the detection of architectural issues is independent by an index focused on code level issues. For this analysis we considered the index provided by SonarQube, since this tool is probably the most known and used tool for software quality assessment and provides a technical debt index whose computation is essentially based on code level issues. The answer to this research question is important since, if the two indexes effectively are able to measure code and architectural debt and they are independent, this means that there is no relation or not a strong relation between the occurrence of issues at code and architectural level. Hence, developers/maintainers have to take care of both the two types of debt: removing code debt could not necessarily imply the reduction/removal of architectural debt and vice versa.

We compute the Architectural Debt Index on 109 open source projects of the Qualitas Corpus [13], where the projects have been classified in different categories. We evaluated the ADI also according to these categories and the size of the projects in terms of number of packages. Moreover, we analyzed the evolution of our index and the one of SonarQube in different major releases of 10 projects.

The paper is organized through the following sections: in Section II we outline some related works on tools which provide the computation of different kinds of technical debt indexes; in Section III the architectural smells detected through the Arcan tool; in Section IV the new Architectural Debt Index and in Section V its computation and its evaluation on a large dataset of projects; in Section VI the possible correlations among the Architectural Debt Index and the index of SonarQube. in Section VII the threats to validity and finally in Section VIII, we provide the answers to the Research Questions and discuss some future developments.

## II. Related Work

Different quality or technical debt indexes have been defined in the literature to evaluate a project, as the old Maintainability Index of Coleman [14]. We briefly cite below the indexes provided by some tools, while a more extended description of some of them can be found in [7].

Sonargraph tool evaluates Structural Debt[1] quantified through two measures: *Structural Debt Index* (SDI) and *Structural Debt Cost* (SDC). SonarQube[2] implements the *SQALE* [15] model for the estimation of Technical Debt. Three values are computed on the analyzed project, i.e., Technical Debt (TDI), Technical Debt Ratio (TDR) and SQALE Rating (SR). The TDI computation does not take into account architectural or dependency information. CAST[3] estimates the amount of principal in the Technical Debt (TD-Principal) of an application based on detectable structural problems. Structure101[4] shows a *Structural over-Complexity (SoC)* view to estimate the percentage of the system involved in architectural issues. Obviously, other tools are available which are able to compute a huge number of metrics also related to architectural issues, e.g., Massey Architecture Explorer[5] computes an Antipatterns Score and the Tangledness metric [16], Lattix[6] provides Stability, Cyclicality, and Coupling metrics, and STAN[7] supports the computations of different R. Martin's metrics [17]. With respect to the previous indexes, our index is focused only on architectural issues (e.g., AS) and takes into account different features not considered, according to our knowledge, in the previous indexes.

[1]Metrics and Queries Documentation v.7.2, hello2morrow, May 2011

[2]https://www.sonarqube.org

[3]https://www.castsoftware.com

[4]https://structure101.com/

## III. Architectural smells detected

We developed a tool to detect AS in Java projects, called Arcan [12]. We currently focused our attention on AS based on dependency issues, since components highly coupled and with a high number of dependencies cost more to maintain and hence can be considered more critical. Obviously dependencies are not the only indicator of AD, hence other AS or debt indicators will be considered in the upcoming future work. Arcan detects:

- *Unstable Dependency (UD)*: describes a subsystem (component) that depends on other subsystems that are less stable than itself. This may cause a ripple effect of changes in the system [18]. Detected on packages.
- *Hub-Like Dependency (HL)*: this smell arises when an abstraction has (outgoing and ingoing) dependencies with a large number of other abstractions [19]. Detected on classes and packages.
- *Cyclic Dependency (CD)*: refers to a subsystem (component) that is involved in a chain of relations that break the desirable acyclic nature of a subsystem's dependency structure. The subsystems involved in a dependency cycle can be hardly released, maintained or reused in isolation [18]. Detected on classes and packages and according to different shapes [12].
- *Implicit Cross Package Dependency (ICPD)*: captures hidden dependencies among files belonging to different packages [20], [21]. For hidden dependencies, we mean co-change relations that we can find only in the history of the project and not in the code. Files changed frequently together with hidden dependencies lead to a lack of modularity. Detected on files.

All computations are based on the dependency graph built from reading the project compiled files through a specialized Java library named BCEL. The graph represents the static structure of the code and illustrates the relationships among different software elements at different abstraction levels, such as composition, inheritance, package afference etc. The detailed description of the detection algorithms of CD, HL, UD smells and the building of the dependency graph can be found in [22]. For what concerns the *Implicit Cross Package Dependency*, the detection is done through the analysis of the dependency graph's evolution from the version history of the

[5]http://xplrarc.massey.ac.nz/

[6]https://lattix.com/

[7]https://stan4j.com/

project development. Some of the above AS are detected also by other tools prototypes, but with Arcan we exploit a different approach: the project-under-analysis is represented through a dependency graph which contains all the information regarding the project static structure and the results of the detection.

## IV. An Architectural Debt Index

This section is focused on the definition of a new index based only on the detection of the AS currently identified through Arcan. Obviously other AS can be considered in the future by considering other issues impacting architectural debt.

The index computation, integrated in the Arcan tool, takes into account:

- the *Number* of AS in a project;
- the *Severity* of an AS: assuming that some instances of AS are more critical than others, the Index takes into account a Severity measure defined according to each architectural smell type;
- the *History* of AS: the presence of an AS in the history of a project can have a different impact on the index (for example if an AS involves an increasing number of classes and packages in its evolution, it is considered more critical than other AS).
- the *Dependency metrics* of Martin [17] (Instability, Fan In, Fan Out, Efferent and Afferent Coupling) used for the AS detection.

By considering these factors, the Architectural Debt Index ($ADI$) of a project $P$ is defined as follows:

$$ADI(P) = \sum_{k=1}^{n} \left( \frac{1}{W}(ASIS(AS_k) * w(AS_k)) * History(AS_k) \right) \quad (1)$$

where:

- $n$: number of AS instances in a project $P$;
- $AS_k$: k-instance of an architectural smell;
- $W$: the total number of dependencies involved in at least one AS for all the AS in the project;
- $ASIS(AS_k)$: the *Architectural Smell Impact Score* (defined below in IV-A);
- $w(AS_k)$: the *Architectural Smell Weight*, i.e., the number of dependencies associated to the $AS_k$;
- $History(AS_k)$: the score associated to the trend evolution of the $AS_k$ (defined below in IV-B).

The dependencies are considered according to the dependency graph of a project, on which all the AS found in a project are mapped on. The dependencies are the number of unique vertices (classes or packages) of the subgraph directly affected by an architectural smell.

### A. Architectural Smell Impact Score

The *Architectural Smell Impact Score*, $ASIS$, is based on both the estimation of the severity of an AS and the importance of the subsystem where the AS is found. It is defined as the product of the *SeverityScore* associated to the $AS_k$ smell and the *PageRank* value of the $AS_k$, which estimates the importance of the project subsystem affected by the $AS_k$ smell (defined below).

It is not possible to define a general formula for the Severity Score and the PageRank computation suitable for all types of architectural smells. In addition, since we are combining the two values linearly, we need to mitigate potential non-linearities in their distribution, avoiding masking effects due to extremely large or small values.

The *SeverityScore* and *PageRank*, defined below, will both assume values in the range $[0, \infty)$ mapped to integer values in the range $[0, 1]$ (low to high respectively), through the $quantile(x)$ function which is the quantile associated to $x$ in the reference dataset.

Hence, we decided to compute the quantiles of the two values (Severity Score and PageRank) on a large dataset of 109 projects (see Table II) of the Qualitas Corpus [13] and use them to assign values to some thresholds such as: low, medium low, medium, medium high and high.

The $ASIS$ is defined as follows:

$$ASIS(AS_k) = SeverityScore(AS_k) * PageRank(AS_k) \quad (2)$$

.

Since both $SeverityScore()$ and $PageRank()$ return values between 0 and 1, $ASIS$ represents a $SeverityScore$ weighted by the "importance" ($PageRank$) of the subsystem where the AS appears.

*1) Severity Score:* The $SeverityScore(AS_k)$ is a value defined for each instance of AS according to each type of AS. The $SeverityScore$ for the AS detected by Arcan (Unstable Dependency (UD), Hub-like Dependency (HL), Cyclic Dependency (CD)) is defined as follows:

- If $AS_k$ is an UD, $SeverityScore(AS_k)$ is defined as: $quantile(NumberOfUnstableDependencies)$
- If $AS_k$ is a CD, $\forall$ e $\in$ edges(CD), $SeverityScore(AS_k)$ is defined as: $quantile(NumberOfelementsInCD{*}min(n\_occ(e)))$ where the $n\_occ(e)$ is the number of times the same type of edge among two vertices (e.g., class or package) occurs.
- If $AS_k$ is a HL, $SeverityScore(AS_k)$ is defined as: $quantile(TotalNumberOfDependencies)$.

The $SeverityScore$ of the ICPD smell computed considering the history of a project is evaluated through the last factor of the ADI computation, the $History(AS_k)$ (see Section IV-B).

*2) PageRank:* The $PageRank(AS_k)$ estimates whether the AS is located in an important part of the project, where the importance is defined by the value of the $PageRank$ algorithm executed on the dependency graph of the project (to evaluate if many parts (subsystems) depend on the part where the AS is involved). The PageRank, $PR$ is modeled starting from the one implemented by Brin and Page [23], as explained below:

$$PR(v) = \frac{1-d}{N} + d \left( \sum_{k=1}^{n} \frac{PR(p_k)}{C(p_k)} \right) \quad (3)$$

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 [23]. It can be changed according to the $PageRank$ value needed for every vertex and its minimum associated level of $PageRank$.

The $PR$ value is computed only on vertices of the dependency graph of both class and package types. $PageRank$ value $PR$ is used for all the types of AS detected by Arcan, but the $PageRank$ of an AS which involves multiple classes and packages is considered differently, e.g, a CD smell that involves two or more classes or packages. To compute the $PageRank$ when an AS involves more than one vertex, it is necessary to aggregate the data; a method to aggregate multiple values could be to take the maximum $PR$ value of the group.

The $PR$ of all the AS and the max of $PR$ of AS involving multiple classes or packages is computed as follows:

$$PageRank(AS_k) = \begin{cases} \text{If } AS_k \text{ is an AS among classes or packages:} \\ quantile(max_{j=1}^{n} PR(v_j)) \\ \text{If } AS_k \text{ is an AS of a class or a package:} \\ quantile(PR(v)) \end{cases}$$

where $v$ is the vertex (class or package) affected by $AS_k$, $n$ is the number of classes $v_j$ involved in an AS (among classes) or the number of packages $v_j$ involved in an AS (among packages).

## B. History

The presence of an AS in the history of a project can have a different impact on the index. Figure 1 shows an example of project evolution through a graphical annotation. Version 1 (V1) is one version of a project, followed by Version 2 (V2). V1 has two architectural smells, such as: AS1 and AS2. AS1 has been deleted from the V2 of the project. AS2 is equal to the AS3 smell detected in V2. The comparison is made by extracting the subgraphs (SG) affected by the AS and checking whether the SGs in V2 contain, extend or are equal to any of the subgraphs of the precedent version V1. In the case of deletion of AS, a SG in V1 would not be in relation with any SG in V2.

Hence, the factor of the ADI index related to the history of a project, $History(AS_k)$, is defined as the weight assigned to the trend of each type of smell as follows:

$$History(AS_k) = \begin{cases} \zeta & \text{If } AS_k \text{ has a Decreasing Trend} \\ \theta & \text{If } AS_k \text{ has an Increasing Trend} \\ \eta & \text{If } AS_k \text{ has a Stable Trend} \end{cases} \quad (4)$$

where the trends are evaluated as:

- *Decreasing Trend*: when the number of classes, files or packages involved in an AS is decreasing.

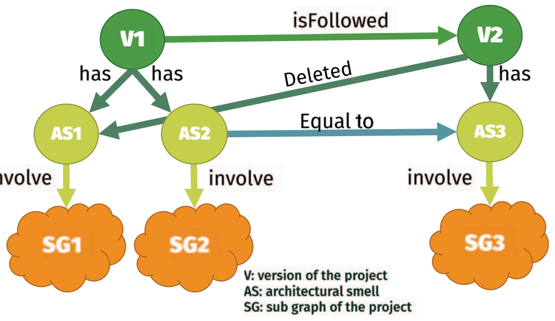


Figure 1: Example of AS evolution

- *Increasing Trend*: when the number of classes, files or packages involved in an AS is increasing.
- *Stable Trend*: when the number of classes, files or packages involved in an AS is stable.

and where $\zeta$, $\eta$ and $\theta$ are fixed according to the validation of the Index. Currently the values are fixed intuitively at 0.5, 1 and 2 respectively.

## V. Architectural Debt Index evaluation

In this section we evaluate the ADI index with and without considering the History component in the index computation.

*1) The ADI evaluation without History:* In the ASIS computation we have to consider the $SeverityScore$ and $PageRank$. We analyzed their values on a dataset of 109 projects of the Qualitas Corpus [24]: the quantile values of the dataset distribution are reported in Table I, where the $SeverityScore(AS_k)$ and $PageRank(AS_k)$ assume values between 0 and 1. In this evaluation we have considered only UD, HL and CD smells, while ICPD has been excluded since ICPD is based on the history of a project (see next subsection). Table I reports also the metrics used to compute the $SeverityScore$ (SS) described above: the number of unstable dependencies (NUD) for UD; the number of total dependencies (NoTD) for HL; the number of vertices (NoV) i.e., number of classes or packages according to the granularity level and the number of involved cycle (NoC) for CD.

Table II reports the values of the ($ADI$) index in the referenced dataset (without considering the factor related to the History of the projects) and its quantification as a score value in a range among 1 and 5 through the following function $q(ADI(P))$:

$$q(ADI(P)) = \begin{cases} 1 & \text{If } 0.00 \leq quantile(ADI(P)) \leq 0.20 \\ 2 & \text{If } 0.20 < quantile(ADI(P)) \leq 0.40 \\ 3 & \text{If } 0.40 < quantile(ADI(P)) \leq 0.60 \\ 4 & \text{If } 0.60 < quantile(ADI(P)) \leq 0.80 \\ 5 & \text{If } 0.80 < quantile(ADI(P)) \leq 1 \end{cases} \quad (5)$$

where $quantile(ADI(P))$ is the quantile associated to the $ADI$ computed for the project $P$ of the reference dataset, e.g, given a project with $q(ADI(P))$ of 5, its ADI is worse than a project with a $q(ADI(P))$ of 2. Moreover, Table II shows every single factor involved in the computation of the ADI: $\Sigma ASIS$, $\Sigma ASIS * w$, $W$ and the number of the AS found in the projects for UD, CD and HL smells, both at package (PgK) and class (Cl) level.

From the data of Table II we can observe that projects with a high number of architectural smells have a higher $q(ADI(P))$, in fact Eclipse project has $q(ADI(P))$ of 5 and 20915 architectural smells; Cayenne has 8 architectural smells and has $q(ADI(P))$ of 1. Moreover, we can see that two projects with similar $W$ (the number of dependency of the project) got different $q(ADI(P))$, such as Batik and Freecol got 3 and 5 respectively due to the different number of architectural smells and the higher $ASIS$. Batik and Findbugs have similar $W$ and different $q(ADI(P))$ of 3 and 5 respectively, because the number of architectural smells are similar but the $\Sigma ASIS * w$ is bigger for Findbugs than Batik. Hence, Findbugs has bigger and worse architectural smells than Batik.

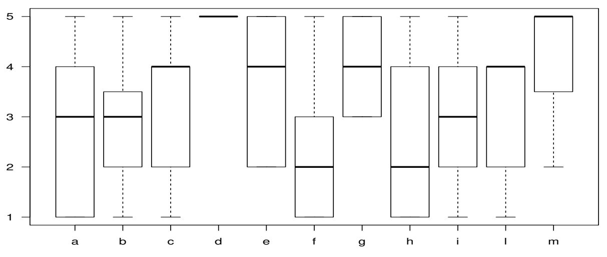
The JHotDraw and Tapestry projects have 389 and 398 architectural smells respectively, but they have $q(ADI(P))$ value of 1 and 2 respectively. Moreover, these values are lower than for projects with less architectural smells (e.g., JParse got the value 3 for $q(ADI(P))$), since the high number of architectural smells is mitigated mainly by the higher $W$.

Figure 2 shows the distribution of the $q(ADI(P))$ for all the projects of Table II according to the categories to which the projects belong (Figure 2a) and the number of packages per project (Figure 2b). All the projects in the Qualitas Corpus repository are classified in the following categories [13]: *a)* parsers/generators/make, *b)* 3D/graphics/media, *c)* diagram generator/data visualization, *d)* programming language, *e)* database, *f)* middleware, *g)* IDE, *h)* testing, *i)* tool, *l)* SDK and *m)* games. As shown in Figure 2a, all the projects in the programming language category got $q(ADI(P))$ of 5, it is the worst category of projects together with the games and IDE categories. The testing, generators and middleware projects are the best ones with box of boxplot starting from 1 and not bigger than 4. Moreover, the worst categories have big projects in terms of the number of packages and high $\Sigma ASIS * w$ (if compared to the better ones). As shown in Figure 2b, the highest $q(ADI(P))$ is related to projects having packages between 200 and 300, but the highest values belong to projects with more than 300 packages.

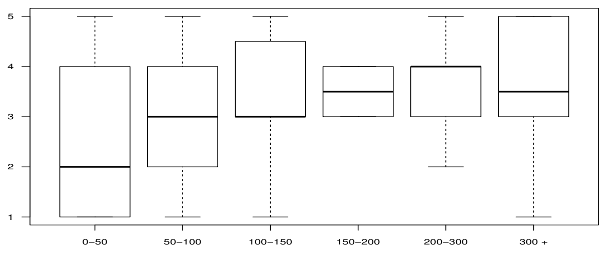
*2) The ADI evaluation with History:* For this computation we have to consider the project evolution. Hence, we evaluated ADI on more than 100 versions of 10 projects shown in Table III chosen in 7 different categories of the Qualitas Corpus repository.

Figure 3a shows the evolution of the Architectural Debt Index in several versions (i.e., major releases) per project. The majority of the projects have an increasing trend (i.e. the projects are getting worse) of the ADI index, i.e., the ADI in the last version is higher than in the first version. Although, Checkstyle shows a decreasing trend of ADI before a major release publication and after that an opposite trend.

We have considered also the evolution of the ADI in the projects according to the Lines of Code (LOC) of the projects. Figure 3b shows the evolution of the LOC metric. We can see that the LOC value in most of the projects is stable, with the



(a) $q(ADI(P))$ by Qualitas Corpus's Category



(b) $q(ADI(P))$ by NoP

Figure 2: Boxplots of $q(ADI(P))$ detected on the Qualitas Corpus projects

Table I: ADI's components Quantile and value associated

| Quantile | Unstable Dep. Package |  |  | Hub-Like Dep. Class |  |  | Cyclic Dependency Class |  |  |  | Cyclic Dependency Package |  |  |  |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
|  | $PR$ | SS | NUD | $PR$ | SS | NoTD | $PR$ | SS | NoC | NoV | $PR$ | SS | NoC | NoV |
| 0.00 | 1.08 | 1 | 1 | 4.35 | 1 | 21 | 0.78 | 1 | 1 | 2 | 1.84 | 1 | 1 | 2 |
| 0.05 | 2.85 | 2 | 1 | 9.08 | 1 | 33 | 0.90 | 35 | 1 | 2 | 8.62 | 28 | 1 | 2 |
| 0.10 | 3.85 | 3 | 1 | 11.54 | 1 | 39 | 0.96 | 72 | 1 | 2 | 12.97 | 74 | 1 | 2 |
| 0.15 | 4.77 | 5 | 1 | 15.44 | 1 | 44 | 1.05 | 113 | 1 | 2 | 17.61 | 132 | 1 | 2 |
| 0.20 | 5.63 | 6 | 1 | 18.24 | 1 | 52 | 1.16 | 162 | 1 | 2 | 22.71 | 197 | 1 | 3 |
| 0.25 | 6.71 | 8 | 1 | 22.38 | 1 | 59 | 1.32 | 218 | 1 | 2 | 28.53 | 254 | 1 | 3 |
| 0.30 | 7.81 | 10 | 1 | 25.93 | 1 | 66 | 1.53 | 283 | 1 | 2 | 34.82 | 305 | 1 | 4 |
| 0.35 | 8.94 | 12 | 1 | 30.04 | 2 | 69 | 1.82 | 354 | 1 | 2 | 42.11 | 354 | 1 | 4 |
| 0.40 | 10.38 | 14 | 1 | 34.53 | 2 | 74 | 2.20 | 435 | 1 | 2 | 51.55 | 435 | 1 | 5 |
| 0.45 | 12.14 | 17 | 1 | 37.71 | 2 | 81 | 2.84 | 527 | 1 | 3 | 62.94 | 542 | 1 | 5 |
| 0.50 | 13.69 | 20 | 2 | 46.65 | 2 | 85 | 3.97 | 632 | 1 | 3 | 75.38 | 679 | 1 | 6 |
| 0.55 | 15.83 | 24 | 2 | 51.08 | 3 | 92 | 5.66 | 747 | 1 | 4 | 90.00 | 861 | 1 | 7 |
| 0.60 | 18.46 | 28 | 2 | 56.68 | 3 | 96 | 8.59 | 889 | 1 | 5 | 106.77 | 989 | 1 | 8 |
| 0.65 | 22.85 | 33 | 2 | 64.48 | 3 | 102 | 14.32 | 1069 | 1 | 7 | 126.19 | 1159 | 1 | 9 |
| 0.70 | 26.70 | 39 | 3 | 75.57 | 3 | 108 | 23.62 | 1325 | 1 | 10 | 150.30 | 1428 | 1 | 11 |
| 0.75 | 32.64 | 46 | 3 | 82.23 | 4 | 117 | 35.32 | 1664 | 1 | 14 | 185.52 | 1744 | 1 | 12 |
| 0.80 | 41.35 | 57 | 4 | 93.06 | 4 | 129 | 51.64 | 2052 | 1 | 18 | 256.20 | 2216 | 1 | 15 |
| 0.85 | 51.60 | 75 | 5 | 114.51 | 5 | 139 | 79.75 | 2498 | 1 | 24 | 380.28 | 3416 | 1 | 19 |
| 0.90 | 72.29 | 95 | 6 | 137.18 | 5 | 165 | 139.14 | 3360 | 1 | 33 | 553.30 | 4366 | 1 | 27 |
| 0.95 | 117.32 | 128 | 9 | 167.26 | 7 | 194 | 263.39 | 4920 | 2 | 54 | 836.93 | 5751 | 2 | 49 |
| 1.00 | 1766.90 | 207 | 42 | 428.95 | 11 | 893 | 418.22 | 7231 | 93 | 102 | 1599.88 | 7214 | 36 | 79 |

$PR$: PageRank, SS: Severity Score, NoC: Number of Cycle, NoV: Number of vertices, NoTD: Number of Total Dependency, NUD: number of unstable dependencies.

exception of two projects (Hibernate and Ant). In Nekohtml, Emma and PicoContainer projects, we can observe that even if developers introduce few new LOC, the ADI increases (see Figure 3a). If we look at both Figures 3a and Figure 3b, we can observe that the ADI values increase also if the LOC values remain stable. In fact ADI does not consider LOC in its computation, but the number of AS.

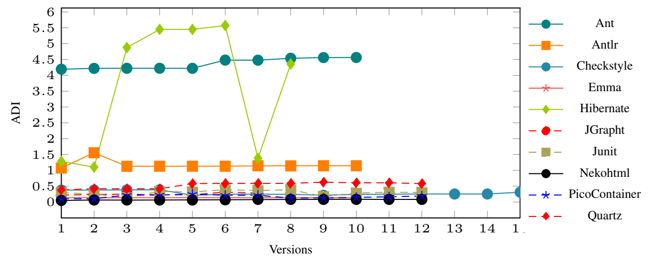
For what concerns an initial validation of our index, we looked for consistency between an existing architectural eval-

Table II: ADI computation and AS detection

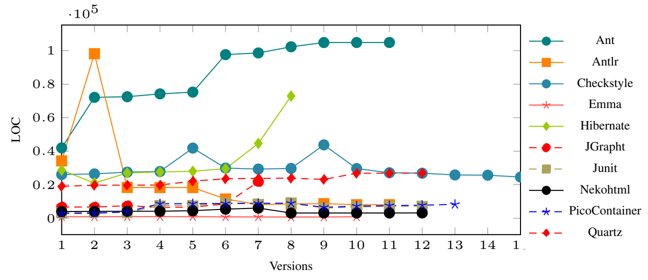
| | Index | | | | | Architectural Smells | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | UD | CD | | HL | | Total |
| Project | ΣASIS | ΣASIS*w | $W$ | $ADI$ | $q(ADI)$ | Pkg | Cl | Pkg | Cl | Pkg | AS |
| ant-1.8.2 | 263.1 | 2353.5 | 564 | 4.173 | 5 | 26 | 730 | 190 | 4 | 6 | 956 |
| antlr-3.4 | 37.7 | 318.5 | 255 | 1.249 | 4 | 8 | 120 | 8 | 4 | 0 | 140 |
| aoi-2.8.1 | 517.8 | 7506.6 | 656 | 11.443 | 5 | 7 | 1007 | 30 | 4 | 2 | 1050 |
| argouml-0.34 | 85.5 | 620.9 | 489 | 1.270 | 4 | 25 | 305 | 114 | 0 | 3 | 447 |
| aspectj-1.6.9 | 1655.3 | 35149.6 | 1476 | 23.814 | 5 | 52 | 2882 | 135 | 6 | 2 | 3077 |
| axion-1.0-M2 | 6.2 | 20.7 | 67 | 0.309 | 2 | 5 | 30 | 23 | 0 | 1 | 59 |
| azureus-4.7.0.2 | 3548.5 | 130507.8 | 4504 | 28.976 | 5 | 168 | 5497 | 1741 | 3 | 3 | 7412 |
| batik-1.7 | 162.3 | 895.8 | 1141 | 0.785 | 3 | 34 | 617 | 92 | 8 | 2 | 753 |
| castor-1.3.3 | 84.0 | 613.4 | 706 | 0.869 | 3 | 58 | 289 | 139 | 1 | 8 | 495 |
| cayenne-3.0.1 | 0.4 | 1.8 | 77 | 0.024 | 1 | 1 | 4 | 1 | 1 | 1 | 8 |
| checkstyle-5.6 | 20.8 | 79.2 | 185 | 0.428 | 2 | 3 | 77 | 6 | 2 | 2 | 90 |
| c_jdbc-2.0.2 | 85.8 | 519.1 | 526 | 0.987 | 3 | 35 | 236 | 129 | 3 | 4 | 407 |
| cobertura-1.9.4.1 | 1.2 | 3.0 | 111 | 0.027 | 1 | 4 | 14 | 5 | 4 | 2 | 29 |
| collections-3.2.1 | 15.0 | 49.0 | 287 | 0.171 | 1 | 5 | 122 | 21 | 4 | 0 | 152 |
| colt-1.2.0 | 56.7 | 306.6 | 241 | 1.272 | 4 | 9 | 173 | 42 | 4 | 2 | 230 |
| columba-1.0 | 62.9 | 727.6 | 459 | 1.585 | 4 | 60 | 163 | 187 | 1 | 4 | 415 |
| compiere-330 | 403.0 | 4992.6 | 926 | 5.392 | 5 | 19 | 922 | 111 | 4 | 5 | 1061 |
| derby-10.9.1.0 | 434.0 | 5252.5 | 1063 | 4.941 | 5 | 51 | 1087 | 153 | 1 | 3 | 1295 |
| displaytag-1.2 | 6.4 | 18.1 | 66 | 0.275 | 2 | 5 | 30 | 13 | 1 | 0 | 49 |
| drawswf-1.2.9 | 25.4 | 119.2 | 160 | 0.745 | 3 | 14 | 87 | 15 | 2 | 0 | 118 |
| drjava-20100913 | 761.5 | 8672.7 | 1818 | 4.770 | 5 | 14 | 1818 | 76 | 9 | 3 | 1920 |
| eclipse_SDK-3.7.1 | 6266.9 | 92813.4 | 18058 | 5.140 | 5 | 608 | 18040 | 2258 | 3 | 6 | 20915 |
| emma-2.0.5312 | 8.9 | 27.5 | 141 | 0.195 | 1 | 8 | 56 | 19 | 3 | 0 | 86 |
| exoportal-v1.0.2 | 18.5 | 55.1 | 239 | 0.230 | 2 | 49 | 107 | 40 | 0 | 0 | 196 |
| findbugs-1.3.9 | 411.0 | 5864.4 | 1106 | 5.302 | 5 | 13 | 909 | 105 | 8 | 2 | 1037 |
| fitjava-1.1 | 3.0 | 9.4 | 69 | 0.136 | 1 | 0 | 12 | 0 | 2 | 0 | 14 |
| fitlibrary-2011 | 109.4 | 1080.5 | 351 | 3.078 | 4 | 38 | 181 | 172 | 1 | 5 | 397 |
| freecol-0.10.3 | 1611.9 | 31660.1 | 1169 | 27.083 | 5 | 20 | 2479 | 124 | 11 | 4 | 2638 |
| freecs-1.3. | 42.7 | 374.4 | 155 | 2.415 | 4 | 6 | 99 | 22 | 4 | 0 | 131 |
| freemind-0.9.0 | 347.3 | 2696.9 | 846 | 3.188 | 4 | 16 | 837 | 82 | 4 | 5 | 944 |
| galleon-2.3.0 | 125.5 | 686.9 | 672 | 1.022 | 3 | 6 | 529 | 26 | 5 | 3 | 569 |
| ganttproject-2.1.1 | 141.0 | 788.1 | 700 | 1.126 | 3 | 20 | 484 | 99 | 2 | 3 | 608 |
| geotools-9.2 | 562.9 | 6518.7 | 2446 | 2.665 | 4 | 206 | 1433 | 808 | 1 | 5 | 2453 |
| hadoop-1.1.2 | 330.0 | 3326.0 | 1743 | 1.908 | 4 | 48 | 1120 | 298 | 4 | 5 | 1475 |
| heritrix-1.14.4 | 49.7 | 354.5 | 369 | 0.961 | 3 | 16 | 137 | 108 | 3 | 3 | 267 |
| hibernate-4.2.0 | 625.6 | 11485.3 | 1408 | 8.157 | 5 | 124 | 1038 | 636 | 0 | 3 | 1801 |
| hsqldb-2.0.0 | 354.4 | 7324.4 | 427 | 17.153 | 5 | 11 | 651 | 45 | 7 | 2 | 716 |
| htmlunit-2.8 | 410.0 | 7226.4 | 613 | 11.789 | 5 | 7 | 765 | 53 | 2 | 0 | 827 |
| informa-0.7.0-a | 1.2 | 2.7 | 113 | 0.024 | 1 | 3 | 19 | 4 | 3 | 0 | 29 |
| iReport-3.7.5 | 767.7 | 8001.8 | 2560 | 3.126 | 4 | 44 | 2316 | 206 | 5 | 4 | 2575 |
| itext-5.0.3 | 183.1 | 2171.1 | 368 | 5.900 | 5 | 6 | 373 | 20 | 5 | 1 | 405 |
| ivatagr.w.-0.11.3 | 0.2 | 2.4 | 132 | 0.018 | 1 | 20 | 2 | 4 | 1 | 2 | 29 |
| jag-6.1 | 40.0 | 105.0 | 185 | 0.567 | 2 | 7 | 158 | 19 | 1 | 0 | 185 |
| james-2.2.0 | 1.8 | 5.3 | 164 | 0.032 | 1 | 6 | 42 | 6 | 1 | 2 | 57 |
| jasperreports-3.7.4 | 130.5 | 982.2 | 730 | 1.345 | 4 | 21 | 362 | 87 | 5 | 2 | 477 |
| javacc-5.0 | 3.8 | 10.4 | 89 | 0.117 | 1 | 2 | 19 | 0 | 3 | 1 | 25 |
| jboss-5.1.0 | 143.5 | 1121.4 | 1291 | 0.869 | 3 | 93 | 639 | 119 | 5 | 4 | 860 |
| jchempaint-3.0.1 | 98.3 | 651.5 | 488 | 1.335 | 4 | 37 | 171 | 215 | 2 | 3 | 428 |
| jedit-4.3.2 | 524.0 | 7554.0 | 922 | 8.193 | 5 | 14 | 1161 | 48 | 7 | 1 | 1231 |
| jena-2.6.3 | 267.1 | 3532.1 | 699 | 5.053 | 5 | 21 | 620 | 116 | 2 | 4 | 763 |
| jext-5.0 | 102.1 | 686.6 | 459 | 1.496 | 4 | 13 | 300 | 24 | 4 | 2 | 343 |
| jfreechart-1.0.13 | 10.3 | 162.6 | 418 | 0.389 | 2 | 15 | 60 | 36 | 5 | 4 | 120 |
| jgraph-5.13.0.0 | 62.3 | 366.7 | 331 | 1.108 | 3 | 8 | 216 | 11 | 4 | 2 | 241 |
| jgraphpad-5.10.0.2 | 11.4 | 43.0 | 242 | 0.178 | 1 | 4 | 80 | 3 | 4 | 0 | 91 |
| jgrapht-0.8.1 | 1.2 | 3.7 | 74 | 0.050 | 1 | 6 | 14 | 11 | 1 | 0 | 32 |
| jgroups-2.10.0 | 50.2 | 268.5 | 506 | 0.531 | 2 | 7 | 269 | 39 | 2 | 2 | 319 |
| jhotdraw-7.5.1 | 34.1 | 130.2 | 683 | 0.191 | 1 | 22 | 325 | 43 | 4 | 4 | 398 |
| jmeter-2.5.1 | 148.9 | 2117.0 | 513 | 4.127 | 4 | 35 | 207 | 274 | 3 | 4 | 523 |
| jmoney-0.4.4 | 27.2 | 60.2 | 148 | 0.407 | 2 | 3 | 138 | 4 | 0 | 0 | 145 |
| joggplayer-1.1.4s | 12.1 | 30.4 | 209 | 0.146 | 1 | 3 | 96 | 2 | 3 | 0 | 104 |
| jparse-0.96 | 9.6 | 43.1 | 58 | 0.743 | 3 | 1 | 30 | 2 | 1 | 0 | 34 |
| jpf-1.5.1 | 0.9 | 2.2 | 72 | 0.030 | 1 | 2 | 18 | 2 | 1 | 0 | 23 |
| jrat-1-beta1 | 19.2 | 100.3 | 176 | 0.570 | 3 | 19 | 69 | 46 | 1 | 2 | 137 |
| jre-1.6.0 | 3136.1 | 81534.4 | 3997 | 20.399 | 5 | 145 | 5761 | 633 | 1 | 4 | 6544 |
| jrefactory-2.9.19 | 129.4 | 1172.0 | 600 | 1.953 | 4 | 37 | 372 | 125 | 1 | 3 | 538 |
| jruby-1.7.3 | 2586.6 | 128738.1 | 1904 | 67.615 | 5 | 46 | 3592 | 324 | 3 | 9 | 3974 |
| jspwiki-2.8.4 | 87.4 | 711.7 | 313 | 2.274 | 4 | 17 | 181 | 78 | 4 | 2 | 282 |
| jsXe-04_beta | 32.8 | 135.0 | 286 | 0.472 | 2 | 7 | 145 | 8 | 6 | 0 | 166 |
| jtopen-7.8 | 196.7 | 1790.9 | 618 | 2.898 | 4 | 3 | 613 | 4 | 1 | 0 | 621 |
| jung-2.0.1 | 6.4 | 18.2 | 141 | 0.129 | 1 | 14 | 61 | 23 | 0 | 2 | 100 |
| junit-4.10 | 6.4 | 20.3 | 122 | 0.167 | 1 | 11 | 45 | 22 | 1 | 1 | 80 |
| log4j-2.0-beta | 15.7 | 80.6 | 156 | 0.516 | 2 | 13 | 64 | 60 | 0 | 3 | 140 |
| lucene-4.2.0 | 204.1 | 1150.6 | 1618 | 0.711 | 3 | 66 | 956 | 241 | 1 | 1 | 1265 |
| marauroa-3.8.1 | 9.5 | 43.5 | 173 | 0.252 | 2 | 12 | 41 | 35 | 4 | 1 | 93 |
| maven-3.0.5 | 19.5 | 205.1 | 180 | 1.139 | 3 | 27 | 28 | 102 | 0 | 6 | 163 |
| megamek-0.35.18 | 551.8 | 7712.7 | 876 | 8.804 | 5 | 20 | 1106 | 72 | 3 | 1 | 1202 |
| mvnforum-1.2.2 | 60.3 | 376.9 | 349 | 1.080 | 3 | 26 | 201 | 56 | 1 | 2 | 286 |
| myfaces-2.1.10 | 56.8 | 554.4 | 806 | 0.688 | 3 | 41 | 243 | 113 | 5 | 2 | 404 |
| nakedobjects-4.0.0 | 111.9 | 758.3 | 766 | 0.990 | 3 | 99 | 369 | 279 | 0 | 2 | 749 |
| nekohtml-1.9.14 | 0.5 | 1.1 | 43 | 0.027 | 1 | 2 | 5 | 2 | 1 | 0 | 10 |
| netbeans-7.3 | 6898.9 | 49950.4 | 31592 | 1.581 | 4 | 1043 | 28865 | 2016 | 3 | 1 | 31928 |
| openjms-0.7.7-b | 13.4 | 39.2 | 227 | 0.173 | 1 | 19 | 115 | 16 | 0 | 2 | 152 |
| oscache-2.3 | 0.9 | 2.3 | 35 | 0.066 | 1 | 4 | 10 | 9 | 0 | 1 | 24 |
| picocont.-2.10.2 | 4.8 | 20.3 | 76 | 0.266 | 2 | 4 | 33 | 11 | 0 | 1 | 49 |
| pmd-4.2.5 | 14.6 | 70.1 | 179 | 0.392 | 2 | 17 | 71 | 34 | 0 | 2 | 124 |
| poi-3.6 | 195.1 | 1709.0 | 864 | 1.978 | 4 | 43 | 416 | 232 | 5 | 8 | 704 |
| pooka-3.0-080505 | 437.7 | 6926.1 | 743 | 9.322 | 5 | 8 | 955 | 28 | 7 | 0 | 998 |
| proguard-4.9 | 12.5 | 43.8 | 148 | 0.296 | 2 | 15 | 85 | 31 | 0 | 2 | 133 |
| quartz-1.8.3 | 16.4 | 38.8 | 122 | 0.318 | 2 | 8 | 63 | 18 | 0 | 1 | 90 |
| quickserver-1.4.7 | 18.1 | 55.6 | 137 | 0.406 | 2 | 6 | 95 | 14 | 0 | 1 | 116 |
| quilt-0.6-a-5 | 4.6 | 13.8 | 31 | 0.447 | 2 | 3 | 25 | 2 | 0 | 0 | 30 |
| roller-5.0.1 | 12.4 | 106.3 | 353 | 0.301 | 2 | 23 | 41 | 65 | 4 | 2 | 135 |
| rssowl-2.0.5 | 834.5 | 10870.3 | 2116 | 5.137 | 5 | 51 | 2127 | 182 | 8 | 2 | 2370 |
| sablecc-3.2 | 1.1 | 4.6 | 73 | 0.062 | 1 | 1 | 18 | 1 | 1 | 0 | 21 |
| sandmark-3.4 | 52.2 | 227.1 | 394 | 0.576 | 3 | 23 | 238 | 22 | 2 | 0 | 285 |
| spring-3.0.5 | 222.2 | 1472.0 | 1453 | 1.013 | 3 | 100 | 741 | 296 | 3 | 4 | 1144 |
| squirrel_sql-3.1.2 | 23.3 | 52.6 | 116 | 0.453 | 2 | 2 | 106 | 2 | 0 | 0 | 110 |
| struts-2.2.1 | 63.7 | 454.1 | 537 | 0.846 | 3 | 43 | 236 | 122 | 1 | 4 | 406 |
| sunflow-0.07.2 | 50.6 | 364.4 | 146 | 2.496 | 4 | 7 | 115 | 38 | 1 | 2 | 163 |
| tapestry-5.1.0.5 | 59.0 | 324.8 | 575 | 0.565 | 2 | 29 | 269 | 84 | 2 | 5 | 389 |
| tomcat-7.0.2 | 107.7 | 646.0 | 796 | 0.812 | 3 | 35 | 452 | 92 | 4 | 1 | 584 |
| trove-2.1.0 | 0.9 | 2.1 | 11 | 0.195 | 2 | 0 | 8 | 0 | 0 | 0 | 8 |
| velocity-1.6.4 | 20.5 | 78.6 | 90 | 0.873 | 3 | 14 | 45 | 34 | 0 | 2 | 95 |
| wct-1.5.2 | 18.9 | 168.9 | 268 | 0.630 | 3 | 35 | 81 | 69 | 1 | 3 | 189 |
| webmail-0.7.10 | 6.2 | 44.1 | 109 | 0.405 | 2 | 7 | 20 | 6 | 3 | 1 | 37 |
| weka-3-6-9 | 297.3 | 2109.3 | 1079 | 1.955 | 4 | 32 | 911 | 236 | 2 | 2 | 1183 |
| xalan-2.7.1 | 423.8 | 4715.5 | 535 | 8.814 | 5 | 21 | 838 | 57 | 2 | 2 | 920 |
| xerces-2.10.0 | 67.0 | 422.4 | 269 | 1.570 | 4 | 12 | 220 | 38 | 1 | 1 | 272 |
| xmojo-5.0.0 | 0.1 | 0.1 | 5 | 0.022 | 1 | 1 | 2 | 0 | 0 | 0 | 3 |

Table III: Projects selected for the ADI evolution evaluation

| Project | Versions | Category | Project | Versions | Category |
|---|---|---|---|---|---|
| Ant | 10 | Tool | JGraphT | 7 | Tool |
| Antrl | 10 | Parser | Junit | 8 | Testing |
| Checkstyle | 15 | IDE | Nekohtml | 12 | Parser |
| Emma | 10 | Testing | PicoContainer | 12 | Middleware |
| Hibernate | 8 | Database | Quartz | 12 | Middleware |



(a) ADI



(b) LOC

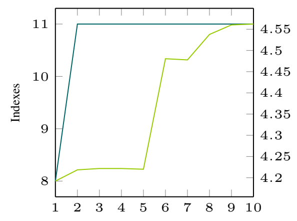
Figure 3: Evolution of ADI and LOC by project

uation of a project and our index. As we can see in Table II, the Eclipse project has a high value(5) of ADI indicating its poor quality. This is consistent with the evaluation of architectural erosion previously conducted by other authors on the same version(3.X) of the Eclipse project [25].

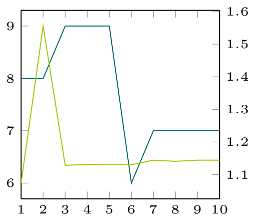
## VI. ADI and SonarQube TDI

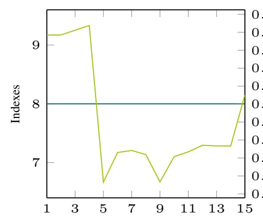
In order to answer RQ3 we have analyzed the possible correlations existing between the Technical Debt Index (TDI) of SonarQube (see Section II) and ADI.

Figure 4 shows the comparison of the two indexes for each project: the y-left-axis is referred to Technical Debt index and the y-right-axis is referred to ADI, but the attention is focused on the trends of the indexes since their values have different scales: TDI and ADI improve (less debt) when they have a decreasing trend. As shown in Figure 4, the trend of TDI is stable in 4 projects on 10, i.e., TDI has the same value in the first and last version; while, ADI has increasing trends for all projects. TDI is not improved in Ant, Junit and PicoContainer, and also ADI is worse. Moreover, ADI and TDI have the same trend in Antrl and Emma projects, while ADI is getting slightly worse and TDI increases in Ant projects. In conclusion, we can assert that the majority of the projects analyzed by using our proposed implementation of ADI has increasing trends for ADI (a worse overall software

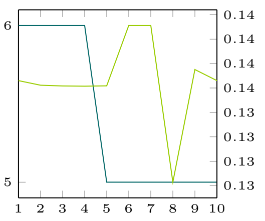


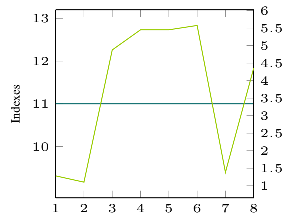
(a) Ant


(b) Antlr

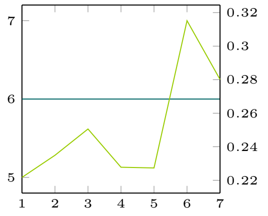


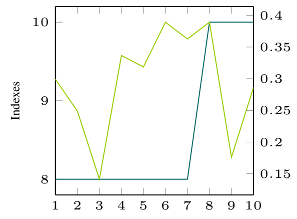
(c) Checkstyle


(d) Emma

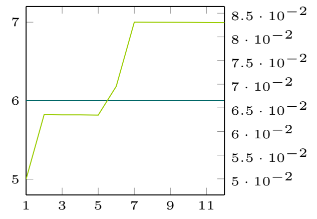


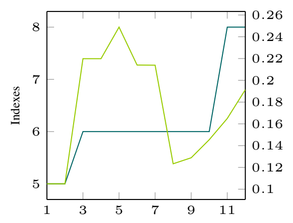
(e) Hibernate


(f) JGrapht

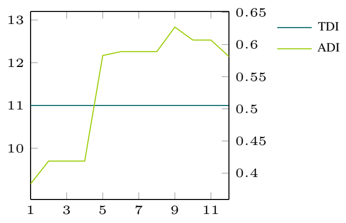


(g) Junit


(h) Nekohtml



(i) PicoContainer



(j) Quartz

Figure 4: Evolution of ADI and TDI Index by project

Table IV: Kendall Correlation and Krippendorff's Alpha inter-rate test both among ADI and TDI (Bugs and LOC)

| | **Kendall Correlation** | | | | | | **Krippendorff's $\alpha$** | | |
|---|---|---|---|---|---|---|---|---|---|
| | **ADI - TDI** | | **ADI - Bugs** | | **ADI - LOC** | | **ADI** | **ADI** | **ADI** |
| Project | Tau | P-value | Tau | P-value | Tau | P-value | **TDI** | **Bugs** | **LOC** |
| Ant | 0.447 | 0.164 | 0.764 | **0.003** | **0.867** | **0.007** | 0.225 | 0.033 | 0.038 |
| Antlr | 0.229 | 0.447 | 0.207 | 0.469 | 0.449 | 0.088 | 0.025 | 0.023 | 0.033 |
| Checkstyle | 0.175 | 0.512 | 0.183 | 0.405 | 0.44 | **0.026** | 0.320 | 0.027 | 0.032 |
| Emma | 0.342 | 0.283 | 0.000 | 1.000 | 0.345 | 0.205 | 0.113 | 0.113 | 0.028 |
| Hibernate | 0.084 | 0.844 | 0.074 | 0.900 | 0 | 1 | 0.311 | 0.037 | 0.057 |
| Jgraph | 0.499 | 0.210 | 0.150 | 0.759 | 0.619 | 0.072 | 0.308 | 0.005 | 0.047 |
| Junit | 0.084 | 0.807 | 0.431 | 0.074 | 0.785 | **0.001** | 0.109 | 0.008 | 0.031 |
| Nekohtml | 0.297 | 0.129 | 0.279 | 0.273 | 0.333 | 0.150 | 0.317 | 0.030 | 0.040 |
| Picocontainer | 0.331 | 0.213 | 0.772 | **0.001** | 0.748 | **0.001** | 0.104 | 0.016 | 0.030 |
| Quartz | 0.245 | 0.215 | 0.381 | 0.109 | 0.473 | **0.039** | 0.324 | 0.014 | 0.035 |

In **bold** are reported P-value lower (or equal) than 0.05 and Tau grater (or equal) than 0.8

architecture quality), while in contrast to ADI, the TDI has a stable or improved trend. This trend difference discovered among ADI and TDI is probably related to their computation: ADI is focused on the evaluation of the AS of the project and AS evolution; TDI is focused essentially on the detection of code and object-oriented violations. Hence, both the two indexes have to be considered in order to identify code and architectural debt.

Moreover, we performed three different tests to better analyze the possible correlation/independence between ADI and TDI index of SonarQube: Kendall [26] correlation test, Cohen Kappa [27] and Krippendorff Alpha [28]. We tested also ADI with some metrics used in the TDI computation of SonarQube: we considered LOC and Bugs.

Kendall correlation results reported in Table IV among ADI and TDI indexes show that there is no correlation among them, since only Ant project has good values of Tau and P-value in ADI - LOC tests.

Krippendorff Alpha results between ADI and TDI indexes, shown in Table IV, outline that there is independence between ADI and TDI, since all the tests had values lower than 0.4 in the ADI - TDI tests: this value is much lower than 0.8, considered as the minimum value for inter-rater reliability. This is confirmed for the other two tests case, i.e., the ADI - Bugs and ADI - LOC.

As last test, we conducted the Cohen's Kappa tests and Table V shows the results among ADI and TDI indexes. The obtained results confirmed what Table IV showed before.

Given the results of the tests on the ADI and TDI indexes and their evolution shown in Figure 4, we can observe that ADI and TDI are behaving in different way, since they seem to not have any correlation or inter-related dependence.

## VII. Threats to Validity

In this Section, we introduce the threats to validity, following the structure suggested by Yin [29] and we debate the different tactics adopted to mitigate them. Threats to *Construct Validity* concerns the identification of the measures adopted for the concepts studied in this work. Regarding this threat, the first issue is related to the detection accuracy of the adopted tools. For this purpose, we relied on an existing detection tool

Table V: Cohen's Kappa correlations test among ADI and TDI

|  | Cohen's Kappa |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  | ADI - TDI |  |  | ADI - Bugs |  |  | ADI - LOC |  |  |
| Project | Kappa | z | P-value | Kappa | z | P-value | Kappa | z | P-value |
| Ant | 0.023 | 1.65 | 0.0986 | 0.042 | 0.942 | 0.346 | 0.052 | 1.25 | 0.211 |
| Antlr | 0.042 | 1.3 | 0.194 | 0.050 | 1.11 | 0.266 | 0.101 | 2.45 | **0.014** |
| Checkstyle | 0 | 0 | **0** | 0.025 | 1.25 | 0.212 | 0.094 | 2.55 | **0.011** |
| Emma | 0.032 | 1.11 | 0.267 | 0 | 0 | 1 | 0.052 | 1.03 | 0.305 |
| Hibernate | 0.150 | 2.15 | 0.504 | 0.011 | 0.245 | 0.807 | 0.010 | 0.202 | 0.84 |
| Jgraph | 0.062 | 1.5 | 0.341 | 0.062 | 1.5 | 0.134 | 0.051 | 1.06 | 0.291 |
| Junit | 0.007 | 0.367 | 0.714 | 0.149 | 2.49 | **0.013** | 0.14 | 3.13 | **0.002** |
| Nekohtml | 0 | 0 | **0** | 0.113 | 2.88 | **0.004** | 0.049 | 1.19 | 0.235 |
| Picocontainer | 0.023 | 1.35 | 0.176 | 0.081 | 2.12 | **0.034** | 0.102 | 2.38 | **0.017** |
| Quartz | 0.074 | 1.67 | 0.081 | 0.084 | 1.74 | 0.081 | 0.091 | 2.27 | **0.023** |

In **bold** are reported P-value lower (or equal) than 0.05 and Kappa grater (or equal) than 0.8

already used in previous research work [12], [22], where the authors report a precision of 100%, since Arcan found only correct instances of architectural smells. The second issue is related to the completeness of the measures we use to characterize architectural debt, i.e. Architectural Smells (AS). We have considered only four AS, while many others have been defined in the literature, hence we have to extend this study and analyze the severity and the index computation and evolution according to a larger set of smells related to different architectural debt indicators. As for numerical issues deriving from the scale of the adopted measures, the ADI formula is defined using the quantile of distribution on a large dataset of projects. This allows mapping unbounded values (most of the times distributed exponentially or as a power law) to values in the range of $[0, 1]$. This kind of normalization has been designed to make the combination of the scores originated by different AS representative of all its parts.

Threats to *Internal Validity* concern factors that could have influenced the obtained results. We cannot claim that our results fully represent every Java project. In order to mitigate this issue, we considered a large set of 109 projects and we analyzed 10 versions of 10 open source projects. This dataset includes projects from different domains, different sizes and with different software architectures. In this way, we mitigate the possibility that one of these factors could influence our results.

Threats to *External Validity* concern the possibility of generalizing our results. To make our results as generalizable as possible, we analyzed a large dataset of projects. More case studies are needed in order to establish whether our observations concerning the evolution of ADI and the independence between ADI and the TDI of SonarQube are applicable also considering other projects, and in particular industrial projects.

Threats to *Reliability* refer to the correctness of the conclusion reached in the study. We applied non-parametric tests and rank-based correlation methods since software metrics often do not have normal distributions. We used a standard R package to perform all statistical analyses since it allows simple replications of them and gives good confidence on the quality of the results.

## VIII. Conclusion and Future Developments

In this paper a new index oriented to the evaluation of architectural issues as AS has been proposed and integrated in the Arcan tool. Severity for all the architectural smells detected by Arcan are given and explained. An evaluation of the index has been performed on a dataset of 109 open source projects. The evolution of the ADI and a comparison with the SonarQube TDI index has been performed on 10 projects considering more than 100 versions in total. In the following, we provide the answers to our research questions.

With respect to RQ1 *How should a new index be formulated to more exhaustively evaluate the architectural debt?* According to our proposal, the index should take in consideration the architectural smells of the project and their development history. Hence the defined index is computed through the values of $ASIS$ (Architectural Smell Impact Score) and $History$. The first element estimates the criticality of the AS in terms of the negative impact that they could have on the project. The second element measures the variation of the number of AS in a project during the development history, as explained in Section IV-B. This index can be used to identify and prioritize the most critical classes or packages in the projects; in this way the developers/maintainers can easily identify and focus their attention on them. Moreover, the index can be used to monitor and check the architectural debt during a project evolution. We assessed that our index reflects the architectural debt of the projects by comparing an evaluation done on Eclipse [25] and the value of ADI, both stating the poor architectural quality of the project.

With respect to RQ2 *How can we estimate the severity of an Architectural Smell?* It is possible to estimate the severity for each architectural smell. We considered both $SeverityScore$ and $PageRank$ indexes, described in Section IV-A. The severity evaluation could be also used to estimate the *cost-solving* of architectural smells. The $PageRank$ has been identified in Section IV-A as a *smell-agnostic* way to weight the severity of an AS, since it indicates the most important and popular place in the dependency graph (i.e., the elements of the dependency graph that are the hardest to refactor). With agnostic, we mean that the $PR$ uses the dependency graph and it is independent from the AS type.

With respect to RQ3 *Is the new index based on architectural smell detection independent from another existing index based on code level issues?* The indexes seem not to have correlation and inter-dependence according to the three tests we conducted in Section V. Moreover, there is no correlation of ADI with both LOC and Bugs used for the computation of the TDI of SonarQube. TDI showed different trends with respect to ADI highlighting its weakness on architectural quality evaluation, since TDI cannot evaluate any aspect of the architecture of a project which is computed by ADI. In fact we observed that ADI got worst and TDI was stable or getting better in the majority of the projects. This highlights the independence between issues at code level (detected by TDI) and at architectural level (detected by ADI), making our index a valid support for developers to assess architectural quality.

According to future developments, we aim to evaluate the index on a large dataset of both open source and industrial projects to get the feedback of the developers [30]. Hence, the weights assigned for example to the History could be changed according to new validations. We focused our effort on the detection of the AS described in Section III, obviously in the future other AS can be considered in the index and detected by Arcan. For example we could consider AS related to Interface design and evolution, such as the AS Ambiguous Interface, Redundant Interface, Overused Interface and Unstable Interface [10]. We would like also to detect different categories of AS which could impact different quality attributes, such as performance and security; in this direction, we could identify and compute different ADI index profiles according to the impact of the AS on specific quality attributes.

Moreover, we aim to extend the index or define a new one to consider also the cost to remove the architectural smells ( *cost-solving*). This could allow developers/maintainers to make a business case (costs vs benefits) and help them to set the order in which they want to remove the AS. Towards the definition of this index oriented to the evaluation of the *cost-solving*, we are also interested to work on the development of some kind of automatic/semiautomatic refactoring support by studying the different refactoring opportunities of each AS. The index now is focused only on the evaluation of the architectural debt derived by AS. Other factors could be considered, e.g. FOSS/COTS obsolescence, legacy monolithic applications and lack of information architecture hardening.

## References

[1] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," in *Proc. of the 2012 Joint Working IEEE/IFIP Conf. on Soft. Arch. (WICSA) and European Conf. on Soft. Arch. (ECSA)*. Finland: IEEE, 2012, pp. 91–100.

[2] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software* , vol. 101, pp. 193–220, 2015.

[3] E. Tom, A. Aurum, and R. T. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013.

[4] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and 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*, ser. ESEC/FSE 2015. New York, NY, USA: ACM, 2015, pp. 50–60. [Online]. Available: http://doi.acm.org/10.1145/2786805.2786848

[5] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, "Identifying and quantifying architectural debt," in *Proceedings of the 38th International Conference on Software Engineering* , ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 488–498. [Online]. Available: http://doi.acm.org/10.1145/2884781.2884822

[6] Z. Li, P. Liang, and P. Avgeriou, "Chapter 9 - architectural debt management in value-oriented architecting," in *Economics-Driven Software Architecture*, I. Mistrik, , R. Bahsoon, , R. Kazman, , and Y. Zhang, Eds. Boston: Morgan Kaufmann, 2014, pp. 183 – 204. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B978012410464800009X

[7] F. Arcelli Fontana, R. Roveda, and M. Zanoni, "Technical debt indexes provided by tools: A preliminary discussion," in *2016 IEEE 8th Inter. Work. on Managing Technical Debt (MTD)*, Oct 2016, pp. 28–31.

[8] T. D. Oyetoyan, J. R. Falleri, J. Dietrich, and K. Jezek, "Circular dependencies and change-proneness: An empirical study," in *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, March 2015, pp. 241–250.

[9] M. Lippert and S. Roock, *Refactoring in Large Software Projects: Performing Complex Restructurings Successfully*. Wiley, Apr. 2006.

[10] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in *CSMR 2009*. Germany: IEEE, 2009, pp. 255–258.

[11] R. Kazman, Y. Cai, R. Mo, Q. Feng, L. Xiao, S. Haziyev, V. Fedak, and A. Shapochka, "A case study in locating the architectural roots of technical debt," in *Proceedings of the 37th IEEE International Conference on Software Engineering (ICSE 2015)*, vol. 2, May 2015, pp. 179–188.

[12] F. Arcelli Fontana, I. Pigazzini, R. Roveda, D. E. Tamburri, M. Zanoni, and E. D. Nitto, "Arcan: a tool for architectural smells detection," in *To Appear at IEEE International Conference on Software Architecture (ICSA 2017)*, 2017.

[13] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble, "The qualitas corpus: A curated collection of java code for empirical studies," in *Proc. 17th Asia Pacific Software Engineering Conference (APSEC 2010)*. Sydney, Australia: IEEE, December 2010, pp. 336–345.

[14] D. Coleman, D. Ash, B. Lowther, and P. Oman, "Using metrics to evaluate software system maintainability," *Computer*, vol. 27, no. 8, pp. 44–49, Aug. 1994. [Online]. Available: http://dx.doi.org/10.1109/2.303623

[15] J.-L. Letouzey, "The SQALE method for evaluating technical debt," in *MTD 2012*, June 2012, pp. 31–36.

[16] S. M. A. Shah, J. Dietrich, and C. McCartin, "Making smart moves to untangle programs," in *2012 16th European Conference on Software Maintenance and Reengineering*, March 2012, pp. 359–364.

[17] R. C. Martin, "Object oriented design quality metrics: An analysis of dependencies," *ROAD*, vol. 2, no. 3, Sept–Oct 1995.

[18] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development*, vol. 56, no. 5, pp. 9:1–9:13, 2012.

[19] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for Software Design Smells*, 1st ed. Morgan Kaufmann, 2015.

[20] E. Kouroshfar, M. Mirakhorli, H. Bagheri, L. Xiao, S. Malek, and Y. Cai, "A study on the role of software architecture in the evolution and quality of software," in *Proc. 12th Working Conf. Mining Software Repositories*, ser. MSR '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 246–257.

[21] R. Mo, Y. Cai, R. Kazman, and L. Xiao, "Hotspot patterns: The formal definition and automatic detection of architecture smells," in *12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, Montreal, QC, Canada, May 4-8, 2015*, 2015, pp. 51–60. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2015.12

[22] F. Arcelli Fontana, I. Pigazzini, R. Roveda, and M. Zanoni, "Automatic detection of instability architectural smells," in *Proc. of the 32nd Intern. Conf. on Software Maintenance and Evolution (ICSME 2016)*. Raleigh, North Carolina, USA: IEEE, Oct. 2016, eRA Track.

[23] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," in *Seventh International World-Wide Web Conference (WWW 1998)*, 1998. [Online]. Available: http://ilpubs.stanford.edu:8090/361/

[24] R. Terra, L. F. Miranda, M. T. Valente, and R. S. Bigonha, "Qualitas.class Corpus: A compiled version of the Qualitas Corpus," *Software Engineering Notes*, vol. 38, no. 5, pp. 1–4, 2013.

[25] B. Merkle, "Stop the software architecture erosion: Building better software systems," in *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*, ser. OOPSLA '10. ACM, 2010, pp. 129–138. [Online]. Available: http://doi.acm.org/10.1145/1869542.1869563

[26] W. Hoeffding, *Econometrica*, vol. 25, no. 1, pp. 181–183, 1957. [Online]. Available: http://www.jstor.org/stable/1907752

[27] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit." *Psychological bulletin*, vol. 70, no. 4, p. 213, 1968.

[28] K. Krippendorff, *Content analysis: An introduction to its methodology*. Sage, 2004.

[29] R. K. Yin, *Case Study Research: Design and Methods, 4th Edition (Applied Social Research Methods, Vol. 5)*, 4th ed. SAGE Publications, Inc, 2009.

[30] A. Martini, F. Arcelli Fontana, A. Biaggi, and 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)*. Madrid, Spain: Springer, Sep. 2018.