Sen4Smells: A Tool for Ranking Sensitive Smells for an Architecture Debt Index

J. Andres Diaz-Pace*, Antonela Tommasel*, Ilaria Pigazzini‡ and Francesca Arcelli Fontana‡
*ISISTAN, CONICET, UNICEN University, Argentina {andres.diazpace,antonela.tommasel}@isistan.unicen.edu.ar,
‡Department of Informatics, Systems and Communication, Milano-Bicocca University {ilaria.pigazzini,francesca.arcelli}@unimib.it

***Abstract*—Technical debt indexes are metrics for assessing the quality of a software system. Both academic and commercial tools have begun to provide computations of such indexes based on design violations and smells (e.g., cycles among system elements). When computing a debt index for a given project, a common use case is that engineers look at the index values for spotting design issues that negatively affect system evolution and quality. In this context, those smells being critical for the system architecture should be promptly identified so as to evaluate proper remediation actions. However, the interpretation of an index value in terms of problematic smells is usually a manual and labor-intensive task for engineers. To help with this task, we propose a tool called *Sen4Smells* that performs an automated sensitivity analysis for a given debt index based on the evolution of both the index values and the corresponding smells across (past) system versions. The *Sen4Smells* output is a ranking of smells that, due to their variations or instability, are major contributors to the debt index, and thus, can impact on architecture quality. *Sen4Smells* is designed as a pipeline that combines information from existing tools for smell detection, predefined debt index formulas, and the Sobol method for sensitivity analysis. As a demonstration of the tool functionality, we briefly present implementations for the Arcan and Sonargraph tools with their respective debt indexes.**

***Index Terms*—tool support, architectural smells, debt index, sensitivity analysis, system evolution**

I. Introduction

The quality of a software system can be evaluated by considering the technical debt accumulated in the system [9]. To this end, several tools support the computation of debt indexes, such as: Sonargraph, CAST, Arcan, and SonarQube, among others [1]. Having a *debt index* (DI) is useful to provide engineers with a quality indicator of the overall system health. Yet more importantly, a DI should also assist engineers to identify sub-optimal parts of the system that could be improved (e.g., via refactoring). Recent works [2, 23] have argued for the need of taking architectural issues into account in debt indexes, and focused on the analysis of *architectural smells* [7]. In this view, an architectural smell (AS) indicates a violation of key design principles or decisions that might lead to high maintenance and evolution costs [5]. Thus, smells are regarded as an important source of technical debt.

In practice, once engineers have chosen a DI and applied it to their project, a relevant aspect for them is the "interpretation" of the index values [15]. By interpretation, we refer to the ability of examining the index values in order to spot those system elements that are the main *contributors* to the current design problems. This task involves looking at system elements with different *granularity* (e.g., packages, classes, or smells), and also considering the history of a system element (e.g., the evolution of a smell across a range of system versions). Common questions posed by engineers include: (i) which packages are the most sensitivity ones for the current architecture health?, or (ii) which smells have suffered instabilities in the past system versions that might compromise the design in future versions? As a result, the ASs or packages being sensitive for the system architecture should be brought to the developer's attention, due to their impact on system evolution. Unfortunately, the examination of DI values in terms of ASs or packages is often a cumbersome and time-consuming task for engineers. In addition, the instabilities in ASs cannot be simply detected by means of a static analysis tool that identifies the smells in the current system version (e.g., JDeodorant, or SonarQube); rather, the evolution history of those ASs need to be taken into account.

In this context, we propose a tool called *Sen4Smells* that performs an automated sensitivity analysis (SA) for a collection of system values provided by a predetermined DI. These values are linked to system elements, such as ASs or packages. Our approach relies on two building blocks: (i) the adaptation of an existing SA method to DIs based on ASs, and (ii) a strategy for decomposing the DI of choice according to different levels. At the lowest level, we leverage on ASs and DI metrics for those smells (e.g., number of incoming dependencies of a smell). The goal is to assess how DI variations can be attributed to variations in metrics of system elements [3]. To do so, the tool performs a screening of the various system elements affecting the index over time, and returns a ranking with the most sensitive ones to the tool user. The inputs for this analysis are: a list of previous system versions, the formula for computing a particular DI, and the desired granularity of system elements. *Sen4Smells* is designed as a pipeline, in which existing modules for detecting AS and computing metrics from the software system can be wrapped, and configured based on the selected DI.

The main contribution of this tool is the assistance for engineers to interpret system-level DI values in terms of problematic ASs and packages, as *indicators* of system quality trends. We are not developing a new SA technique, but rather selecting a suitable one and adapting it to our index interpretation problem. To evaluate the *Sen4Smells* functionality, we have

978-1-7281-5957-7/20/$31.00 ©2020 IEEE

instantiated the pipeline with two indexes: ADI (Architecture Debt Index) and SDI (Structural Debt Index). The first index is supported by the Arcan academic tool [17], which also provides a static analyzer for detecting three AS types in Java systems. The second index is provided by the Sonargraph commercial tool[1], which computes cycles and metrics on Java systems. The SA is currently implemented via the well-known Sobol method [18]. Other DI formulas, smell analyzers, or alternative SA methods, can be easily integrated into the pipeline. The rest of the paper is structured as follows. Section 2 analyzes related work. In Section 3, we describe both the design of the tool pipeline and details of how the SA is performed. Section 4 presents the integration of *Sen4Smells* with Arcan and Sonargraph, and also reports on preliminary results of applying the tool to three systems. Finally, Section 5 presents the conclusions and outlines future work.

II. Related Work

Several works in the literature have discussed the management of TD at different levels (e.g., architecture, design, code, test, social, documentation and technology) [13, 16, 21]. In this regard, several quality or DIs have also been proposed, which are briefly described below.

SonarQube[2] computes a Technical Debt Index based on estimations of the time needed for fixing the violations to code and design rules found by the tool. It proposes the SQALE [12] model for estimating technical debt by considering the remediation cost of each issue and the ratio between the remediation cost of the issues and the cost of starting over. CAST [4][3] estimates technical debt as the cost of remediating violations of good architectural or coding practices in production code based on the detection of structural problems. Structure101[4] index shows a Structural over-Complexity view to estimate the proportion of the system that is affected by architectural issues. Particularly, the complexity and proportion of the system involved in tangles are taken into account.

Roveda et al. [17] defined the ADI based on the detection of ASs by means of the Arcan tool. Analogously, Wu et al. [23] defined a Standard Architecture Index including structure, class/functionality and global measures referring to both source code and software models, which reflect on recurring architectural problems (or smells). Verdecchia et al. [22] proposed a step-by-step method to build architectural debt indexes based on architectural violations that can be identified through static analysis rules. To validate their approach, they implemented an index prototype that considered a selection of the most architecturally-related SonarQube rules.

Despite the increasing interest on the definition of DIs and how to compute them, the identification of critical ASs and design issues has received comparatively less attention. In this regard, Shahbazian et al. [19] aimed at preventing the adverse effects of architectural decay by automatically detecting architecturally-significant issues based on their textual information and their recovered architecture. On this basis, a predictive model based on machine learning techniques was developed to predict the architecture significance of a newly-submitted issue.

[1]https://www.hello2morrow.com/products/sonargraph
[2]https://www.sonarqube.org
[3]https://www.castsoftware.com
[4]https://structure101.com/

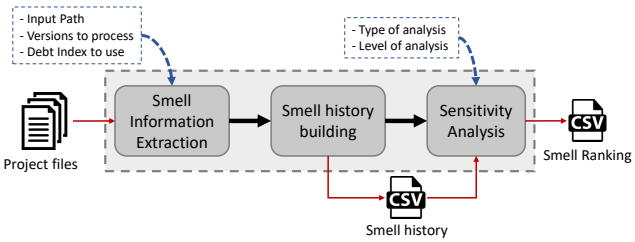


Figure 1. Main processing stages and parameters of *Sen4Smells*

III. Tool Architecture & Background

Sen4Smells is designed as a 3-stage pipeline architecture, as shown in Figure 1. A prototype and examples are publicly available in GitHub[5].

The first stage, called *Smell Information Extraction*, processes a sequence of Java system versions for the project under analysis, in order to detect instances of ASs and to compute metrics associated to them. The system versions (project files) to analyze are provided as inputs by the tool users. We assume they are looking for design problems at the current version (i.e., the latest provided version). The version processing is normally based on a static code analysis, which is delegated to wrappers of existing tools (e.g., Arcan, Sonargraph) based on the DI selected by the user. In our context, a DI is based on predefined AS types and metrics. For example, Sonargraph Structural Debt Index (SDI) is a cumulative function of the cycles (i.e., the smell type) and the number of dependencies to be removed (e.g., a metric) to break those cycles.

To analyze the trends in the evolution of ASs (or AS aggregations), the second stage, called *Smell History Building*, creates the evolution history of the different ASs across the range of versions. This evolution history can be seen as a matrix [10], in which each column represents a version at time t and each row represents a smell instance. The cells of a given row keep the values of a smell metric across versions. This way, we can trace "paths" of smell variations (for a metric of interest) over time. The metrics (or scores) to consider for a given DI are configured by the user. Note that the user's focus is on ASs appearing in the current version (i.e., the last known version), whose behavior can be explained through the history of the smell metrics in past versions. Once computed, the evolution history of each AS is stored in a `CSV` file to be processed by the SA. Table I shows the evolution of scores for a subset of ASs (left-most column) detected in 8 versions of Apache OpenJPA. The codes cd, ud and hl correspond to instances of three types of ASs used in our work, namely: *Cyclic Dependency*, *Unstable Dependency*, and *Hub-like Dependency* [14, 20], respectively.

[5]https://github.com/tommantonela/Sen4Smells

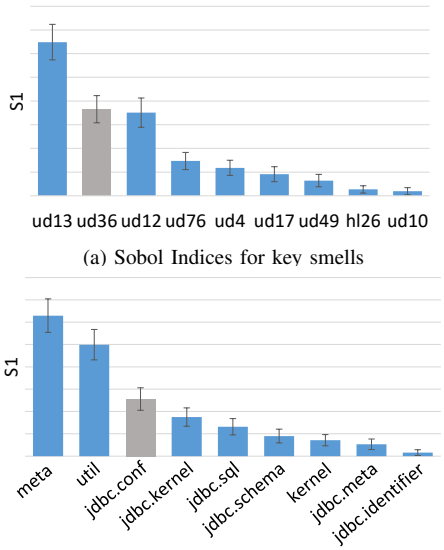
	openjpa 1.0.0	openjpa 1.1.0	openjpa 1.2.0	openjpa 2.0.0	openjpa 2.1.0	openjpa 2.2.0	openjpa 2.3.0	openjpa 2.4.1
ud13	1.85	4.1	2.05	4.5	2.25	2.3	2.3	2.25
ud36	1.89	2.28	2.56	1.89	1.62	1.35	1.62	0.6
ud12	1.8	2.05	2.1	2.25	4.7	4.8	2.4	2.35
ud76	0.6	1.7	1.7	2.43	1.8	2.43	1.8	2.43
ud4	0.81	0.81	1.26	1.26	1.8	1.8	1.8	2.43
ud17	0.68	0.68	0.36	1.66	1.57	1.42	1.42	1.66
ud49	0.8	0.8	0.8	1	1.53	1.53	1.53	1.53
hl26	0.35	0.5	0.5	0.45	0.5	0.55	1.2	1.2
ud10	0.48	0.13	0.13	0.21	0.28	0.21	0.21	0.28
cd34	-	-	-	-	1.05	0.9	0.9	1.05
cd35	-	0.27	0.2	0.27	0.27	0.3	0.3	0.3
cd48	1.26	1.26	1.26	1.59	1.59	1.59	1.59	1.59
ud7	-	-	-	0.26	0.26	-	-	0.26

Table I
EVOLUTION OF SCORES FOR SMELLS ACROSS DIFFERENT OPENJPA VERSIONS

At last, the third stage is the *Sensitivity Analysis*, which takes as input the smell evolution paths and the chosen DI to report a ranking of smells (or other system elements) to the user. Given a DI formulation (e.g., the ADI definition [17]), we see it as a black-box model that relates inputs (the values of the ASs metrics) to a numeric output (the index value for each system version). In this model, the relations between inputs and output are generally not direct, as the model might be non-linear and change over time. This model is exercised with the Sobol method, which is a global SA method for measuring the contribution of the inputs to the output variance (other SA methods could also be applied). As a result, a sensitivity value (also known as Sobol index) is assigned to each AS. The higher the Sobol index for a smell, the higher the chance that variations in the DI are due to that smell. Thus, we interpret the Sobol index of a smell as the sensitiveness of the system design portion affected by that smell with respect to the DI, which is a proxy for the overall system quality. The SA can be performed at the level of AS (which is a fine-grained level), but also at other granularity levels. For instance, the user can be interested in grouping the ASs according to the system packages or to smell types. In the former case, the smell features are grouped per top-level package, and the SA returns a ranking of critical packages. In terms of the tool concepts, this means that the DI can be decomposed using different criteria, and then the SA is accordingly adjusted. For example, Figure 2 shows a report of key ASs and key packages for OpenJPA (X axes) using the Sobol method.

A. Architectural Smells and Debt Indexes

Architectural smells can be defined as a combination of architectural constructs that often indicate modifiability problems in the system [7]. An AS usually comes from a poorly-understood or sub-optimal design decision. Different ASs have been catalogued in the literature. In *Sen4Smells*, various types of ASs can be processed, based on the underlying detection tools available in the *Smell Information Extraction* module. As mentioned in previous sections, we currently support the following dependency-based smells: *Cyclic Dependency (cd)*, *Unstable Dependency (ud)*, and *Hub-like Dependency (hl)* [14, 20]. A *cd* refers to a set of packages being involved in a chain of (usage) relations that breaks the desirable acyclic



(a) Sobol Indices for key smells

(b) Sobol Indices for key packages

Figure 2. Results of sensitivity analysis for OpenJPA based on ADI

nature of a system dependency structure. A ud describes a package that depends on other packages that are less stable than itself. A hl arises when a package has outgoing and incoming dependencies with many other packages. These ASs have been recently connected to architecture degradation issues (e.g., in the form dependency violations) [8].

For the purpose of the tool, the target DI is seen as a function of the ASs that are present in the system. This perspective has also been adopted in recent index formulations[6] [17, 23]. For instance, the ADI definition relies on two metrics (*pagerank* and *severity*) that characterize each AS [17]. Furthermore, a DI is *sensitive* to changes in metric values for certain ASs. For instance, an AS might have dependencies being added or cycles being enlarged from one version to another, which impact on the index values.

B. Decomposition of the Index Score

For analyzing DI values at the current system version based on the previous ones, we perform a decomposition of the (global) index formula into its constituent parts. These parts (or elements) can refer to different granularity levels, namely: (i) the ASs themselves (bottom level), or (ii) groups of ASs using a predefined criterion. For example, we might group the ASs based on either their type or the package structure of the system. At each level, the corresponding elements are characterized by the metrics of the index formula, which altogether provide index values for the elements. This decomposition strategy is sketched in Figure 3 for the ADI, showing two levels of AS aggregations (packages and AS types).

From a temporal perspective, the decomposition strategy looks at the index values over a sequence of system versions. This way, we can analyze trends in the evolution of ASs, or in the evolution of AS aggregations. Departing from the index

[6]https://blog.hello2morrow.com/2018/12/a-promising-new-metric-to-track-maintainability/

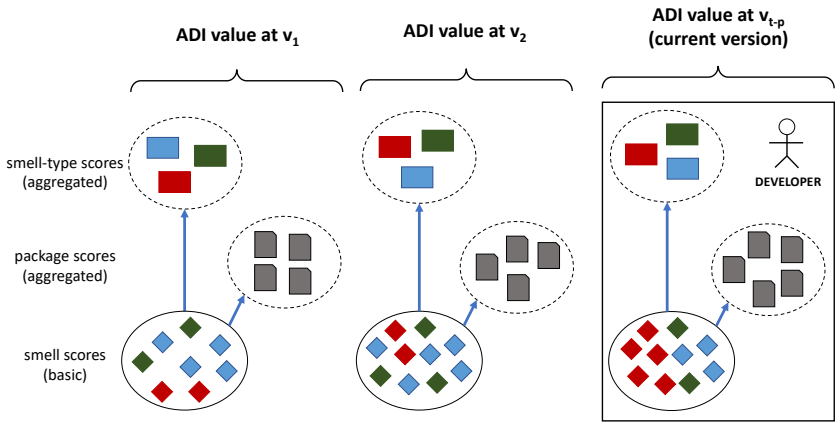


Figure 3. Decomposing a debt index (e.g., the ADI) at different levels of granularity, and looking at previous versions

decomposition, the elements at each level are seen as variables whose values follow a particular distribution over time. These distributions are fed into the SA for interpreting the DI.

C. Sensitivity Analysis

SA techniques study how the variation in the output of a model can be apportioned among model inputs [18]. In our problem, the input variables are the scores (at a given decomposition level), while the model output is the global DI value. If a change in a variable results in a relatively large change in the DI, then we say that the DI is sensitive to that variable. From this analysis, a ranking of key variables (e.g., ASs, packages, or smell types) can be obtained.

The *Sobol* method [18] is a global SA technique, which allocates the output variability to the variability of the inputs, taking into account all the variables and interactions among them. The results of this method are the so-called *Sobol indices* for the input variables. The higher the value of a sensitivity index, the more influential the respective variable is for the model. First-order indices ($S1$) reflect the main effect and measure the fractional contribution of a single variable to the output variance. Total-order indices (ST) take into account the main, second-order and also higher-order effects of variables on the output variance. Figure 2a shows the $S1$ values for the top-10 ASs for the ADI. For instance, an $S1$ of 0.18 was obtained for `ud36`, which indicates that this smell is very likely to influence the ADI. A plausible developer's interpretation of the situation is that a few ud smells are more problematic (in terms of increased dependencies) than a large group of cd smells (if their cyclic dependencies remain stable). If we instead analyze package `org.apache.openjpa.conf` in Figure 2b, an $S1$ value of 0.12 reflects an influential role of (the ASs in) the package for the ADI. The analysis reveals other influential packages as well, such as: `org.apache.openjpa.meta` and `org.apache.openjpa.util`.

The transformation of DI metrics into SA variables involves mainly three tasks: (i) specifying the decomposition level for the variables (e.g., smells, packages), (ii) sampling values from the evolution history of those variables, and (iii) computing the Sobol indices based on that sample. For sampling the values of the variables, we rely on their distribution in a range of system versions. For cd smells, the distribution often shows cycles becoming larger (or smaller) over time; while for ud and hl smells, the variations are due to an increase (or decrease) in the number of dependencies to the packages affected by the ASs. When ASs are aggregated in packages or types, an analogous distribution can be derived.

IV. Case-studies

Currently, *Sem4Smells* works with the ADI (Arcan) and the SDI (Sonargraph) indexes. Each DI needs an integration with the corresponding tool, which is adapted and parameterized in the pipeline. These integrations are briefly explained below. Afterwards, we describe a number of experiments performed with the tool for ranking ASs.

A. Arcan & the Architecture Debt Index

The *Smell Information Extraction* component takes the `JAR` files of the system versions, and passes them on to a wrapper for the Arcan engine. When invoking Arcan on a given version, it generates a `version` object that contains all the detected ASs and their ADI features. To ensure extensibility, this version structure is independent of the particular DI. Then, the *Smell History Building* component merges all versions into a smell evolution matrix, which is saved to a standard `CSV` file.

The *Sensitivity Analysis* component reads from the matrix to run the Sobol analysis. The desired granularity level for the analysis is configured in this component, which triggers the creation of AS groups as Sobol variables. This functionality is general for any index or AS type. In addition, the ADI formula that maps the variables to index values needs to be set. For ADI, this formula was implemented via a general class of the component. A `CSV` file for each ranking is finally computed as output.

B. Sonargraph & the Structural Debt Index

Unlike Arcan, each system version needs to be initially processed by Sonargraph, and then exported as an `XML` report

System	Description	Version	HL	UD	CD
Apache Camel	Framework for message-oriented middleware with a rule-based routing and mediation engine that provides a Java object-based implementation of the Enterprise Integration Patterns using an application programming interface to configure routing and mediation.	apache-camel-2.2.0	3	10	48
		apache-camel-2.3.0	3	10	49
		apache-camel-2.4.0	3	10	49
		apache-camel-2.5.0	3	10	49
		apache-camel-2.6.0	3	10	49
		apache-camel-2.7.0	3	6	26
		apache-camel-2.8.0	3	11	78
		apache-camel-2.9.0	3	11	90
		apache-camel-2.10.0	3	11	92
		apache-camel-2.11.0	3	11	97
		apache-camel-2.12.0	3	11	97
		apache-camel-2.13.0	3	11	98
		apache-camel-2.14.0	3	11	98
		apache-camel-2.15.0	3	11	98
Apache Cxf	It is a relational database management system that can be embedded in Java programs and used for online transaction processing.	apache-cxf-2.0.6	4	52	110
		apache-cxf-2.1.1	4	61	132
		apache-cxf-2.2.1	3	74	150
		apache-cxf-2.3.0	3	79	197
		apache-cxf-2.4.0	4	86	199
		apache-cxf-2.5.0	3	88	206
		apache-cxf-2.6.0	3	92	146
		apache-cxf-2.7.0	3	95	196
Hibernate	It is an object-relational mapping tool for the Java programming language that provides a framework for mapping an object-oriented domain model to a relational database.	hibernate-core-4.0.0.Final	5	30	153
		hibernate-core-4.1.0.Final	5	33	163
		hibernate-core-4.2.0.Final	6	32	171
		hibernate-core-4.3.0.Final	6	62	230

Table II
RANGES OF ANALYZED VERSIONS AND DESCRIPTION OF THE THREE SYSTEMS

file. This file contains information about cycles and SDI features. In the *Smell Information Extraction* component, we rely on a Sonargraph API[7] for accessing each `XML` file as a version object. The process continues with the generation of the smell evolution file by the *Smell History Building* component. Finally, the *Sensitivity Analysis* component runs the Sobol method as in the Arcan case, except that the SDI formula is configured as the index to be used.

C. Preliminary Evaluation

As an initial assessment of the applicability of the proposed approach, we analyzed three open-source Java systems, namely: Apache Camel, Apache Cxf, and Hibernate, which have already been used in other smell-based analyses [6, 11, 19]. These systems comprise several versions, including major and minor releases. Using open-source projects is a common practice in order to calibrate metrics and indexes. Apache was chosen as it is one of the largest open-source organizations that produces well-maintained systems. In addition, we included a non-Apache system in order to keep our analysis general. We considered a predefined range of minor versions in between two major versions per system, as summarized in Table II. The number of AS instances per type are also listed. When selecting the ranges of versions, we tried to ensure that there were enough ASs of different types in the versions, and also that these ASs had some degree of variability over time. We considered the ADI as the target DI, and the Arcan tool was used for detecting the AS instances and computing the ADI values.

Our goal was to investigate whether the key ASs resulting from the SA exhibited correspondences with design issues (or problems) in the analyzed systems.

The analysis was only performed for individual ASs. The rankings of smells produced by the Sobol method for the 3 systems are shown in Table III. We found particular AS instances with high Sobol indices, which were unnoticed when simply computing their ADI scores or looking at the ADI contributions per smell type. For Hibernate and Cxf, an interesting observation was that many key ASs corresponded to ud and hl smells, rather than to cd smells. These indicators somehow differ from the common belief about the sole importance of cd smells [15], which is also shared by other debt indexes. The results seem to implicate that a DI should also incorporate hl and ud smells in their computations.

In order to understand whether the sensitive elements of the SA rankings were related to actual design issues, we inspected the publicly available information about the systems. In particular, we took information from the Jira platform[8], which allows to perform queries on the issues of a given project. An issue is a message regarding a bug, a task or another project issue. We considered three types of Jira issues for packages, namely: *bug*, *improvement* and *new feature*. Our hypothesis was that a sensitive AS should manifest as "some work to do" with respect to one or more packages in the system, either in the form of functional changes (e.g. new feature) or maintenance tasks (e.g., bug, improvement). If so, we can tag the affected packages as being part of a design problem. Hence, for each package affected by at least one key AS, we counted the number of Jira issues in which the package name appeared. Then, each ud and hl smell was assigned to the issue count of the corresponding package. For cd smells, since they affect several packages, instead of using the issue count, we computed the mean of the number of issues affecting the packages of the cycle.

On this basis, we decided whether a smell s_i (from an SA ranking) correlates with a design problem at package p_j (assuming that s_i affects p_j) using the following rule: *if the issue count for p_j exceeds a given threshold, then a correlation between s_i and p_j regarding a design problem is asserted*. We assigned a different threshold to each project under analysis. The threshold was computed as the median of the distribution of Jira issues of the project. In this way, the Camel threshold was set to 7.76, the Cxf threshold was set to 4, and the Hibernate threshold was set to 1. According to these settings, the percentages of key ASs that were actually linked to design problems by our rule were: 55% for Apache Camel, 55% for Apache Cxf, and 80% for Hibernate.

Overall, we were able to validate 50% or more ASs reported by the Sobol method for the three systems. In addition, the analysis of specific ASs showed correlations between the ASs and certain design issues.

V. CONCLUSIONS

In this paper, we have described the *Sen4Smells* tool that performs a sensitivity analysis of a debt index based on the architectural smells involved in the debt index formula. A

[7]https://github.com/sonargraph/sonargraph-integration-access

[8]https://www.atlassian.com/software/jira

Apache Camel				Apache Cxf				Hibernate			
Smell	**I**	**DP?**	**Affected Packages**	**Smell**	**I**	**DP?**	**Affected Packages**	**Smell**	**I**	**DP?**	**Affected Packages**
hl203	17.00	yes	∗.impl	ud56	8.00	yes	⋆.interceptor	ud569	1.00	yes	○.persister. entity
ud925	9.00	yes	∗.util	ud124	3.00	no	⋆.endpoint	ud304	3.00	yes	○.mapping
hl104	66.00	yes	∗.processor	ud162	6.00	yes	⋆.transport	ud1092	61.00	yes	○
ud447	26.00	yes	∗.model	ud120	2.00	no	⋆.ws.policy	hl863	2.00	yes	○.cfg
cd173	7.36	no	many	ud301	250.00	yes	⋆	ud959	0.00	no	○.pretty
cd596	8.36	yes	many	ud842	2.00	no	⋆.frontend	ud1440	0.00	no	○.dialect. function
cd612	7.90	yes	many	ud707	7.00	yes	⋆.jaxrs.utils	hl1093	1.00	yes	○.engine
cd835	7.78	yes	many	ud777	4.00	yes	⋆.binding.soap	hl470	10.00	yes	○.type
cd776	6.22	no	many	ud697	3.00	no	⋆.tools.common	ud1545	1.00	yes	○.loader
hl759	9.00	yes	∗.builder	ud4	1.00	no	⋆.tools.common. model	ud1371	4.00	yes	○.dialect
ud659	5.00	no	∗.model. language	ud380	0.00	no	⋆.jaxws.support	ud1571	1.00	yes	○.proxy
cd808	7.75	yes	many	cd392	4.30	yes	many	ud706	1.00	yes	○.event
cd627	6.89	no	many	cd299	3.10	no	many	ud1258	1.00	yes	○.criterion
cd510	6.90	no	many	cd121	4.10	yes	many	cd1200	1.57	yes	many
cd177	6.63	no	many	cd824	4.38	yes	many	ud554	1.00	yes	○.exception
cd630	6.63	no	many	cd809	4.50	yes	many	cd950	1.57	yes	many
cd301	8.71	yes	many	cd597	4.00	yes	many	cd1707	1.30	yes	many
cd861	7.90	yes	many	cd479	4.78	yes	many	ud818	1.00	yes	○.impl
cd377	6.30	no	many	cd752	3.80	no	many	cd109	1.78	yes	many
cd567	7.29	no	many	ud306	0.00	no	⋆.tools.java2wsdl. processor.internal	ud801	0.00	no	○.hql.ast.util

Legend: **I**=#Issues, **DP?**=Design Problem? ∗=org.apache.camel, ⋆=org.apache.cxf, ○=org.hibernate

Table III
RANKING OF TOP-20 KEY SMELLS FOR THE ANALYZED SYSTEMS

direct benefit of this analysis is that makes a DI actionable for engineers, by enabling the identification of key ASs and problematic packages.

The initial results of applying the tool for ADI and computing AS rankings have been promising. Nonetheless, some threats to validity should be mentioned. A first threat is our reliance on the Arcan tool for detecting the ASs in the system versions. Other tools, such as Sonargraph, could have detected different smell instances, and thus, led to a different evolutionary history. A second threat (for the evaluation) is the ADI formula, as the SA assesses the variability of the AS scores in terms of that formula. The ADI computation depends on the AS type. For hl and ud smells, whose topology is based on incident dependencies to a given package, their ADI scores can be affected by the addition of dependencies from a version to another. For the cd smells, in turn, the variations in topology are due to cycles getting bigger (or smaller, or even disappearing) over time. If SDI were to be used, its formula is only based on cd smells. Regarding external validity, the results of the evaluation were based on a manual analysis of the system repositories by the authors. This analysis was confined to predefined ranges of versions. The selection of different ranges of versions could affect the outputs of the SA methods. Furthermore, the rankings of ASs should be also validated for design issues by human developers.

Regarding the Sobol method for the SA, we found it useful because it makes no assumptions about the index formulations. However, in projects with a large number of smells, we observed that the computational efforts required by Sobol might increase rapidly with the number of variables. In such cases, more efficient methods should be explored.

As future work, we plan to integrate the tool pipeline within a build process of a project, and including support for creating visualizations and reports based on the SA rankings. As regards the experimental work, we need to further validate whether the rankings computed by *Sen4Smells* are correlated with critical parts of the analyzed systems. Finally, we plan to extend *Sen4Smells* with more indexes and smell types.

REFERENCES

[1] F. Arcelli Fontana, R. Roveda, and M. Zanoni. Technical debt indexes provided by tools: A preliminary discussion. In *2016 IEEE 8th MTD*, pages 28–31, Oct 2016.

[2] F. Arcelli Fontana, I. Pigazzini, R. Roveda, D. E. Tamburri, M. Zanoni, and E. D. Nitto. Arcan: a tool for architectural smells detection. In *IEEE ICSA 2017*, 2017.

[3] H. Christopher Frey and S. R. Patil. Identification and review of sensitivity analysis methods. *Risk Analysis*, 22 (3):553–578.

[4] B. Curtis, J. Sappidi, and A. Szynkarski. Estimating the size, cost, and types of technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)*, pages 49–53, 2012.

[5] N. Ernst, S. Bellomo, I. Ozkaya, R. Nord, and I. Gorton. Measure it? manage it? ignore it? software practitioners and technical debt. In *2015 10th ESEC/FSE*.

[6] F. A. Fontana, I. Pigazzini, R. Roveda, and M. Zanoni. Automatic detection of instability architectural smells. In *2016 IEEE ICSME*, pages 433–437.

[7] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic. Identifying architectural bad smells. In *CSMR 2009*, pages 255–258. IEEE.

[8] S. Herold. An initial study on the association between architectural smells and degradation. In A. Jansen, I. Ma-

lavolta, H. Muccini, I. Ozkaya, and O. Zimmermann, editors, *Software Architecture*, pages 193–201, Cham, 2020. Springer. ISBN 978-3-030-58923-3.

[9] 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 *12th MSR*, pages 246–257, Piscataway, NJ, USA, 2015. IEEE Press. ISBN 978-0-7695-5594-2.

[10] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *4th IWPSE*, pages 37–42, New York, NY, USA, 2001. ACM. ISBN 1-58113-508-4.

[11] D. Le, D. Link, A. Shahbazian, and N. Medvidovic. An empirical study of architectural decay in open-source software. In *IEEE ICSA, 2018*, pages 176–185.

[12] J.-L. Letouzey. The SQALE method for evaluating technical debt. In *MTD 2012*, pages 31–36, June 2012.

[13] Z. Li, P. Avgeriou, and P. Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101, 2015. ISSN 0164-1212.

[14] R. Marinescu. Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5), 2012. ISSN 0018-8646.

[15] 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 *ECSA*. Springer, 2018.

[16] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. In search of a metric for managing architectural technical debt. In *IEEE/IFIP WICSA and ECSA*, pages 91–100, Finland, 2012. IEEE.

[17] R. Roveda, F. Arcelli Fontana, I. Pigazzini, and M. Zanoni. Towards an architectural debt index. In *44th SEAA 2018*, pages 408–416.

[18] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto. *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Halsted Press, New York, NY, USA, 2004. ISBN 0470870931.

[19] A. Shahbazian, D. Nam, and N. Medvidovic. Toward predicting architectural significance of implementation issues. In *15th MSR*, pages 215–219, 2018.

[20] G. Suryanarayana, G. Samarthyam, and T. Sharma. *Refactoring for Software Design Smells*. Morgan Kaufmann, 1 edition, 2015. ISBN 978-0-12-801397-7.

[21] E. Tom, A. Aurum, and R. T. Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6): 1498–1516, 2013.

[22] R. Verdecchia, P. Lago, I. Malavolta, and I. Ozkaya. Atdx: Building an architectural technical debt index. In *Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2020.

[23] W. Wu, Y. Cai, R. Kazman, R. Mo, Z. Liu, R. Chen, Y. Ge, W. Liu, and J. Zhang. Software architecture measurement - experiences from a multinational company. In *12th ECSA, 2018*, pages 303–319.