Splicing Community and Software Architecture Smells in Agile Teams: An industrial Study

Damian A. Tamburri
TU/e - JADS
d.a.tamburri@tue.nl

Rick Kazman
University of Hawaii / SEI-CMU
kazman@hawaii.edu

Willem-Jan Van den Heuvel
Universiteit van Tilburg - JADS
W.J.A.M.v.d.Heuvel@jads.nl

Abstract

Software engineering nowadays largely relies on agile methods to carry out software development. In often highly distributed organizations, agile teams can develop organisational and socio-technical issues loosely defined as community smells*, which reflect sub-optimal organisational configurations that bear additional project cost, a phenomenon called* social debt*. In this paper we look into the co-occurrence of such nasty organisational phenomena—community smells—with software architecture smells—indicators that software architectures may exhibit sub-optimal modularization structures, with consequent additional cost. We conclude that community smells can serve as a guide to steer the qualities of software architectures within agile teams.*

1. Introduction

Software development is about people: organised crowds of developers collaborate (hopefully in harmony) with designers and managers, to accommodate communities of stakeholders and users—this is even more true in agile teams, whose performance and *velocity* greatly relies on the performance and individual engagements of involved individuals and stakeholders around them [1, 2]. The impact of this notion was originally perceived in 1968, when a computer programmer by the name of Melvin Conway observed that the structure of a software system reflects the social structure of the organisation that produced it [3]. On one hand, a social structure, or *community*, in such agile teams emerges as a consequence of social interactions and arrangements between individuals that share the common agile software practice [4, 5]. On the other hand, distributed and complex agile organizations often develop sub-optimal patterns of recurrent organisational behavior [4]—also known as *community smells* [6, 5, 7]—that may be complicating the lives and success of such agile teams.

In this paper we seek to address two research questions: (1) *to what extent do community smells occur in agile teams?* — agile teams have a known prowess in developing software [8, 9], but nobody has addressed yet whether the organisational structures behind agile teams exhibit known community smells [5] which could be potentially corrected for further organisational improvement; *(2) to what extent do community smells in agile organisational structures reflect good software quality outputs?* — as a proxy to software quality we consider software architecture smells [10, 11] that is, known patterns that may cause nasty software maintenance and evolution issues [11] and which may cause technical debt [12, 13].

To address these research questions we conducted a large-scale longitudinal industrial study over 30 software organisations that recently underwent a switch to employ agile methods. The software development teams in question operated in 9 large companies who work with well-known agile methods and approaches (e.g., Scrum [14]). We chose to focus on organizations that claimed to have recently adopted such methods since agile adoption is reportedly difficult and often causes heavy organisational strains [15, 16].

We found that community smells occur very frequently across our sample, with a specific case for smells that reflect organisational turmoil, rearrangement, or instability. Furthermore, community smells correlate considerably (+0,31 with P-value $<<$0,05) with architecture smells and are diffused across over 80% of our dataset, thereby representing a force to be reckoned with in the scope of agile teamwork and team governance.

We conclude that community smells can be used in the scope of: (1) agile retrospectives — to diagnose potential organisational and socio-technical issues encountered by agile teams; and (2) iteration-0 architectures — to avoid the nasty phenomena connected to sub-optimal community and architectural structures.

URI: https://hdl.handle.net/10125/60140
ISBN: 978-0-9981331-2-6

This article makes three novel contributions: (1) a systematic exploration of community smells in agile software teams; (2) the quantitative analysis of co-occurrence between community smells and the software architecture quality of those teams; (3) a comprehensive replication package to encourage the verifiability of our results as well as further replications of this study in both (closed source) industrial projects and open-source projects. Practitioners may benefit from these contributions in avoiding the identified community smells, or by restructuring their software architectures to remove smells and better align to the social structures. At the same time, our contributions encourage further research into the topics covered by our results.

Structure of the paper. Section 2 briefly outlines the background of this study, providing terms and definitions as well. Section 3 outlines our research design. Later, Section 4 shows our results while Section 5 discusses our contributions and their threats to validity. Section 6 outlines related work. Section 7 concludes the article hinting to future work.

2. Background and Scope of Our Study

2.1. Software Community Structures and Smells

Since the original publication of Conway's Law [3], positing the relationship between the structure of a system and the structure of the organisation that designed it, several studies have tried to understand more about this intriguing relation contained in the so-called "law".

On the one hand, the works by Cataldo et al. and Herbsleb et al. around socio-technical congruence [17, 18, 19] study software development as a social-technical activity, in which the technical and the social components need to be aligned to succeed. These and similar works introduce valuable socio-technical factors to be addressed and tracked for software communities to succeed. The fundamental component, Herbsleb [20], Damian [21] and others say, is to achieve an effective coordination among teams, whose organisational structures are a key dimension that should be considered as much as project plans, processes, and coordination mechanisms [19].

On the other hand, community smells reflect recurring sub-optimal organisational structure conditions (i.e., *patterns*) wherefore additional friction is added to functional software teams connected to nasty organisational behaviour, e.g., sub-optimal knowledge sharing, recurrent sharing delays, misguided collaboration and more. In the past researchers tried to identify the recurrent conditions around community smells for the benefit and use of software practitioners [22, 6, 5].

Table 1 (tailored from [5]) showcases the smells previously found in industrial practice and also re-used as objects of study in the scope of this work; the table reports community smells from literature that have a direct relation to software artefacts and the process in which they are produced and maitained. Columns 1 and 2 give a name for the smell and a brief description (including frequency of occurrence in parentheses besides smells' name, as reported in previous work [5]).

2.2. Software Architectures and Smells

Software architecture is an abstraction of a software system: its components, their properties, their relationships, as well as the choices that led to the system [23]. In this article we use architecture issues and smells, reported by practitioners, as a way to understand and characterise the organisational structures under study. We designed our study to minimize bias as much as possible and thus we let our practitioners describe their projects' architectures, using reference models from the Microsoft Application Architecture Guide (MAAG), freely available on MSDN[1] and previously known to all practitioners involved in our study. In the scope of this study, we used the architecture issues or "smells" described in the MAAG, since: (a) practitioners used them to report which issues were present in their architectures; (b) we used these to evaluate organisational structures, characteristics, and relations found. The list of issues follows below:[2]

1. **Impossible Component Swap:** this issue reflects architecture components which are too tightly connected to the rest of the architecture which may lead to substitution problems.

2. **Untraceable Business Requirement:** this issue reflects overly fine-grained architectures where it becomes difficult to trace high-level business requirements across the entire architecture. This weighs on the ability to analyze the quality aspects behind that requirement and consequently the overall quality of the architecture.

3. **Sloppy Modularisation:** this issue reflects carelessly modularised architectures, e.g., architectures that randomly decrease cohesion

[1] `https://msdn.microsoft.com/en-us/library/ff650706.aspx`

[2] Note that the terms used here are our own descriptive names for the issues reported in the MAAG.

Table 1. Community Smells in industry — an overview from the state of the art [5].

Smell (#occurrence)	Description
Time-Warp (27x)	Change in organizational structure and process leads people to make wrong assumptions that communications will take less time and explicit additional coordination is not needed.
Cognitive Distance (24x)	In software engineering, cognitive distance can be thought as the distance that developers perceive on the physical, technical, social and cultural level with respect to peers with considerable background differences.
Newbie Free-riding (24x)	Newcomers are left to themselves when it comes to understanding what to do and for whom, with consequent free-riding of older employees, i.e., the economic free-rider problem applied to software engineering.
Power Distance (24x)	The distance that less powerful or less responsible members of a software development community perceive, accept and/or expect with power-holders.
Disengagement (24x)	Developers think product is "mature enough" and send it to Ops even though it may not be ready.
Priggish Members (13x)	Developers tend to demand of others (e.g., Ops) pointlessly precise conformity or exaggerated propriety, especially in a self-righteous or irritating manner.
Cookbook Development (13x)	Developers are stuck in the ways they are used to work according to their cookbook and refuse innovative ideas or ways of working (e.g., agile methods or DevOps).
Institutional Isomorphism (11x)	Institutional isomorphism is the similarity of the processes or structure of one sub-community to those of another, be it the result of imitation or independent development under similar constraints. The danger here is lack of innovation, stagnance, lack of communication/collaboration.
Hyper-Community (14x)	A hyperconnected community is sensible of group-think but also influences other (smaller) subcommunities in its fold.
DevOps Clash (29x)	Clashes in the mix between Dev and Ops from multiple geographical locations with contractual obligations to either Dev or Ops lead to slower Dev and ineffective Ops.
Informality Excess (10x)	Excessive informality means the relative absence of information management and control protocols leads to information spillover.
Unlearning (12x)	New technological or organizational advancements or best practices, e.g., as part of training courses become unfeasible when shared with older members as a consequence of very high experience diversity. The new accumulated knowledge or best practice risks of being gradually lost.

for no good reason. This weighs heavily on the architecture's evolvability, its usefulness as a knowledge conveyor, as well as a means for division of work.

4. **Unscalable Architecting:** this issue reflects an architecture that can barely perform its functions in the face of rapidly increasing service demands; the consequent issues on architecture quality are obvious.

5. **Inflexible Architecture / God Classes:** this issue reflects architectures where a small number of components or modules contain significant functionality, creating bottlenecks for maintainability.

6. **Unmodifiable Core:** this issue reflects architectures that have gradually grown out of a core which is more and more becoming functionally untouchable and incomprehensible. This is often a consequence of Sloppy Modularisation.

7. **Spike-Centric Architecture:** this issue reflects architectures which emerged directly from *architecture spikes*, i.e., "[...] architectural spike is a test implementation of a small part of the application's overall design or architecture". Relying heavily on architecture spikes may cause severe integration and evolution issues.

8. **Quality-Implicit Architecture:** this issue reflects architectures for which quality analysis of one (or more) software architecture properties (e.g., performance, reliability, safety, etc.) is so complex that it is never actually done.

9. **Architecture Monolith:** this issue reflects architectures whose design has grown into a fully connected mesh and therefore cannot be evolved other than by incurring heavy refactoring costs. Such monoliths may have originated as God Classes.

10. **Insensitive Information Spreading:** this issue reflects architectures in which one or more components carelessly disseminate the data that the software architecture is manipulating. This can cause security vulnerabilities as well as privacy-policy violations.

Finally, we define the *architectural inefficiency* of an organisational structure pattern X as the count of software architecture structural flaws and issues (selected from the list above) corresponding to that pattern, for all subject projects:

Architectural Inefficiency:
$$\chi = \sum A_i(a_1, ...a_n);$$

where χ represents the architectural inefficiency of an organisational structure pattern while A_i represents reported software architecture issues for projects $a_1, ...a_n$. For the purpose of quantifying the relation between organisational and socio-technical community smells and software architectures, we are interested in using the above measurement and evaluating how this quantifies the relation between community and architecture smells.

3. Research Design

To obtain evidence for this research we used a mixed-methods approach featuring 3 phases of data collection and quality-assessment as well as 2 subsequent analysis phases: (1) survey design and Delphi study [24]; (2) online survey; (3) confirmatory interviews; (4) data summary and observer reliability assessment; (5) content analysis. This section outlines our research design in detail, starting from a walkthrough of our theoretical framework, target population, and sampling strategy.

3.1. Research Problem and Questions

The research problem we address reflects the potentially sub-optimal conditions that may affect software agile teams and how those conditions affect the quality of those teams' outputs. More specifically, we aim at (1) studying the occurrence of community smells in the context of industrial, closed-source agile teams and (2) correlating that occurrence with some indicator of software quality (i.e., architectural inefficiency). To this purpose, we formulate and address four research questions:

RQ1 to what extent do community smells occur in agile teams?

Sub-RQ1.1 does the diffuseness vary per agile method type? With the term *diffuseness* we refer to the degree to which community smells are actually occurring in our dataset; in the scope of this SRQ, we are interested to understand whether that quantity varies meaningfully with the variation of agile methods type.

Sub-RQ1.2 does diffuseness vary per experience level in teams? Referring to the above notion of diffuseness, in the scope of this SRQ we are interested to understand whether the experience with agile methods is itself a mediator.

RQ2 to what extent do community smells in agile organisational structures reflect variations in quality of software outputs?

3.2. Target Population and Sampling Strategy

Our study covers more than 4 years of software design and development organisational activity. Invitation letters were sent over this time-window to 136 practitioners with 4+ years of experience with software and its engineering. Following a strategic sampling strategy [25], the initial set of practitioners were extracted from our own networks: developers and architects who had previously worked with one or more of the authors of this study. Practitioners were initially contacted with an email asking of their interest in the study and, if they were interested, they were asked for contact details of 2 additional colleagues who might be available for the study as well. The process of establishing agility was built-in within our invitation letter—in particular, we employed the practices defined in previous work [26] to map the practitioners onto agile practices.

Out of our initial sample, 42 people responded. We filtered out 12 respondents as inappropriate, leaving a final sample size of 30. Our filtering criteria were:

- codebase size: split evenly among medium-sized (200-500 KLOC) projects, 33% large (500-850 KLOC) projects, and 33% very large ($>$ 850 KLOC) projects;
- main programming language - Java, C#, C, Python. In addition YAML and other scripting languages are included for projects in our sample;
- team size - our size distribution is evenly split among three ranges: medium ($<$10 members), large (10$>$20 members) and very-large ($>$20 members);
- team age - our project team age distribution is evenly split among three ranges: young ($<$24 months), established (24$>$32 months), and mature ($>$32 months);
- process maturity - our sample is evenly split across all maturity levels of the standard CMMI scale [27] starting from level 2. We excluded level 1 since this would introduce an element of uncertainty with respect to organisational structures (CMMI level 1 is disorganised by definition);

- architecture type - all our projects were sampled choosing service-oriented or service based applications based on concepts and definitions from literature [28] — this design choice is connected to the reported proneness of certain architecture styles with certain architecture issues [23, 29];

As a result of our sampling, the population for our study involved 90 practitioners from 9 different organizations. The market segments represented are: aerospace, heavy automotive industry, mobile-phone manufacturing, information systems consulting (two organizations), healthcare informatics, banking information systems, food production, and electronics.

3.3. Phases 1 and 2 - Survey Design and Execution

In this study, we aimed at a defensible and replicable qualitative-quantitative research design, striving to minimise our own interpretation of raw data directly elicited from the involved practitioners. In so doing, we hard-coded a Delphi-inspired approach [24] to refine a survey questionnaire which would directly and reproducibly yield the data we sought. In particular, a Delphi study relies on a panel of experts to answer questionnaires in two or more rounds (two rounds, in our case). In our case, after a first round of survey (Phases 1 and 2), a facilitator provided an anonymised summary of the previous two experts' forecasts. The third practitioner (Phase 3, see Sec. 3.4) was then encouraged to confirm, deny, or revise answers in light of the replies of other members of their panel. In summary, we used three people from one team, two of which answered the survey and the third would be interviewed to provide for a confirmatory connotation.

3.4. Phase 3 - Confirmatory Interviews

To instrument phase 3 of our study, we reached back to the previously uninvolved practitioners for every project in our sample. These practitioners were interviewed with a variation of our complete survey questionnaire (see Sec. 3.3) intended to confirm all the observations of their colleagues about their team as well as clarify any clearly conflicting answers. For this phase, we adopted an interview guide approach.[3] Thus structured, phase 3 led to a total of 31 interviews amounting to over 50+ hours of recorded material. While we cannot disclose the raw data from survey Phases 1 to 3, we have prepared a spreadsheet containing aggregated data from responses and confirmatory interviews. To encourage verifiability, this dataset is openly and freely available.[4]

3.5. Data Analysis

To analyse the results in this paper from a statistical perspective [30], we prepared descriptive statistical plots for our data as well as Pearson correlation calculations across the dimensions we studied, namely, we computed Pearson correlation coefficients between the occurrence of reported architecture smells and the occurrence of prominent organisational community smells[6, 5].

The use of Pearson product-moment correlation is well-formed since our study design "flattens" the quantities involved (i.e., the magnitude of organisational characteristics and of individual architecture smells) to linear sums which are more appropriately correlated by means of product-moment analysis.

4. Results

4.1. RQ1 - Community Smells Diffuseness in Agile Teams

A total of 106 community smells from Table 1 were reported throughout our sample dataset. fig. 2 plots an overview of occurrence frequency, with a median occurrence of 4,5 smells per type of smell, and 3 smells per all agile teams, regardless of their experience level. The most frequently occurring smells are clearly 3: (1) *Time-warp* (occurring 12 times); (2) *Cognitive Distance* (occurring 17 times); (3) *DevOps Clash* (occurring in every team). To give a representative view of how diffused the community smells are across our sample, Fig. 1 offers an overview of smells diffuseness across the sample using a radial diagram.

Furthermore, we registered (after data sample normalisation) almost no difference between agile method types, with respect to the frequency of community smells, with an occurrence of 579 community smells for Scrum and 557 community smells for other agile methods and approaches, achieving a standard deviation of 12,22 counts.

Finally, with respect to team experience, we registered a strong correlation of +0,315 (P-value $<<$0,05) between community experience and the proliferation of more community smells.

> **Summary for RQ1.** Our data indicates a strong occurrence and diffuseness of community smells in agile software development teams, with a prominent presence of smells connected to

[3] interview guide available here: `http://tinyurl.com/kl97cgq`

[4] anonymized - will be added in the accepted submission

organisational disruption (e.g., Time-Warp and DevOps clash). Furthermore, experience plays a role *not* in reducing the amounts of smells but, surprisingly (as our data indicates) the number of smells *increases* linearly with the number of months that the teams work with agile methods.

We conclude that the role of community smells is prominent in agile teams governance—further research is needed to fully elaborate and quantify the dimensions and implications of the above finding.

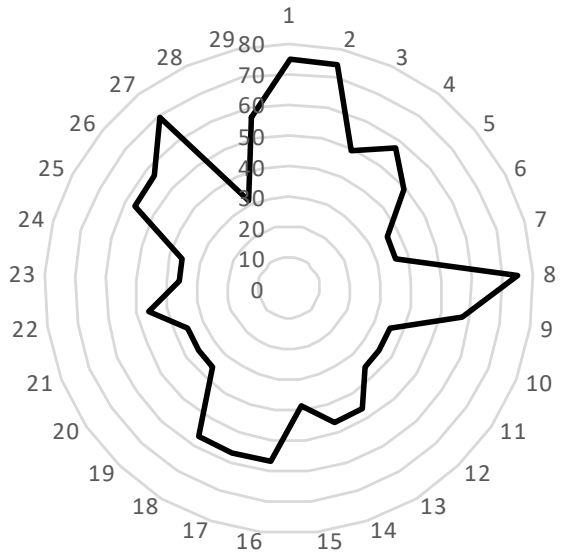

Figure 1. Community smells diffuseness across our sample - a radial diagram; datapoints appear as numbers around the outmost circle while occurrence scale appears at 12 o'clock on the plot.

4.2. RQ2 - to what extent do community smells reflect good quality?

To understand the degree to which community smells in agile teams reflect higher (or lower) quality of outputs, we used the architecture inefficiency measurement previously introduced in Sec. 2 and correlated that inefficiency with the occurrence of community smells, normalising both quantities per frequency.

First, we discovered a strong linear correlation of +0,34 (P-value $<<$0,05) between the two quantities involved.

Second, we discovered that 84% of our dataset shows evidence of overlap between community and architecture smells. That is, not only are the two quantities correlated but also their correlation persists in a subsantial portion of our dataset. This condition is represented by the radial overlap diagram in Fig. 3.

The diagram shows the overlap in diffuseness across our entire sample both of community smells (continuous line) and of architecture smells (dotted line)—an almost total overlap is evident.

Summary for RQ2. Our data indicates a strong correlation between software community smells and software architecture smells; furthermore, the correlation is present all across our sample.

We conclude that community smells are emergent phenomena to be reckoned with and addressed jointly with the quality of software artefacts being produced and maintained by software teams—even those working with agile methods. Perhaps this is to be expected, since agile teams typically do not have personnel and expertise specifically dedicated to software architecture nor the detection and resolution of architecture smells.

5. Discussions

This section outlines the observations and lessons learned with respect to the scope of our study, as well as detailing the practical and research impacts of the contributions offered in this paper. Finally, this section recaps threats to validity for the scope of this study.

5.1. Observations

In the scope of this study, we made two major observations.

First, when it comes to sub-optimal organisational patterns as well as *social debt*, agile methods differ little with respect to classical organisational structures for software engineering. Indeed, the smells we report in the context of this study are more or less identical, and occur almost as frequently as the community and organisational structure smells reported in prior research [6, 22, 5]. This could indicate a lack of maturity around measuring and managing *social debt* all across software engineering practices and methods. To this end, further research should be invested in understanding and measuring the phenomenon to encourage its management via quantitative and operationalized tooling.

Second, software architectures, even though often not explicitly addressed in the scope of most agile methods, are proxies for community smells and could be therefore used as ways to understand social debt and community smells as well as offering ways to mitigate the associated debt. For example, imagine re-modularising a software architecture to address

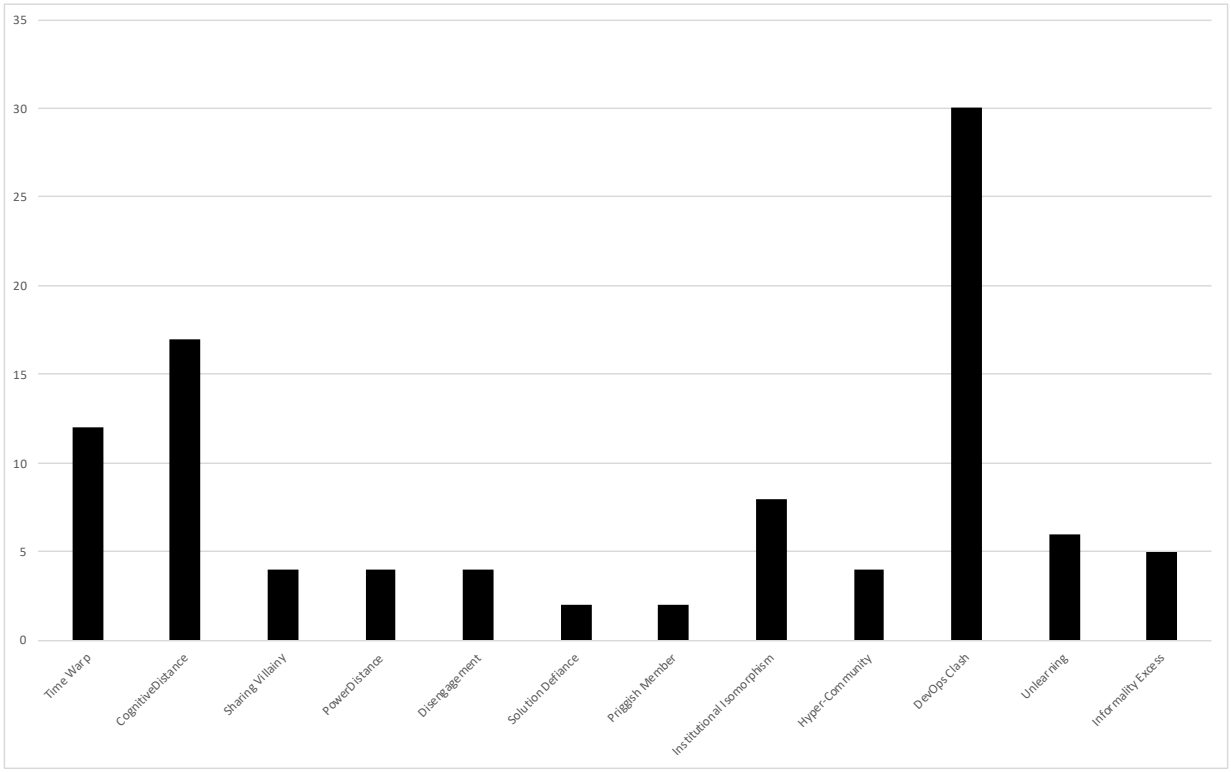

Figure 2. Occurrence of community smells across our sample.

teams' organisational differences; this circumstance is quite typical although little understood and even less used in the scope of agile teams. Further understanding and practical experimentation of the relationships between software architectures and the context of agile team governance could reveal vital insights into this relatively unexplored research area.

Third, our data suggests that the use of agile methods may in fact lead teams to develop a counterbalance between software architecture smells and community smells. For example, across our sample, a maximum of 76 months was reported for the use of agile methods — in the scope of those teams, practitioners equally diverged towards reportedly formally- and informally-structured ways of working which correspond to conflicting amounts of community smells. This reflection indicates that further study should be devoted to: (a) understanding the role of community smells in the context of agile methods; (b) understand and rank the influence and costs around community smells in that context; (c) anticipate and classify the organizational changes. These insights may prove valuable in steerint young agile teams towards organizational stability [4].

5.2. Usage of Findings and Practical Impact

The practical impact of the findings in this paper is threefold.

First, practitioners can use the reported occurrence of community smells as an analysis lens, e.g., in the scope of their *agile retrospective meetings*, to understand whether certain smelly conditions are manifesting themselves, or whether the preconditions and causes of those sub-optimal patterns may indeed be emerging.

Second, a smell is not a bad thing per se, but could cause distress in the organization that may need further attention. In this respect, *product owners* may use the reported community smells and their overlap with architecture issues to try and diagnose any reported sub-optimal quality in the emerging software architecture of software products. Similarly, *scrum masters* may be able to harness the knowledge around community smells and their relations with architecture smells to possibly govern and help fight the occurrence of undesired community smells as counterparts to product owner using the smells the other way around, that is, anticipating the emergence of architecture smells by witnessing and governing on the occurrence of software community smells.

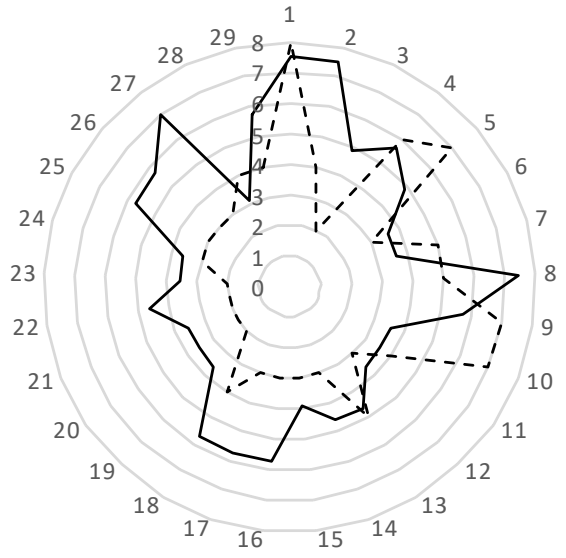

Figure 3. Community smells diffuseness vs. software architecture issues across our sample - a radial overlap diagram; scales and outline are identical to Fig. 1.

Finally, *agile team members* can use the correlations and analyses we found in the scope of this study to drive their *iteration-0* software architecture bootstrap. This phase is a delicate inception activity where the team decides what to do (in terms of structuring the system), how to do it, and by whom. Knowing the organisational consequences and relations around software architecture smells may aid team members troubleshooting their iteration-0 software architecture design activities.

5.3. Impacts on Further Research

From a research perspective, the findings and contributions of this paper are twofold.

On one hand, our focus on community smells in the scope of agile teams provides even more evidence of the presence and impact of community smells such that future researchers may understand, better characterise, and quantify the phenomena.

On the other hand, researchers actively pursuing an understanding of software architecture smells and technical debt can use our findings to highlight where those smells are coming from, and to what extent they weigh on the organisational structures as well. This insight is critical to provide metrics and automated tools for the simultaneous management of both social and technical debt.

Furthermore, our sampling criteria were reduced to a controllable minimum for a study of this scope. However, further study should be devoted to explore any interconnections between, e.g., codebase-size, team-size, team-age, process maturity and other team- and organizational-structure related factors. The previous factors might provide further lens for analysis with respect to the results reported in this manuscript.

5.4. Threats to Validity

Like any study of comparable magnitude and scale, this study is affected by several threats to validity [31]. In what follows we outline the major ones in our study design and execution.

5.4.1. Internal and Sampling Validity Internal validity refers to the internal consistency and structural integrity of the empirical research design. More specifically it refers to how many confounding factors may have been overlooked. For example, the patterns we report cover 24 out of 30 datapoints, with 6 outliers that were ultimately discarded. This issue may negatively affect the generalisability of our results since those datapoints may reflect unknown community types or characteristics and quantities that emerge with a more fine-grained analysis lens (e.g., focusing on single software artefacts and the communication/collaboration around them). We attempted to address this threat with a sampling strategy where we controlled as many variables as possible to: (a) ensure a meaningful variety of our sample; (b) ensure that important variables for the objects of our study, namely organisational structure and architecture, were controlled. For example, we controlled for organisational maturity influences over organisational structures by selecting agile teams that had recently successfully adopted agile methods. Also, we controlled for process maturity, by selecting a heterogeneous sample according to a standard CMMI scale. This notwithstanding, there are up to 90 factors from the state of the art in organisations research [4] that may still be affecting our findings and results, also, the quantities and effect size of the factors themselves were not addressed in this study at all. Stemming from this major limitation, we are planning further study of our target subjects in follow-up quantitative research that may lead to confirming the validity of this work. Furthermore, the statistical validity of the results reported in this manuscript relies heavily on multiple comparisons among possibly related quantities, which leads to a statistical analysis issue known as the “Multiple Comparisons Problem” [32] — this condition is typically addressed by increasing sample size and correcting with the well-known (but overly-conservative) Bonferroni correction [33]. Because our study was exploratory, we did not perform

oversampling or apply the afore-mentioned correction so our findings remain potentially affected by this issue. In the future we plan to replicate this work in an attempt to further generalise the findings and increase their validity. In this new version of the study we are planning to design the analyses for hypothesis testing, factoring in the possible control operations entailed by tests such as the Bonferroni correction.

5.4.2. Construct and External Validity As previously discussed, our decision to strengthen internal and sampling validity by focusing on agile teams alone inherently introduced a flaw in our construct validity. Also, although our measurements, observations, and findings are based on valid content (i.e., reported by practitioners who were directly involved with and witnesses to the reported subjects) and valid criteria, the external validity connected to the above-mentioned flaw may be compromised as well. For example, we used simple non-weighted and aggregate sums to evaluate the quantities involved in this study so we have no way of knowing whether the entity and *-arity* of the involved quantities may yield different results. We are planning further studies using a more quantitative approach including rigorous statistical modelling and testing.

5.4.3. Conclusion Validity Conclusion validity represents the degree to which conclusions about the relationship among variables are reasonable. In the scope of the discussions of our results we made sure to minimise possible interpretations, designing the study with reference to known hypotheses. Also, our conclusions were drawn from statistical analysis of our dataset. This not withstanding, the conclusions drawn from our study also reflect assumptions which, although sound and reflecting the need to avoid research design mishaps, may compromise the conclusion validity. For example, we chose to focus on the most basic roles for software practitioners in our sample, not reaching out explicitly, for example, to software architects. This decision may have compromised our ability to go beyond the simple numbers and capture speculative interpretations of architects who work on multiple projects, possibly within different organizations. Thus, our study remains subject to this threat.

6. Related Work

To the best of our knowledge, little or no work is available in the way of finding community smells in the scope of agile software engineering teams as well as correlating that occurrence with software architecture quality. However, several works address the sub-optimal organisational conditions that may emerge as connected to agile practice. For example, Meyer [34] explores systematically the role of agile practices as connected to software development and offers a birds-eye view over their organisational and technical effects. Furthermore, other research is available around sub-optimal organisational practices emerging during migration to agile methods. For example, Noordeloos et al. [35] study the effects of software migration processes from the Rational Unified Process (RUP) to the Scrum agile methodology and highlight the connected sub-optimal organisational circumstances. Similarly, Subhajit et al. [36] study the evolution of collaboration patterns in the scope of a large, globally-distributed agile project.

In summary, the literature that we found on subjects close to our own either deals with aspects loosely related to community smells or does not analyse such aspects in relation to the quality of software outputs in the study subjects. We sought to shed light in that exact condition, trying to highlight the possible consequences and patterns thereof.

7. Conclusions and Future Work

This paper offers an empirical, large-scale, longitudinal study over the relations between (1) community smells, that is, nasty sub-optimal patterns of organisational and socio-technical behaviour during software engineering and (2) software architecture smells, that is, nasty sub-optimal patterns of architecture elements and structures. We operationalized our study in the scope and *context* of agile teams, where software architectures are often not explicitly designed, documented, or maintained and where the organisational structure is emergent.

We found that community smells are indeed a force to be reckoned with in the target context: smells are diffused, and they impact heavily on the quality of involved software artefacts.

We conclude that: (1) community smells should be better studied to understand their birth, growth, measurement, and mitigations; (2) agile practitioners can use them, e.g., in the scope of their retrospectives or their iteration-0 planning as rules to be avoided, i.e., "what not-to-do".

In the future we are planning several replications of this study to address the scope and threats to validity we highlighted in the previous sections. For example, we hope to provide operationalisations for the community smells we revealed such that a quantitative approach can be taken.

Acknowledgements

Some of the authors' work is partially supported by the European Commission grant no. 0421 (Interreg ICT), Werkinzicht and the European Commission grant no. 787061 (H2020), ANITA.

Replication Package

The data and all materials used in the scope of preparing this paper are available online to encourage verifyability of the findings as well as replication of our study. Link: `anonymized`.

References

[1] M. P. Boerman, Z. Lubsen, D. A. Tamburri, and J. Visser, "Measuring and monitoring agile development status.," in *WETSoM@ICSE* (R. Tonelli, E. D. Tempero, S. Counsell, and C. A. Visaggio, eds.), pp. 54–62, IEEE Copmuter Society, 2015.

[2] P. Diegmann, "Measuring the effect of team diversity and collective intelligence in agile teams on software development efficiency.," in *AMCIS*, Association for Information Systems, 2017.

[3] M. E. Conway, "How do committees invent," *Datamation*, vol. 14, no. 4, pp. 28–31, 1968.

[4] D. A. Tamburri, P. Lago, and H. van Vliet, "Organizational social structures for software engineering.," *ACM Comput. Surv.*, vol. 46, no. 1, p. 3, 2013.

[5] D. A. Tamburri, R. Kazman, and H. Fahimi, "The architect's role in community shepherding.," *IEEE Software*, vol. 33, no. 6, pp. 70–79, 2016.

[6] D. A. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "Social debt in software engineering: insights from industry.," *J. Internet Services and Applications*, vol. 6, no. 1, pp. 10:1–10:17, 2015.

[7] D. Tamburri, P. Kruchten, P. Lago, and H. van Vliet, "What is social debt in software engineering?," in *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*, pp. 93–96, May 2013.

[8] C. de O. Melo, D. S. Cruzes, F. Kon, and R. Conradi, "Agile team perceptions of productivity factors.," in *AGILE*, pp. 57–66, IEEE Computer Society, 2011.

[9] C. Zannier, H. Erdogmus, and L. Lindstrom, eds., *Extreme Programming and Agile Methods - XP/Agile Universe 2004, 4th Conference on Extreme Programming and Agile Methods, Calgary, Canada, August 15-18, 2004, Proceedings*, vol. 3134 of *Lecture Notes in Computer Science*, Springer, 2004.

[10] T. Sharma and D. Spinellis, "A survey on software smells.," *Journal of Systems and Software*, vol. 138, pp. 158–173, 2018.

[11] G. Samarthyam, G. Suryanarayana, and T. Sharma, "Refactoring for software architecture smells.," in *IWoR@ASE* (A. Ouni, M. Kessentini, and M. . Cinnide, eds.), pp. 1–4, ACM, 2016.

[12] A. Elgebeely and A. Kamel, "Architecture and technical debt agile planning methodology for software production," *Computer Science & Information Technology (CS & IT)*, vol. 7, pp. 15 – 21, Dec. 2017.

[13] A. MacCormack and D. J. Sturtevant, "Technical debt and system architecture: The impact of coupling on defect-related activity.," *Journal of Systems and Software*, vol. 120, pp. 170–182, 2016.

[14] K. Schwaber and M. Beedle, *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice Hall, 2002.

[15] C. Baham, "The impact of organizational culture and structure on the routinization of agile software development methodologies.," in *AMCIS*, Association for Information Systems, 2016.

[16] J. Iivari and N. Iivari, "Organizational culture and the deployment of agile methods: The competing values model view.," in *Agile Software Development* (T. Dingsyr, T. Dyb, and N. B. Moe, eds.), pp. 203–222, Springer, 2010.

[17] M. Cataldo, J. D. Herbsleb, and K. M. Carley, "Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity.," in *ESEM* (H. D. Rombach, S. G. Elbaum, and J. Mnch, eds.), pp. 2–11, ACM, 2008.

[18] J. D. Herbsleb, M. Cataldo, D. Damian, P. T. Devanbu, S. M. Easterbrook, and A. Mockus, "Socio-technical congruence (stc 2008).," in *ICSE Companion* (W. Schfer, M. B. Dwyer, and V. Gruhn, eds.), pp. 1027–1028, ACM, 2008. 978-1-60558-079-1.

[19] J. D. Herbsleb, A. Mockus, T. A. Finholt, and R. E. Grinter, "An empirical study of global software development: distance and speed," in *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, (Washington, DC, USA), pp. 81–90, IEEE Computer Society, 2001.

[20] J. Herbsleb and R. Grinter, "Architectures, coordination, and distance: Conway's law and beyond," *Software, IEEE*, vol. 16, pp. 63 –70, sep/oct 1999.

[21] I. Kwan, A. Schrter, and D. Damian, "Does socio-technical congruence have an effect on software build success? a study of coordination in a software project.," *IEEE Trans. Software Eng.*, vol. 37, no. 3, pp. 307–324, 2011.

[22] D. A. Tamburri, P. Lago, and H. van Vliet, "Uncovering latent social communities in software development," *IEEE Software*, vol. 30, pp. 29 –36, jan.-feb. 2013.

[23] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. SEI series in software engineering, Addison-Wesley, 2012.

[24] T. Gnatzy, *Delphi studies and scenario planning: methodological advancements and cases*. PhD thesis, 2011.

[25] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, and B. Regnell, *Experimentation in Software Engineering*. Springer, 2012.

[26] B. Meyer, *Agile!* Cham: Springer, 2014.

[27] CMMI Product Team, *CMMI for Development*. Software Engineerung Institute, Carnegie Mellon University, 1.3 ed., 2010.

[28] T. O. Group, "Soa source book," *http://www.opengroup.org/projects/soa-book/*.

[29] 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 International Conference on Software Engineering 2015*, 2015.

[30] B. Ratner, "The correlation coefficient: Its values range between +1/1, or do they?," *Journal of Targeting, Measurement and Analysis for Marketing*, vol. 17, pp. 139–142, Jun 2009.

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering: an introduction*. Norwell, MA, USA: Kluwer Academic Publishers, 2000.

[32] A. Gelman and E. Loken, "The garden of forking paths: Why multiple comparisons can be a problem, even when there is no ?fishing expedition? or ?p-hacking? and the research hypothesis was posited ahead of time," 2013.

[33] A. Terada and J. Sese, "Bonferroni correction hides significant motif combinations.," in *BIBE*, pp. 1–4, IEEE Computer Society, 2013.

[34] B. Meyer, *Agile!* Cham: Springer, 2014.

[35] R. Noordeloos, C. Manteli, and H. van Vliet, "From rup to scrum in global software development: A case study.," in *ICGSE*, pp. 31–40, IEEE Computer Society, 2012.

[36] S. Datta, R. Sindhgatta, and B. Sengupta, "Evolution of developer collaboration on the jazz platform: a study of a large scale agile project," in *Proceedings of the 4th India Software Engineering Conference*, ISEC '11, (New York, NY, USA), pp. 21–30, ACM, 2011.