An Architectural Smell Evaluation in an Industrial Context

Francesca Arcelli Fontana
University of Milano-Bicocca
Milano, Italy
email: arcelli@disco.unimib.it

Federico Locatelli
Anoki
Milano, Italy
email: f.locatelli@anoki.it

Ilaria Pigazzini
University of Milano-Bicocca
Milano, Italy
email: i.pigazzini@campus.unimib.it

Paolo Mereghetti
Anoki
Milano, Italy
email: p.mereghetti@anoki.it

***Abstract*—A known symptom of architectural erosion is the presence of architectural smells in software systems. They are the result of design decisions which negatively impact on software quality, and may lead to what is called Architectural Technical Debt. When such problems arise, developers feel difficulties in maintaining and evolving their architectures. Some tools have been developed to automatically identify architectural smells and in this study we propose the evaluation of architecture erosion in an industrial context through Arcan, an analysis tool able to identify eight architectural smells. In particular, we report the results of an industrial case study born from the collaboration between a Laboratory of the University of Milano-Bicocca and an italian company active in the software consulting field. The study has been structured as a survey on the architectural smells detected by the tool, from which we collected the feedback and opinions of the three projects' developers. Developers learned about architectural smells and became aware of the fact that their project had additional problems with respect to what they knew. We propose this work as a pilot for future works on the perception of AS in industrial context.**

Keywords–Architectural Smells; Architectural Debt; Industrial study; Refactoring; Criticality.

I. INTRODUCTION

Architectural Smells (AS) are design decision which negatively impact on software quality. They represent the main source of investigation in order to evaluate and manage architectural debt [1][2]. While code smells [3], in particular some of them (such as Large Class, Long Methods, Duplicate Code) are all well known from the developer/practitioner perspective, architectural smells are not so well known. Developers are often unaware of these smells, and they focus their attention on short term tasks, such as bug fixing and other code level issues. They are not aware of the possible accumulating debt due to the presence of architectural smells in their projects.

In order to better investigate how architectural smells are perceived by the developers, we describe in this work an evaluation we performed in an industrial context on the detection and perception of architectural smells by the developers/practitioners. With this purpose, we started a collaboration between the Evolution of Software Systems and Reverse Engineering Laboratory (ESSeRE Lab) of the University of Milano Bicocca [4] and the Anoki company in Milano [5]. The architectural smells considered in this evaluation are those currently recognized by the Arcan tool [6] developed by the ESSeRE Lab. The developers of the company received material and explanations on the considered architectural smells. Then we proposed a survey with different questions that they had to answer related to each instance of the inspected smells.

Through this study we aim to answer the following Research Questions:

RQ1: How are architectural smells perceived in an industrial context? With the answer to this question we aim to investigate whether the concept of AS is known and whether developers perceive them as problems with some kind of impact on software quality attributes. We are also interested in exploring other possible smells/problems present in the analyzed project considered harmful by developers, in order to improve the Arcan tool with new detectors.

RQ2: What practitioners suggest according to the refactoring of the smells? The effort required to fix AS, such as Cyclic Dependency, is higher than fixing a code smell [3], because it implies to move/modify both methods and classes of a system [7]. For this reason, we aim to investigate the opinion of the developers about the possible actions to be taken regarding the refactoring of the AS.

RQ3: Which are the most critical smells according to the practitioners perception? We aim to identify the smells perceived as most critical according to the evaluation of the interviewees. This information could be useful during software development by trying to avoid them and remove them first, since the most critical smells could lead to a progressive architecture degradation.

Moreover, with our study we aimed to reach further goals related to the validation of the Arcan tool, namely: *The evaluation of 8 different types of architectural smells* detected by Arcan through the feedback of the developers of the company, which could provide also useful hints to enhance the AS detection strategies and the possible definitions of new metrics (severities) able to discriminate the different smells by their criticality and *Additional feedback* on the possible usefulness of architectural smell detection, as the one outlined by the practitioners, related to the migration towards a microservice architecture.

The paper is organized through the following sections: in Section 2 we describe some related work, in Section 3 we introduce the AS considered during this evaluation, in Section 4 we describe the study design and the different questions used in the survey, in Section 5 we outline the main results, in Section 6 the lessons learned and in Section 7 the threats to validity. Finally, in Section 8 we report the answers to our RQ, the conclusions and future developments.

II. RELATED WORK

Several works have been done on the evaluation of code smells or other kind of code violations in collaboration with practitioners, such as for example a survey on code smells performed by Yamashita et al. [8] which outlines that a large proportion of developers did not know about code smells. A study of Soh et al. [9] where professionals were hired to perform maintenance tasks in order to assess whether code smells affect maintenance activities. Palomba et al. [10] conducted a study on developer's perception of the nature and severity of code smells. Tahir et al. [11] investigated how developers discuss code smells and anti-patterns across three technical Stack Exchange sites.

Few works focused on the evaluation of AS, in particular in an industrial context through the feedback of the developers/practitioners. Arcelli et al. [6] evaluate the precision of the detection results provided by the Arcan tool on the detection of three smells in two industrial projects. Wu et al. [12] present their experience in using a software architecture measurement standard through a collaboration with a company to evaluate, measure, and improve the architectures of their software products. Mo et al. [13] report their experiences of applying three complementary automated software architecture analysis techniques, in some industrial projects. Pigazzini et al. [14] describe an approach based on AS detection and topic detection for the migration towards a microservice architecture in an industrial case study, where the developer provided also several feedbacks on the usefulness of the AS detection during the migration steps.

In a previous study [2], we conducted an in-depth investigation on the identification and prioritization of architectural debt in an industrial context through a survey, interviews and inspection of the code with the practitioners of an industry in Sweden. With respect to this work, in this study we consider a larger number of smells, eight smells instead of three, the developers evaluated a higher number of instances of smells, the previous detection rules of the three smells have been improved and the survey has been changed through the introduction of new questions.

III. ARCHITECTURAL SMELLS

We consider eight AS, which correspond to the currently detected smells by the Arcan tool [6] on Java *components* i.e. classes and/or packages:

Cyclic Dependency (CD): refers to a component that is involved in a chain of relations that break the desirable acyclic nature of a component dependency structure. Arcan detects this smell on classes (CD-C) and packages (CD-P) and according to different shapes as those described by Al-Mutawa et al. [15].

Hub-Like Dependency (HL): occurs when a component has (outgoing and ingoing) dependencies with a large number of other components [7]. This smell is detected on both classes (HL-C) and packages (HL-P).

Unstable Dependency (UD): describes a component that depends on other components that are less stable than itself, with a possible ripple effect of changes in the system [16]. This smell is detected on packages.

God Component (GC): occurs when a component is excessively large either in terms of Lines Of Code (LOC) or number of classes [17]. This smell is detected on packages.

Dense Structure (DS): arises when components in a project have excessive and dense dependencies without any particular structure i.e. without following a specific architectural design [18]. This smell is detected on the entire project under analysis and occurs when the density (the ratio of dependencies over the components) of the project is high. Hence, only one instance can be detected per single project.

Feature Concentration (FC): occurs when an architectural component implements different functionalities in a single design construct [19]. This smell is detected on packages.

Insufficient Package Cohesion (IPC): occurs when an architectural component has low internal cohesion [18]. This smell is detected on packages.

Scattered Functionality (SF): describes a system where multiple components are responsible for realizing the same high-level concern and, additionally, some of those components are responsible for orthogonal concerns [20]. For *concern*, we mean a software system's role, responsibility, concept, or purpose [21]. This smell is detected on packages.

We considered the above AS because they violate different design principles, so that we can ask for developer feedback on different kinds of architectural problems. In particular, CD, HL, UD and DS are based on dependency issues: dependencies are of great importance in software architecture and components that are highly coupled and with a high number of dependencies are considered more critical, since they have higher maintenance costs. GC and IPC smell violate the modularity principle; finally FC and SF violate the separation of concerns principle [21].

Arcan bases all its computations on the **dependency graph** which is the representation of the project under analysis in form of a directed graph. The basic nodes represent the system entities, such as Java classes, packages and methods. Edges represent the relationships among the various entities. We exploit the graph to detect all the smells which affect the dependencies, e.g., Cyclic Dependency smell, which is caused by the presence of circular dependencies in the graph. However, some smells need other kinds of information in order to be detected, for instance the Feature Concentration and Scattered Functionality smells regard how system features are organized inside a project. Hence, Arcan is also able to generate the **feature graph**, whose aim is to represent the *features* (as synonym of concerns) that can be associated to the different parts of the system architecture. The feature graph associates a name in natural language to a set of project files, enabling developers to *read* how features are disposed across the project. Both Arcan graphs can be stored in the Neo4j [22] graph database, which we exploit also to visualize them.

IV. CASE STUDY DESIGN

In this study, a survey with different questions was given to the practitioners in order to obtain meaningful data on the AS listed in Section III. The practitioners were three and they were all developers belonging to the team that was working on the analyzed project at the time the survey was proposed. The first one was a junior developer with 4 years of experience working on the project analysed in this study. The second one was a middle developer with 9 and a half years of experience of which 1 year and a half spent working on the project. The third one, the team leader, was a senior developer with almost 15 years of experience working on the project for 2 years. We now describe the steps followed in the case study:

1) During a meeting at the company, one of the author introduced the practitioners to the notion of AS by explaining them what they are and why it is important to identify and refactor them. In the same meeting, Arcan and its principal functionalities have been introduced.

2) We then sent them a document containing the detailed descriptions of all the AS detected by Arcan, including a description on how Arcan detects them, the metric thresholds and formulas used for the detection. We gave them a week to read (during their normal work at the company) the document and study the AS meaning and relevance.

3) After that time, we instructed them on how to properly answer the questions of the survey, by briefly explaining the different categories and meaning of the questions.

4) We exploited the Google Form tool to create the survey. We provided the URLs to access it and we assigned 15 days to answer all the questions.

The survey contains 12 questions that the three practitioners had to answer individually for each AS instance. In particular, 19 AS instances were presented. For each instance, we provided the description of the smell type it refers to and we contextualized it by reporting all classes/packages affected by the smell. We also attached a visual representation of the smell with the dependency graph thanks to the Neo4j graph visualizer used by Arcan[23]. We presented 19 instances because we selected for each type of AS the ones that, in our opinion, were the most interesting, trying also to include instances with different granularity (for CD and HL) and different characteristics (for IPC, FC and GC). We call *smell characteristics* the smell properties that can be measured by a metric. For IPC we measure the Lack of Component Cohesion (LCC) [24], which is a metric ranging in $(0, 1]$, where 1 corresponds to packages with a complete lack of cohesion. For SF and FC we consider the number of features inside packages. For GC we compute the number of classes in a package and the LCC metric.

We assume that according to the different characteristics, AS can have different *criticalities* intended as the severity and effort needed to remove them: in this paper we also study if the perceived AS severity has a relation with such characteristics. Table III in Section V contains all the instances considered for each AS. We decided to analyse a greater number of instances for the new smells that we did not evaluate in previous works [6][25].

A. Analyzed Project

The analyzed project is a Business Management System written in Java with a monolithic architecture, but the developers are interested in a migration towards a microservices one. Table I reports the project's metrics and the number of AS instances detected by Arcan on it. The analyzed project is 10 years old and can be considered a medium-large project with 1343 classes and 112 packages. Arcan detected a total of 367 smell instances of which mostly are Cyclic Dependency between classes (135), Insufficient Package Cohesion (107) and Scattered Functionality (81), while smells like Hub-Like (3) and Feature Concentration (4) are much less present.

TABLE I. ANALYZED PROJECTS METRICS AND ARCHITECTURAL SMELLS

metrics		**architectural smells**									
NOC	NOP	#CD (classes)	#CD (package)	#HL (classes)	#HL (package)	#UD	#GC	#DS	#IPC	#FC	#SF
1343	112	135	5	3	3	19	10	1	107	4	81

B. Data Collection through the Survey

The questions asked to the developers in the survey are reported in Table II. Each question aims to gather the developer's evaluation on specific aspects of the analyzed AS, that is particularly valuable considering their deep knowledge on the project. The proposed questions can be grouped by category:

AS detection and awareness $[Q1 - Q3, Q12]$: this set of questions aim to evaluate the precision of the Arcan detection strategies and investigate the awareness of the developers on the presence of the smells.

AS impact$[Q5 - Q6]$: these questions aim to collect information about the perceived impact of AS on different software quality attributes.

AS refactoring$[Q7 - Q9]$: such questions gather information about whether refactoring activities, in the opinion of the developers, should be conducted and the type of refactoring needed to remove the smell.

AS severity, refactoring effort and priority$[Q4, Q10 - Q11]$: these questions aim to evaluate the effort/time needed to apply the refactoring and understand whether the smells can be ranked depending on their criticality (severity), i.e. if it is possible to quantify the smell impact thanks to the evaluation of specific smell characteristics, e.g., smell size (the number of affected classes/packages). To evaluate the answers of these questions, we define and compute three metrics (see section 5 for more details on the metrics computation), namely *Average Severity* of the smells, i.e., the average criticality that developers' associate to the smells, the *Average Effort* needed to refactor the smells and the *Average Priority of refactoring* that can be associated to the smells, i.e., the ordering of the smells depending on which should be refactored first. We chose to compute these values in order to summarize the collected data and be able to compare them.

The proposed questions are of three types: binary questions, where the possible answers are Yes or No; closed-ended questions, with multiple possible answers and open-ended questions, which were optional because we did not want to force the practitioners to spend too much time on them and, in some cases, no answer was needed, e.g., Q3 if the smell instance is considered as a problem by the practitioner. In this way we were able to collect both quantitative and qualitative answers, in particular the latter allowed us to gain insights about the concrete opinions of the practitioners.

V. RESULTS

After an accurate analysis of the answers submitted by the practitioners, significant results were extracted and reported in Table III and Table IV. The collected data reported in this section will be exploited to answer the RQs.

A. AS detection and awareness

One of the information we aimed to obtain from this case study was the classification of each AS instance in true positive or false positive. A smell instance discovered by Arcan was classified as true positive if most of the developers asserted that it is an actual issue/problem in the system, otherwise, it

TABLE II. PROPOSED QUESTIONS

ID	Question	Possible Choices
Q1	Does the reported smell represent a problem in the system?	Yes or No
Q2	Were you aware of the presence of this smell in the system?	Yes or No
Q3	If it's not a problem, do you think that this could be a case of false positive AS? Or an AS not critical? For which reasons?	N/A (open-response)
Q4	How significant are the negative impacts caused by the smell in your opinion?	**0** - Not a problem **1** - Low severity **2** - Mid-Low severity **3** - Mid-High severity **4** - High severity
Q5	If it has negative impacts, which of the following software internal qualities has this type of smell an impact on?	• Reliability (R) • Efficiency (E) • Security (S) • Maintainability (M) • Other
Q6	If not removed, the impact of this type of smell get worse as time passes	**0** - Disagree **1** - Somewhat Disagree **2** - Somewhat Agree **3** - Agree
Q7	What refactoring would you suggest to conduct? (e.g. move class, extract class, extract components, extract layers, etc.. Take in consideration your best option only)	N/A (open-response)
Q8	Do you think that conducting the refactoring would create negative side-effects? If yes which ones?	N/A (open-response)
Q9	If no refactoring should be conducted, which is the reasons?	• Not a real AS (false positive) • The smell does not represent a problem because there is not a better solution • The removal of this smell is too expensive • Other
Q10	How much effort/time can be required to refactor the smell?	**0** - No refactoring needed **1** - Low (< 8 h) **2** - Mid-Low (8-50 h) **3** - Mid-High (50-100 h) **4** - High (>100 h)
Q11	What do you think is the overall priority of refactoring this smell?	**0** - No refactoring needed **1** - Low priority **2** - Mid-Low priority **3** - Mid-High priority **4** - High priority
Q12	There is any architectural issue that you know is present in the system, but was not treated in this survey? If there is, describe it briefly	N/A (open-response)

was classified as false positive. The related questions are Q1 and Q3, but in some cases the answers given by a developer to these questions were incoherent, so we considered as more relevant answer the one provided for Q3 because it is an open answer question. **Among all the 19 AS instances presented in this survey, only 6 were classified as false positives, for an overall precision equals to** 70%. The AS with the higher rate of false positives are the HL on Package with 1/1 and SF with 2/3, while only 2/4 of GC and 1/4 of IPC are false positives. All the other AS instances have been indicated as true positives.

The developers explained also why some instances were false positives, i.e., real smells present in the code which do not represent a problem, in their opinion. For example, some false positives are special cases: the Hub-Like Dependency on Packages instance was detected on a package that contains utility classes which "are supposed to be used by classes of any kind", as stated by the developers; one of the God Component was detected on a package that contains many classes hierarchically organized in that package to avoid *boilerplate code* (sections of code that have to be included in many places with little or no alteration [26]) as declared by the practitioners. Regarding Scattered Functionality, only one of them was considered a true positive: this kind of smell is meant to point out defects in a package-by-feature [27] organization which is desirable in some cases, but not when the actual design is layered, as the project analyzed in this study. Developers got aware of that and signaled it to us, except for the case they considered true. Consequently, they also indicated this type of smell as the less critical, meaning that on their architecture the detected instances do not cause harm.

Another information we acknowledged from the answers to question Q2 was the **awareness of the developers regarding the considered AS**. One developer declared that he/she was already aware of the presence of 15 out of 19 (78%) smell instances, while the others 8 out of 19 (42%) and 5 out of 19 (26%). This information points out that the analysis with Arcan allowed them to discover some smells they were not aware of. Finally, we asked them to list smells or other problems that they knew were present in the system, but not included in the list of smells detected by Arcan in order to identify other possible problems/smells to consider for future extension of the tool detection strategies (Q12). However, no problem has been outlined by the developers in answering this question, so we could not extract useful information in this regard.

B. AS impact

Questions Q5 and Q6 had the purpose of gathering data about which software attributes each smell instance affects and if this negative effect of the smell will get worse as time passes. In question Q5, each developer could select more than one software attribute between Maintainability, Efficiency, Security and Reliability and suggest others ones that were not present among the possible choices. On the other hand, by answering question Q6, they could specify how much they agree with the statement "If not removed, the impact of this type of smell get worse as time passes". This information is summed up in Table III: for each AS instance (first column), the number of practitioners that selected each quality attribute is reported next to the letter indicating the attribute (second column), the Agreement on Q6 answer (third column) was computed by assigning a value (from 0 to 3, see Table II) to each possible choice of question Q6 and summing these values based on the answers of the developers. The higher this sum, the higher the agreement.

One relevant observation is that all the smells were considered affecting maintainability by at least one developer. For one smell instance, *God Component A*, a developer suggested an additional aspect, that "they affect the domain structure" i.e. how the domain model is organized across the different packages. **For what concerns smells that get worse as time passes, we discovered that the developers found Hub-Like on Classes and Feature Concentration the most problematic in these terms.** Even if we sum up the agreement for each type of smell and normalize respect to the number of

TABLE III. RESULTS FOR EACH ARCHITECTURAL SMELL INSTANCE

Architectural Smell Instance	Affected Software Aspects	Agreement on Q6
Cyclic Dependency on Classes	M(3), E(2)	6
Cyclic Dependency on Packages	M(3), E(1), R(1)	6
Hub-Like Dependency on Classes	M(3), E(2), R(1)	**8**
Hub-Like Dependency on Packages	false positive	–
Unstable Dependency	M(3), E(1), R(1), S(1)	5
God Component A	M(1), E(1), Domain structure/feature concentration(1)	4
God Component B	M(3), E(1), S(1)	7
God Component C	false positive	–
God Component D	false positive	–
Insufficient Package Cohesion A	false positive	–
Insufficient Package Cohesion B	M(2), E(1), R(1)	3
Insufficient Package Cohesion C	M(3), E(1), R(1)	3
Insufficient Package Cohesion D	M(3), E(1), R(1)	6
Feature Concentration A	M(2), E(1), R(1)	3
Feature Concentration B	M(3), E(1), R(1), S(1)	**8**
Scattered Functionality A	M(1), R(1)	3
Scattered Functionality B	false positive	–
Scattered Functionality C	false positive	–
Dense Structure	M(1), E(1), R(1), S(1)	5

Key: M: Maintainability, E: Efficiency, R: Reliability, S: Security

instances, the smell which get worse the most is still Hub Like Dependency on classes.

C. AS refactoring

We also asked various questions (Q7, Q8, Q9) regarding the possible refactoring of each smell. Answering two of these questions (Q7,Q8) was optional since they are open-ended questions, thus we collected a limited number of answers. However, the few data helped us in confirming a specific aspect: **smells like Scattered Functionality, Insufficient Package Cohesion and Feature Concentration are meant to point out defects in a package-by-feature [27] organization which is desirable in some cases, but not when the actual design is layered.** These smells are detected by Arcan to provide also a microservice migration support, where a package-by-feature organization is preferable and more useful [14]. Hence, the refactoring of these smells is effective with the aim of reorganizing the project as package-by-feature i.e. transforming the layers into microservices.

Instead, with the answer to Q9, developers provided the reason why they would not refactor the smells instances which they indicated as true positive. In particular, there was a similar response for all CD-P, HL-C, UD and FC asserting that they should not be refactored because “their refactoring would be too expensive”. For the FC instance, there was also another answer declaring that “it should not be refactored because there is not a better solution”. The same answer was given for one GC instance, all the SF and all the IPC. In brief, developers reported that no refactoring should be conducted on the detected smells because the refactoring activity is too expensive and because sometimes the smell presence is unavoidable. We suggested to the developers that a possible solution to avoid the too expensive smells is to periodically run Arcan on the system and remove smells as soon as they appear. Moreover, their new knowledge about AS can help them in avoiding their introduction in the first place.

D. AS severity, refactoring effort and priority

Other useful data collected for each AS are the perceived severity (Q4), refactoring effort (Q10) and refactoring priority (Q11), which we summarize through the metrics introduced in Section IV-B: the *Average Severity*, the *Average Effort* needed to refactor the smell and its *Average Priority of refactoring*. They are computed by assigning a score to each possible choice the developers could pick to answer questions Q4, Q10 and Q11, then calculating, for each smell instance, the sum of the scores indicated by the developers and eventually computing the average sum for each AS. The maximum reachable value is 12, which happens if all developers agree on answering “High”. We found out that the metrics values for all the AS are very close to each other, meaning that developers perceive these three metrics as linearly dependent. The only smells for which one of the three metrics mentioned above has a value that is much higher than the other two metrics are *1)* the Cyclic Dependency smells, which have an Average Priority of refactoring of 8, while the other metrics are equal to 5 and *2)* Dense Structure, which has an Average Effort of 8 and an Average Severity of 5. If we consider the percentages computed on all the smells of the answers regarding Severity, Effort and Priority, we find out that the majority of the smells have been classified as having *Medium-Low* Severity (35%), Effort (33%) and Priority (25%).

Furthermore, we ranked the true positive AS depending on the metrics’ values. We reported the most notable ones in Table IV. As we previously mentioned the developers assigned values very close to each other for every AS, so the logical consequence is that the **AS with the highest severity has the highest priority of refactoring and requires the highest effort to be removed too** and vice-versa. In this scenario, Hub-Like Dependency on Classes is the smell with the highest values for all the metrics and Insufficient Package Cohesion is the one with the lowest.

Moreover, we analyzed the possible correspondence between the severity of the AS as perceived by the developers and the metrics used to detect the AS by Arcan, with the aim of identifying a specific smell characteristic (see Section IV) that may become a severity criterion. We checked the *1)* Insufficient Package Cohesion smell according to the Lack of Component Cohesion (LCC)[24], *2)* Feature Concentration according to the number of features inside packages and *3)* God Component according to the number of classes in a package and the LCC metric. We did not check smells with only one true positive.

TABLE IV. RELEVANT SMELLS

	Severity		Effort		Priority	
AS	Most critical	Least critical	Most hard-to-remove	Least hard-to-remove	Most urgent-to-remove	Least urgent-to-remove
HL-C	X		X		X	
IPC		X		X		X

We observed that all the Insufficient Package Cohesion instances have a severity of 3 except for one instance that has a severity of 4: this is also the only one with a LCC equals to 1, which indicates a complete lack of cohesion among the classes belonging to the affected package. The two instances of Feature Concentration have respectively a severity of 4 and 7 and a number of features inside the package of 14 and 28, so we think that this characteristic may be linked to the severity of the smells. Finally, **God Component's instances did not show a link between the severity and the number of classes contained in the package even considering only true positives instances, but we identified a relationship between the developers' severity perception and the Lack Of Component Cohesion of the affected package.** The packages corresponding to the most severe instances (severity=5) have also the highest values of LCC (0.57 and 0.78), while the instance with a severity of 4 affects a package with a LCC of 0.45 and the package affected by the least severe instance (severity=2) has a LCC inferior to 0.2: the higher the LCC of the package, the higher the severity of the AS instance. In brief, we identified a link between: *1)* the severity of the IPC instances and the LCC metric; *2)* the severity of the GC instances and the LCC metric; *3)* the severity of the FC instances and the number of features inside the affected packages.

VI. LESSONS LEARNED

We briefly describe below the principal lessons learned and feedback we obtained from the survey that we exploit to answer the RQs in the next section.

Lesson learned for the Arcan tool developers First of all, we received a feedback from project experts on the precision of the results of the Arcan tool and their usefulness, in particular we obtained feedback on how many detected AS were not actual issues and, taking a closer look to these smells, we can improve the tool in order to reduce the number of false positives results. Secondly, we gathered useful data on how critical each AS instance is, which software attributes it affects and which AS's characteristics may be linked to its criticality: we can use this information to add new functionalities to the tool, like assigning a severity value to each AS instance in order to establish a priority ranking that can reduce the developers' overall refactoring effort. One remarkable example is the severity of GC, which we hypothesised linked to the progressive higher number of classes inside the package. Actually, we found out that developers considered more critical GC instances which have higher LCC: hence a large package with a very low internal cohesion represents the worst possible case that a developer can face.

Lesson learned for the developers/practitioners of the company The developers got useful feedback while examining the smells we showed them through the survey. When we asked them about what the survey taught them, they outlined that *1)* an analysis made through the support of a tool, like Arcan, can bring up issues that the developers did not notice before: they discovered new AS that they never considered as problems and they will keep them in mind from now on to avoid falling into the same mistakes. *2)* They also learned that AS can become relevant issues when working with a large system, because coding can easily lead to the creation of several little smells instances that become progressively greater as the project grows. This aspect, in their opinion, can make the system hard to maintain and understand, specially when trying to identify and separate the system functionalities, such as for a possible migration to a microservices architecture which requires to identify, isolate and put in the same microservice all the classes that work on the same functionality. *3)* Finally, they also got aware of the fact that the developer's experience is important in a perspective of knowledge of the project he/she's working on: the junior developer, thus the least experienced among the three, was also the least aware of the reported smells even though he/she has been working on the project for a longer period (4 years vs 1.5/2 years) compared to the others.

VII. THREATS TO VALIDITY

As for **construct validity**, there is a possibility that the practitioners misinterpreted what the AS represent or what we asked in the questionnaire. However, we mitigated this threat by explaining each type of smell and also each smell instance with a detailed description and the support of a graph visualization. As for **internal validity**, it is unlikely that the opinions on the negative impact of the smells on the project quality reported by the practitioners would be affected by factors that are not related to the AS, since we explained and contextualized each smell in the survey in detail and practitioners were careful to inquire the main causes of the perceived negative impact inside the code. The threat to **external validity** is due to the fact that the case-study has been conducted in a single company and on a single project, hence the results may not apply to other application domains. The reason of such limited scope is due to the difficulties of finding available industrial projects to analyze: in the future, we aim to extend our work with more of them. Moreover, the number of studied AS was limited and for some types, such as Hub-Like Dependency on package, only one instance was analyzed. However, we mitigate this aspect by interviewing three developers with different skills and seniority and by measuring their accordance. As for **reliability**, we identified cases where the practitioners contradicted themselves, however we mitigated this problem by proposing also open-ended questions and collecting their concrete opinions. Even if the study is replicable, the results are based on practitioners' experience and perception, hence the real impact and severity of the AS might differ from the one reported here. However, we subjected the survey to the developers who are actively working on the project and are directly interested by the possible presence and impact of the AS, so their opinion is the most valuable for validating the AS and Arcan.

VIII. CONCLUSION

In this paper, we described the evaluation of 8 AS through the feedback of 3 developers on one industrial project. In particular, we ran the AS detection tool Arcan on the project and presented the results to the developers. Then, for each analyzed instance of AS, we collected information about AS impact, severity and its refactoring through a survey composed of 12 questions. Every developer individually answered the questions of the survey and on such data we build our study. We now report the answers to our research questions, which summarise the results of our study:

RQ1: How are architectural smells perceived in an industrial context? Developers did not know about the concept of

AS, however they reported that they were aware of some of them. They also confirmed that AS have a negative impact on software internal qualities, in particular on maintainability. They recognized the risk linked to the presence of AS in their architecture and acknowledged the usefulness of automatic tools, like Arcan, which can detect this kind of anomaly.

RQ2: What practitioners suggest according to the refactoring of the smells? We were not able to extract from the survey answers, valuable suggestions concerning the refactoring of the smells. However, the few data pointed out that developers would not refactor some of the detected instances because the refactoring activities could be too expensive and for some cases, the smell could represent the only possible solution. A specific comment was made on the refactoring of Feature Concentration, Scattered Functionality and Insufficient Package Cohesion: the refactoring of such ASs is useful, when the system architecture is layered, to prepare the migration towards microservices, by structuring the packages by feature and thus easing the identification of the candidate services. This comment confirmed a result investigated in our previous work, where we exploited Arcan to detect the candidate microservices of an industrial project and collected the developers feedback on the proposed solution [14].

RQ3: Which are the most critical smells according to the practitioners perception? The most critical smell, in the context of this study and the developers' opinion, is HL on classes, which is also one of the smell which gets worse the most, as time passes. Developers should pay attention to this kind of AS and remove it as soon as it appears. On the contrary, IPC is the least critical AS.

In the future, we aim to carry out more studies like the one presented in this paper, on different projects of different applications domain and companies in order to better understand how AS are perceived in an industrial context and improve the AS detection support. We also aim to study how to prevent the introduction of AS by leveraging on machine learning techniques [28]: in this way practitioners could avoid the extra costs due to the refactoring of the AS.

REFERENCES

[1] R. Verdecchia, "Architectural technical debt identification: Moving forward," in 2018 IEEE International Conference on Software Architecture Companion, ICSA Companion 2018, Seattle, WA, USA, April 30 - May 4, 2018, pp. 43–44.

[2] 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.

[3] M. Fowler, Refactoring: Improving the Design of Existing Code. Boston, USA: Addison-Wesley, 1999.

[4] E. Lab, Evolution of Software Systems and Reverse Engineering Laboratory Official Website, 2020 (accessed August 2020). [Online]. Available: https://essere.disco.unimib.it/

[5] Anoki, Anoki s.r.l., 2020 (accessed August 2020). [Online]. Available: https://www.anoki.it/

[6] F. Arcelli Fontana, I. Pigazzini, R. Roveda, D. A. Tamburri, M. Zanoni, and E. D. Nitto, "Arcan: A tool for architectural smells detection," in Int'l Conf. Software Architecture (ICSA) Workshops, Gothenburg, Apr. 2017, pp. 282–285.

[7] G. Suryanarayana, G. Samarthyam, and T. Sharma, Refactoring for Software Design Smells: Managing Technical Debt, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2014.

[8] A. F. Yamashita and L. Moonen, "Surveying developer knowledge and interest in code smells through online freelance marketplaces," in USER 2013, San Francisco, CA, USA, May 26, 2013, pp. 5–8.

[9] Z. Soh, A. Yamashita, F. Khomh, and Y. Guéhéneuc, "Do code smells impact the effort of different maintenance programming activities?" in SANER 2016, Suita, Osaka, Japan, March 14-18, 2016 - vol. 1, pp. 393–402.

[10] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, and A. D. Lucia, "Do they really smell bad? A study on developers' perception of bad code smells," in 30th IEEE ICSME, Victoria, BC, Canada, September 29 - October 3, 2014, pp. 101–110.

[11] A. Tahir, J. Dietrich, S. Counsell, S. Licorish, and A. Yamashita, "A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites," Information and Software Technology, vol. 125, 2020, p. 106333.

[12] 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 ECSA 2018, Madrid, Spain, September 24-28, 2018, Proc., pp. 303–319.

[13] R. Mo, W. Snipes, Y. Cai, S. Ramaswamy, R. Kazman, and M. Naedele, "Experiences applying automated architecture analysis tool suites," in Proc. of the 33rd ACM/IEEE, ASE 2018, Montpellier, France, September 3-7, 2018, pp. 779–789.

[14] I. Pigazzini, F. A. Fontana, and A. Maggioni, "Tool support for the migration to microservice architecture: An industrial case study," in Software Architecture - 13th European Conference, ECSA 2019, Paris, France, September 9-13, 2019, Proceedings, pp. 247–263.

[15] H. A. Al-Mutawa, J. Dietrich, S. Marsland, and C. McCartin, "On the shape of circular dependencies in java programs," in ASWEC 2014, Milsons Point, Sydney, NSW, Australia, April 7-10, 2014. IEEE Computer Society, 2014, pp. 48–57.

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

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

[18] T. Sharma, M. Fragkoulis, and D. Spinellis, "Does your configuration code smell?" in Proceedings of the 13th International Conference on Mining Software Repositories, ser. MSR '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 189–200.

[19] H. S. de Andrade, E. S. de Almeida, and I. Crnkovic, "Architectural bad smells in software product lines: an exploratory study," in Proc. of the WICSA 2014 Companion Volume, Sydney, NSW, Australia, April 7-11, 2014. ACM, 2014, pp. 12:1–12:6.

[20] J. Garcia, D. Popescu, G. Edwards, and N. Medvidovic, "Identifying architectural bad smells," in 2009 13th CSMR, Kaiserslautern, Germany, 2009, pp. 255–258.

[21] E. W. Dijkstra, "On the role of scientific thought," 01 1974.

[22] N. Inc., Neo4j, 2020 (accessed August 2020). [Online]. Available: https://neo4j.com/

[23] ——, Neo4j graph visualization, 2020 (accessed August 2020). [Online]. Available: https://neo4j.com/developer/graph-visualization/

[24] T. Sharma, P. Mishra, and R. Tiwari, "Designite: A software design quality assessment tool," in Proc. of the 1st Intern. Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities, ser. BRIDGE '16. NY, USA: ACM, 2016, p. 1–4.

[25] 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.

[26] N. Mitchell and C. Runciman, "Uniform boilerplate and list processing," in Proc. of the ACM SIGPLAN Workshop on Haskell Workshop, ser. Haskell '07. NY, USA: ACM, 2007, p. 49–60.

[27] K. Lee, K. C. Kang, W. Chae, and B. W. Choi, "Feature-based approach to object-oriented engineering of applications for reuse," Software: Practice and Experience, vol. 30, no. 9, 2000, pp. 1025–1046.

[28] F. A. Fontana, P. Avgeriou, I. Pigazzini, and R. Roveda, "A study on architectural smells prediction," in 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019, Kallithea-Chalkidiki, Greece, August 28-30, 2019, pp. 333–337.