Refactoring Community Smells in the Wild: The Practitioner's Field Manual

Gemma Catolino
University of Salerno
Fisciano, Italy
gcatolino@unisa.it

Fabio Palomba
University of Salerno
Fisciano, Italy
fpalomba@unisa.it

Damian A. Tamburri
Jheronimus Academy of Data Science
s'Hertogenbosch, The Netherlands
d.a.tamburri@tue.nl

Alexander Serebrenik
Eindhoven University of Technology
Eindhoven, The Netherlands
a.serebrenik@tue.nl

Filomena Ferrucci
University of Salerno
Fisciano, Italy
fferrucci@unisa.it

ABSTRACT

Community smells have been defined as sub-optimal organizational structures that may lead to social debt. Previous studies have shown that they are highly diffused in both open- and closed-source projects, are perceived as harmful by practitioners, and can even lead to the introduction of technical debt in source code. Despite the presence of this body of research, little is known on the practitioners' perceived prominence of community smells in practice as well as on the strategies adopted to deal with them. This paper aims at bridging this gap by proposing an empirical study in which 76 software practitioners are inquired on (i) the prominence of four well-known community smells, *i.e.*, Organizational Silo, Black Cloud, Lone Wolf, and Radio Silence, in their contexts and (ii) the methods they adopted to "refactor" them. Our results first reveal that community smells frequently manifest themselves in software projects and, more importantly, there exist specific refactoring practices to deal with each of the considered community smells.

KEYWORDS

Community Smells; Social Debt; Empirical Software Engineering.

ACM Reference Format:
Gemma Catolino, Fabio Palomba, Damian A. Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2018. Refactoring Community Smells in the Wild: The Practitioner's Field Manual. In *ICSE 2020: IEEE/ACM International Conference on Software Engineering, May 23–29, 2020 - Seoul, South Korea.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Social aspects of software engineering, like communication among developers or their coordination, may have a substantial impact on project's success [24, 44], especially because developers are often separated by physical and cultural distance [22, 30, 48], expertise or power difference [21, 28, 40, 47]. The sum of negative and unforeseen costs/consequences given by sub-optimal social relations among developers has been named *community smells*, namely organizational and socio-technical situations that hamper or altogether impede the straightforward production, operation, maintenance, and evolution of software [41, 52, 58]. As a matter of fact, community smells represent one of the main causes of social debt [54] and, as such, threaten the entire management of software systems since they may substantially decrease the level of trust within a community or the degree to which it is immature or unable to tackle a certain development problem [37].

Recently, community smells have received growing attention from the research community. Researchers have studied perceived harmfulness of community smells [9, 58], their relation to socio-technical metrics [57, 58] (*e.g.*, socio-technical congruence [7]), and their negative impact on technical aspects of software systems [41]. At the same time, research effort has been devoted to the definition of methods to reduce the likelihood of their introduction [8, 10] as well as automated approaches that can detect them [58] or exploit them in the context of software evolution [41].

Nonetheless, there is still a notable lack of studies targeting the problem of removing community smells from software communities. In particular, little is known on the strategies adopted by practitioners to deal with the presence of community smells: We refer to these strategies as "refactoring", inspired by the name coined by Fowler for removal of code smells [16].

In this paper, we aim at bridging this gap of knowledge by presenting an empirical study aimed at eliciting refactoring operations applied by practitioners in presence of four well-known community smells, namely Organizational Silo, Black Cloud, Lone Wolf, and Radio Silence. We design an online survey, recruit 76 practitioners, and inquiry them on (i) the relevance of community smells in their context and (ii) the corrective actions performed to remove them. Key results show how, according to the practitioner's viewpoint, all investigated community smells typically affect software communities having a larger amount of people and, in particular, Lone Wolf and Black Cloud are the smells that more frequently arise in industry. Perhaps more importantly, our analysis enables the definition of a taxonomy of refactoring strategies to deal with those community smells. In particular, common strategies for all the analyzed community smells rely on mentoring, the creation of a communication plan, and restructuring the community. Such

ICSE 2020, May 23–29, 2020, Seoul, South Korea
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-9999-9/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

a taxonomy can be a valuable instrument for project managers, to learn how to increase the health of development communities, researchers, to investigate how to support practitioners with refactoring of community smells, and developers themselves, to improve their communication.

To summarize, this paper provides three main contributions:

(1) Empirical evidence of the practical relevance of all community smells investigated from the practitioner's perspective;
(2) An empirically grounded taxonomy of refactoring strategies suitable to deal with the four community smells considered;
(3) A publicly available replication package [11] reporting the data collected in our study, the categorization analysis and the structure of the survey which may be useful for researchers to want to replicate the study or explore the problem of community smell refactoring using a practitioner's perspective.

Structure of the paper. In Section 2 we report the design of the empirical study and the way we attracted participants, while Section 3 analyzes the results achieved. Practical implications of our findings are discussed in Section 4 and the threats to validity in Section 5. Section 6 overviews the related literature, explaining how our work extends the state of the art. Finally, Section 7 concludes the paper and outlines our future research agenda on the topic.

2 RESEARCH METHODOLOGY

The *goal* of the empirical study is twofold: on the one hand, it aims at understanding the practitioners' perception with respect to the relevance of community smells in practice; on the other hand, it aims to elicit a set of strategies that practitioners adopt in practice to mitigate and remove community smells. The *purpose* is to (i) gather the relevance of community smells in practice and (ii) understand how community-related problems are currently mitigated. The *perspective* is of both software engineering researchers and project managers. The former are interested in understanding how relevant are community smells for practitioners and, perhaps more importantly, what kind of mechanisms can be leveraged to support developers when mitigating or even removing community smells. The latter are interested in learning what is the prominence of community smells in other environments as well as how to resolve them based on the successful stories of other project managers who have dealt with community smells in the past.

2.1 Research Questions

To address the goals of our study, we design two main research questions (**RQ**s). In the first place, we assess the relevance of community smells from the practitioners' perspective with the aim of understanding their viewpoint on the actual prominence of the problem. This leads to our first research question:

RQ$_1$. *What is the relevance of community smells in practice?*

In order to address the first research question, we took into account the experience of practitioners, defining an online survey asking whether situations related to the presence of community smells happened to them. After studying their perceived relevance, we then focus on how practitioners get rid of community smells and, particularly, which are the refactoring strategies they use. For this reason we set ourselves the following research question:

RQ$_2$. *What are the refactoring strategies applied by practitioners to deal with community smells?*

Based on the answers of our participants we defined a taxonomy of refactoring strategies to mitigate community smells. In the next subsections, we describe objects and subjects of the study, *i.e.*, the specific community smells and professional developers taken into account, as well as the methodology adopted when designing and analyzing the survey study.

2.2 Objects of the Empirical Study

The *objects* of the study are represented by a set of known community smells for which we want to study their relevance and refactoring strategies. We select four community smell types: Organizational Silo, Black Cloud, Lone Wolf, Bottleneck or "Radio-silence". We opt for these smells since they have been shown to have a strong negative impact on both social aspects of software communities [54, 55] and the resulting technical processes and products [9, 10, 41]: they are, hence, critical forms of community smells to deal with. Moreover, the relevance of these community-related issues has been previously investigated in open-source [58] and, thus, our study with industrial practitioners can be used to confirm or refine previous findings. We follow the definition provided by Tamburri *et al.* [58] to describe each smell type:

(1) Organizational Silo Effect: This form of social debt refers to the presence of siloed areas of the developer community that do not communicate, except through one or two of their respective members;
(2) Black Cloud Effect: This community smell reflects an information overload due to lack of structured communications or cooperation governance;
(3) Lone Wolf Effect: This smell appears in cases where the development community presents unsanctioned or defiant contributors who carry out their work with little consideration of their peers, their decisions and communication;
(4) Bottleneck or "Radio-silence" Effect: This is an instance of the "unique boundary spanner" [63] problem from social-networks analysis: one member interposes herself into every formal interaction across two or more sub-communities with little or no flexibility to introduce other parallel channels.

2.3 Subjects of the Empirical Study

The *subjects* of our study are professionals having a solid experience in software project development and management. Indeed, to acquire significant data on the relevance and refactoring strategies adopted when dealing with community smells, we have to recruit people that naturally have a broader view of a software system, its management, and its social activity with respect to individual developers who may be just focused on specific sub-teams.

For this reason, we proceeded with a convenience sample recruitment strategy [45]. Rather than spreading our survey over social networks (*e.g.*, Twitter), we first sent it out to a selection of our personal contacts (*e.g.*, former industrial colleagues, friends of ours working as project managers in industry). In the second place, we involved project manager's associations, like the *Project Management Institute*[1] and similar.

Unfortunately, we cannot have a definitive answer with respect to the total number of invitations sent. This because some of the project managers contacted have subsequently privately shared the survey with further colleagues, thus making an estimation hard to do. In any case, we finally collected 76 fully compiled questionnaires that we used to address our research questions.

As for the characterization of our sample, 43% of the respondents are women: in our case, this is an extremely relevant factor since people of different genders may have different ways of solving communication/coordination problems [5, 64] and a more balanced participation allows us to (i) draw more generalizable conclusions and (ii) assess whether gender influences the application of certain refactoring strategies. Furthermore, our participants have high experience when it comes to project management (65% of them evaluate themselves as highly or very highly experienced) and software development (81% rated themselves as highly experienced): this confirms that our recruitment strategy has been pretty successful, since we could attract the practitioners we were actually looking for. More than 30% hold a management position, 28% are professional developers while the rest report such jobs as consultant, software analyst, software architect and SCRUM master. Finally, 38% of the respondents currently work in a large company (more than 2,000 employees/contributors) and 45% work within smaller teams composed of 5 to 10 developers.

From these basic descriptive statistics, we can claim that the opinions collected are likely to provide us with reliable insights into community smells in practice and their corrective strategies.

2.4 Collecting Practitioner's Opinions

To collect the opinions of our participants, we design an online survey composed of five main sections. Following the guidelines of Flanigan *et al.* [15], we consciously keep the survey anonymous, short, and prevent our influence on the answer: when preparing the survey, we estimated a completion time of ≈10 minutes.

The first four sections of the survey describe a vignette-based scenario [14] corresponding to the community smell. For example, for Black Cloud the survey contains the following scenario:

> *"Suppose your development team is working on the definition of a web-based application for the scheduling of resources. During the development, you recognize that the community members suffer of an information overload due to lack of structured communication (*e.g.*, communications among team members are not performed over official channels)."*

We analyze four community smells and have one scenario for each section—the remaining scenarios can be found in the online appendix [11]. We stress that, as not to bias the participants we never explicitly mentioned that we were studying the problem of community smells. After showing the participants a scenario, we ask two questions:

(1) *Has this situation ever happened to you?*
(2) *If you found it problematic, how did you deal with it?*

The first question can be answered *'yes'* or *'no'*. The second one is an open question that practitioners can answer by writing down their opinions and experience with respect to the specific strategies adopted to deal with community smells.

The last section of the survey is finally reserved to demographic information. We include questions related to gender, job, programming/management experience as well as about the size of the company and their team. We implement the survey as a Google form.

2.5 Data Analysis

To analyze the data coming from the practitioners' answers to our survey, we first count how many times they answered *'yes'* to the first question: this allows us to address **RQ**$_1$ and have a measure of how relevant the considered community smells are from the practitioners' viewpoint.

As for **RQ**$_2$, we apply Straussian Grounded Theory [12] to analyze the practitioners' open answers: this is a systematic methodology that involves the construction of theories through methodical gathering and analysis of data. In our case, this methodology is required given the exploratory nature of the open question asked to practitioners. Furthermore, the selection of a Straussian Grounded-Theory methodology is driven by one main reason: it does not assume the presence of any previous theory to be tested over the data but rather it adopts a constructivist theory-proving approach wherefore a theory is directly and purely generated from the data.

We conduct the data analysis as follow:

Microanalysis. In the first place, one of the authors of this paper (*i.e.*, hereafter, the *inspector*) labels survey responses by applying a single label per every piece of text. In particular, the inspector splits sentences using standard text separators (*e.g.*, commas or semicolons) and then assigns initial labels based on the content of the text. Subsequently, other three inspectors validate the initial labels and provide suggestions on how to improve them, *e.g.*, if it makes sense to split one of them or aggregate some. After this first step, the inspectors compute the inter-rater agreement using the Cohen's k coefficient [2]: with respect to other measurements, this is considered to be a more robust statistical test since it takes into account the possibility that an agreement is occurred by chance [2]. The k coefficient measure 0.49, which can be interpreted as *moderate* [2].

Categorization. In a second phase, the feedback coming from the first step is taken into account by the first inspector in order to cluster labels which are semantically similar or even identical. To this aim, the inspector applies the semantic similarity principle [25]. The result of this step consists of the renaming of labels to better reflect the various categories identified.

Saturation. The main inspector, together with the other three, iterate over the labels assigned so far until they can reach an agreement with respect names and meanings of all of them. These results in a theoretical saturation [62], namely the phase in which the analysis of the labels does not propose newer insights and all concepts in the theory are well-developed.

[1]https://www.pmi.org

Taxonomy building. Finally, based on the labels assigned to the practitioners' answers, we can proceed with building a taxonomy of refactoring strategies for each community smell considered in the study. In Section 3 we represent and discuss the tree of categories and labels that allow us to visualize the grounded-theory extracted from our survey responses.

3 ANALYSIS OF THE RESULTS

In this section, we report and discuss the results of our study, by considering each research question independently.

3.1 RQ$_1$. What is the relevance of community smells in practice?

To assess relevance of community smells to industrial practice we have asked the respondents whether they have experienced the situation described in the vignette corresponding to the smells.

The most commonly reported smell among the four we have considered is the Lone Wolf: 66% (50/76) of the respondents reported having encountered this smell in practice. This finding concurs with the previous observation that this smell is also very common in open-source communities and is seen by developers as an important source leading to social debt [58].

The second most commonly recognized community smell is the Black Cloud, reported by 57% (43/76) of the respondents. Importance of this smell for the industrial Integra project has been previously observed by Tamburri *et al.* [56], while a more recent interview study of open-source developers indicates that all interviewees recognize presence of Black Cloud instances in their communities [58]. Thus, our findings confirm and reinforce previous results on the relevance of this smell in practice.

Organisational Silo has been experienced by almost half of the respondents (49%, 37/76), while Radio Silence by more than a third of them (35%, 27/76). Hence, we can also argue that these last two smells are diffused in practice, triangulating what has been recently found on their diffuseness in open-source projects [58].

To broaden the scope of the analysis, we also considered whether certain types of community smells are more prominent in larger or smaller communities. As expected, all the smells appear more frequently in larger companies: indeed, 57% of the respondents who experienced a community smell (43/76) belong to companies having between 250 and 2,000 employees. Similarly, 63% of them (48/76) reported to be in larger teams composed by 10 to 20 people. Specifically, we observed that Organizational Silo and Black Cloud are most often reported by participants working in larger companies, while Radio Silence and Lone Wolf are equally prominent independently from the size of companies. Also in this case, our findings concur with the results reported previously [58]. To statistically verify the relation between larger companies/teams and presence of community smells, we ran the Pearson's Chi-square Test for Independence [36], which in our case indicates how expectations (*i.e.*, the distribution of sizes of companies/teams) compare to actual observed data (*i.e.*, presence of specific types of community smells). As expected, the test indicated a p-value lower than 0.05, thus indicating a statistical significance of our observations, for the relations that larger companies and teams have with the emergence of Organizational Silo and Black Cloud, while the p-value measured 0.16 and 0.23 when considering size of companies/teams with Radio Silence and Lone Wolf, respectively.

Finding 1*. According to the collected opinions, Lone Wolf and Black Cloud are the community smells that more frequently arise in industry. Nevertheless, the prominence of Organizational Silo and Black Cloud is not negligible, especially in larger software communities.*

3.2 RQ$_2$. What are the refactoring strategies applied by practitioners to deal with community smells?

To address our second research question we asked participants to discuss the strategies they adopted when dealing with community smells. Next, we discuss the refactoring strategies for each community smell. Note that, when analyzing our data, we did not observe differences in the answers provided by men and women.

3.2.1 *Organizational Silo.*

Table 1 summarizes the refactoring strategies discussed by participants when considering the Organizational Silo smell. 35 practitioners (out of the total 37 who experienced this smell) reported seven different refactoring actions.

Restructure the community. The first and most popular refactoring (13 mentions) is that of restructuring the community with the aim of improving communication among team members. According to our participants, this may happen in different ways. 7 practitioners claimed that they solved the problem by changing the composition of sub-teams and, particularly, by organizing them so that the heterogeneity increases and the communication improves. For instance, participant #19 reported that:

> *"[I solved the issue by] organizing the sub-teams heterogeneously (including all levels of seniority in each of them, for example juniors and experts)."*

As shown, the practitioner tried to put novice programmers with experts that can advise them, possibly improving the way the entire sub-team communicates. Other three practitioners got rid of the community smells by splitting large teams in smaller ones, thus increasing the chance team members have to communicate with each other. Finally, the last three practitioners followed a "continuous" approach in which team members are swapped periodically, for instance every task or sprint.

Create communication plan. Seven practitioners declared that a good way to remove Organizational Silo instances is to create a detailed communication plan in which the appropriate channels and protocols are discussed and put in place. According to them, in cases where all the communication channels are properly working and delivering efficiently messages, the problems related to the presence of the smell are generally well mitigated - as reported, for example, by participant #41.

Mentoring. The third most common solution to this community smell (7 mentions) is mentoring. In particular, practitioners explained that communication problems can be solved by project

Table 1: Refactoring strategies for the Organizational Silo smell.

Strategy	Definition	#
Restructure the community	Organizing sub-teams heterogeneously, splitting larger teams in smaller ones, swapping team members every task/story/sprint.	13
Create communication plan	Creating appropriate communication channels and protocols.	7
Mentoring	Talking with the team members, trying to solve communication problems.	7
Cohesion exercising	Doing exercises in order to improve the cohesion of the team (*e.g.*, brainstorming, number of meetings).	4
Monitoring	Monitoring closely how team members communicate and share information.	2
Introduce a social-rewarding mechanism	Providing benefits to team members who work and communicate appropriately. (*e.g.*, credits)	1
Not relevant	Opinions not related to refactoring strategies.	3

managers by talking to developers and being sure that everyone is taught on the basic rules to share information about team members and communicate with the rest of the community.

Less common strategies. Besides the three most common refactoring actions discussed so far, a smaller number of practitioners mentioned alternative strategies. Specifically, four of them explained that exercising the cohesion of the community may represent an important factor to mitigate the Organizational Silo smell: this includes, for example, a higher number of meetings aimed at increasing communication. Other strategies refer to the monitoring of the developer's discussions (2 mentions), and providing rewarding mechanisms for the developers who communicate well (1 mention). Finally, the last three practitioners expressed opinions that did not refer to solutions, but rather just better explained that problems connected to the presence of the community smell.

3.2.2 *Black Cloud.*

Table 2 lists the refactoring strategies for the Black Cloud smell. In this case, practitioners reported a lower number of actions when compared to the other smells:

Create communication plan. 80% of the participants who suggested how to remove this smell (28/34) reported that the creation of a good communication plan and revise it periodically represents the best solution to mitigate the lack of structured communication. For example, practitioner #2 reported that Black Cloud instances may depend on size and experience of the development community and the only way to remove them is to define a communication plan in advance and keep revising it in order to take it up to date.

Less common strategies. The last six practitioners who commented on the refactoring of this smell referred to two alternative methods. In the first place, five of them mentioned that restructuring the community, *e.g.*, by splitting large teams into smaller ones or defining more precise/structured rules for communication, mitigated the effects of the smell in their experience. Finally, one practitioner referred to the introduction of a social sanctioning mechanism: this is a strategy in which members of the community that do not respect the common rules defined in the communication plan are penalized and obliged to attend mentoring sessions with senior developers or project managers.

3.2.3 *Radio Silence.*

The refactoring strategies elicited for this smell are summarized in Table 3. Here 19 participants commented on their experience and presented six different actions.

Mentoring. One of the most cited strategy is that of mentoring. Five practitioners reported that to mitigate the presence of a team member that interposes him/herself into every formal communication is to let the project manager of other senior developers to mentor him/her. In this way, the bottleneck would gradually change his/her behavior, leading to a healthier development community.

Cohesion exercising. Five practitioners also reported that putting in place activities that increase the cohesion of the community may be a useful mechanism to mitigate the presence of the smell and improve the communication among team members. According to our participants, a higher cohesion would help team members in sharing knowledge and exchange critical information, thus reducing the likelihood of having people that interpose themselves between sub-teams.

Create communication plan. The definition of rules for sharing knowledge and communicate with other team members represents the second most cited strategy to remove an instance of Radio Silence. As an example, participant #14 explained that *" educating developers in group work and proper communication management is always a necessary step to mitigate this problem"*.

Less common strategies. Seven participants proposed alternative methodologies. These include a restructuring of the community aimed at changing role to the bottleneck or monitoring closely how s/he communicates with other people. Finally, one practitioner mentioned that the introduction of a social sanctioning mechanism led to the removal of the problem.

3.2.4 *Lone Wolf.*

Table 4 overviews the refactoring strategies described by the 50 practitioners who commented on the removal of this smell. The list includes 5 main actions:

Mentoring. This is the most popular strategy by far (42% of participants). According to the opinions left by practitioners, mentoring and helping the lone wolf to better communicate and work with other people represents a key action to mitigate the negative

Table 2: Refactoring strategies for the Black Cloud smell.

Strategy	Definition	#
Create communication plan	Creating appropriate communication channels and protocols.	28
Restructure the community	Organizing sub-teams heterogeneously, splitting larger teams in smaller ones, swapping team members every task/story/sprint.	5
Introduce a social sanctioning mechanism	Motivating people to act as expected.	1

Table 3: Refactoring strategies for the Radio Silence smell.

Strategy	Definition	#
Cohesion Exercising	Doing exercises in order to improve the cohesion of the team (*e.g.*, brainstorming, number of meetings).	5
Mentoring	Talking with the team members, trying to solve communication problems.	5
Create communication plan	Creating appropriate communication channels and protocols.	4
Restructure the community	Organizing sub-teams heterogeneously, splitting larger teams in smaller ones, swapping team members every task/story/sprint.	4
Monitoring	Monitoring closely how team members communicate and share information.	2
Introduce a social sanctioning mechanism	Motivating people to act as expected.	1

Table 4: Refactoring strategies for the Lone Wolf smell.

Strategy	Definition	#
Mentoring	Talking with the team members, trying to solve communication problems.	24
Restructure the community	Organizing sub-teams heterogeneously, splitting larger teams in smaller ones, swapping team members every task/story/sprint.	12
Monitoring	Monitoring closely how team members communicate and share information.	7
Cohesion Exercising	Doing exercises in order to improve the cohesion of the team (*e.g.*, brainstorming, number of meetings).	7
Introduce a social sanctioning mechanism	Motivating people to act as expected.	3

effects this smell can present. For example, participant #23 reported that s/he *"talked to the person to verify the reasons [behind his/her behavior] and understand if the problem can be solved"*.

Restructure the community. A second popular refactoring for this smell is the restructuring of the community (mentioned by 12 participants). In particular, the action concerns with making the lone wolf closer to the other developers of the community and able to communicate/work with them more effectively. For this reason, practitioners suggested to let him/her work in different sub-teams, so that the lone wolf is "enforced" to communicate more in order to be updated with the status of the module being developed by that sub-team.

Less common strategies. While the refactoring actions discussed above are by far the most widely used in practice, other practitioners presented different alternatives, all of them referred to the improvement of communication in different manners. First, an effective monitoring of how team members communicate has been mentioned by seven practitioners, who explained that keeping under control the way team members share information represents not only a method to avoid the emergence of the smell in the first instance, but also to gradually remove the effect of a lone wolf instance. Seven participants mentioned the involvement of the lone wolf in more social activities that can increase the overall cohesion of the community and the engagement of the lone wolf. Finally, three practitioners reported the introduction of social sanctioning mechanisms to deal with this smell.

Finding 2. *Practitioners mentioned the existence of refactoring strategies that increase cohesion of the development team. Common strategies include mentoring, creation of a communication plan, and restructuring the community.*

4 DISCUSSION AND IMPLICATIONS

Our findings provide a number of discussion points and implications for practitioners and research community.

4.1 Key Observations

The results suggest a consistent presence of refactoring strategies that practitioners are enacting. On the one hand, some of the aforementioned strategies are consistent with the state of the art in organizations and project management research. On the other hand, several of the strategies are *socio-technical*, namely they imply short- and longer-term modifications to the community structure as well as the technical structure of the software project.

From the perspective of the state of the art in organizations research as well as management research, we conducted a "loosely-structured focus group" with 3 senior researchers in both fields and identified several interesting contact points with that state of the art and the "refactoring" strategies therein; indeed, several strategies we identify have extremely valuable equivalents. For example, the cohesion-based strategies we identify reflect an interest in structural cohesion practices dating back to the *80's* (*e.g.*, see the work by Burt [6] or even more recent computational methods to measuring structural cohesion such as Lozano *et al.* [34]). Similarly, communications protocols and practices we identify based on such protocols have been studied in social-networks analysis literature, also through automated means [59]. For example, Sommerfeld *et al.* [50] studied the effect of social protocols to identify the effects (whether positive or negative) of indirect reciprocity, namely, the eventual consistency in the manifestation of a certain social relation between two or more parties and the costs connected to such eventuality. Furthermore, the effectiveness of some of the strategies we isolated is still well under debate [49, 61] in their own fields of research [49]. For example, the social sanctioning strategies we previously showcased have seen research in both social networks analysis and organizations research but the negative long-term effects of such sanctioning onto the social and organizational structure around specific products or artifacts [61] is still largely unknown [27] and, back to our software engineering domain, could very well be an underlying social phenomenon connected to the staggering numbers of turnover [46] in software engineering communities of practice (*e.g.*, open-source communities). Overall, this state of the art and informal comparison we operated strongly motivates us in furthering our knowledge around the practices we identified, the sub-optimal structures that they are designed to mitigate and the effects on the long-term sustainability of a software organization.

Furthermore, from the perspective of our current understanding over software processes and software engineering as a socio-technical system, the results of this study reflect a complex phenomenon with its own self-organizative, and emerging approaches to managing and addressing community sustainability [17]. What we remark, however, is a severe gap in the state of practice in terms of (i) software engineering education around such management structural issues and the strategies to mitigate them as well as (ii) industrial-strength practices and tools to measurably improve the short-, medium-, and long-term influences of the structural issues, on one side, and the strategies addressing them, on the other side. From an educational perspective, a quick glance at the current state of the art in software engineering education—as reflected by a quick 3-year review of the software engineering education track (SEET) at the International Conference on Software Engineering (ICSE)—is by far immature to address these gaps since the topic is touched upon only marginally and with little to no systematic studies as of yet. However, this constitutes an opportunity for the community to pursue such research with a more structured approach, possibly picking up from the results and challenges that this paper showcases.

It is important to note that we are aware that we need to triangulate our results *e.g.*, using semi-structured interviews; however, we consider these achievements as a first step to going deeper into the definition of the strategies, as also confirmed by our focus group.

4.2 Practical Impact

As for practitioners, the contents of this article serve as a showcase of what to do in case of "nasty" manifestation of phenomena consistent with community smells. It should be noted that smells themselves, be it in their social or technical manifestation, are not necessarily bad, but rather they constitute sets of phenomena to be taken into account while designing, developing, and operating software. That being said, the catalog we offer can be used by practitioners as a *cookbook* to structure their emergent approach at community management, possibly reusing and improving upon the strategies that we have elicited. From yet another perspective, that of open-source communities, the practices reported can be used as a basis to structure contribution policies or similar community sustainability and governance bylaws. With the ever-growing mass of open-source contribution, it is very common nowadays to see such community management and engagement protocols; the results we report could well be used to structure those protocols starting from a solid and proven basis. To gain evidence of this usefulness, we analyzed a selected sample of gray literature[2] discussing the phenomenon of open-source community forking—namely, when a community is split into two or multiple sub-communities pushing forward disjoint products and technical spaces—and its consequences along the lines of community survivability (*e.g.*, see the seminal work by Gameliesson *et al.* [18]). From this preliminary analysis it is clear that community smells are relevant factors in the phenomenon of community forking; in this scope, some of the strategies we have identified might have mitigated the consequences of the forking or aid in avoiding it altogether.

4.3 Relation with Previous Practices

Although the strategies we reported in this manuscript were elicited with rigorous empirical means and through the engagement of industrial stakeholders, the attempt at gathering socio-technical mitigation approaches is not novel per se. For example, as previously mentioned, previous work on the emerging role of community-shepherding software architects [53] has put forth a number of architecture-based strategies to refactor products (and the organizational structures around them) in line with mitigating community smells other than the ones we report in this paper. Examples of such strategies reflect the use of daily standups, architecture discussion groups or architecture knowledge-sharing practices. On

[2]http://tiny.cc/asrdfz

the one hand, the strategies in this manuscript can be seen as more generically-applicable and smell-specific approaches which do not directly relate to any one specific software or organizational artifacts. On the other hand, strategies reported previously have a domain-specific connotation, e.g., Noll *et al.* [38] work in the scope of general barriers to communication and collaboration in Global Software Engineering (GSE) scenarios or even earlier in time Joiner touches upon management smells akin to community smells specific to software reuse and strategies to reduce the connected risks [33]. Overall, the key message is that this organisational-strategic approach should become a practitioners' field manual in support of their organizational and socio-technical activities with structured and tested-true approaches.

5 THREATS TO VALIDITY

A number of threats might have influenced our findings. In this section, we summarize and explain how we mitigated them.

Threats to construct validity. Threats in this category refer to the relationship between theory and observation. In our study, this may mainly concern the way we have measured (i) how relevant community smells are in practice and (ii) what are the refactoring strategies associated to them. In the former case, we asked practitioners to answer through a *'yes'*/*'no'* option since we just wanted to inquiry them on whether certain situations actually happened to them. In the latter case, we let practitioners write down their opinions and experience: this was required to let them express their thoughts on the matter without biasing them with pre-defined answers. Moreover, we referred to community smells as "situations", thus did not bias the participants to believe that those represent a problem. In our future research agenda, we include a partial replication of our study aiming at measuring the experience of practitioners through semi-structured interviews, which can potentially help and complement the analysis of the refactoring strategies applied when dealing with community smells.

Threats to conclusion validity. As for the concerns between treatment and outcome, a major threat refers to the way we analyzed data coming from the practitioners' answers. To address $\mathbf{RQ}_1$, we computed basic statistics reporting the number of times they answered that community smells appeared in their experience. As for $\mathbf{RQ}_2$, we relied on a Straussian Grounded Theory [12] approach when analyzing the practitioners' open answers. As explained in Section 2, this research approach does not assume the presence of any previous theory to be tested over the data: this perfectly fits our goal of extracting knowledge on community smell refactoring starting from the practitioners answers. Nevertheless, we recognize that further replications of the empirical study might provide additional insights and stronger conclusion validity.

Threats to external validity. These mainly concern with the generalizability of the reported findings. In our case, we aimed at collecting opinions from practitioners having a solid background in both software development and management. This constraint naturally limits the survey population, *i.e.*, not all practitioners may be invited to answer our survey as they may have not the required experience to answer our questions. However, we ended up with a total amount of 76 professionals: most of them had practical experiences with managing software teams and, perhaps more importantly, most of them dealt with community smells in the past, thus being able to provide us with valid insights to address our research questions properly. Furthermore, we were able to collect answers from both women and men: this was an important requirement since women and men may have had different ways of solving socio-technical problems arising in software communities [5, 64]. The two aspects discussed above, *i.e.*, management experience and gender diversity, make us confident of the generalizability of our findings. Nonetheless, we are aware that more responses may have led to slightly different results or even highlighted additional refactoring strategies. For this reason, we aim at replicating our study taking into account a larger set of experienced practitioners.

6 RELATED WORK

The presence of community smells reflects both the health of the organization as well as the quality of the software produced (and also its life cycle). [42]. So, in the context of our work, we had to deal with both software engineering and organizational research. In this section, we outline related work in (i) establishing, measuring, tracking or otherwise improving the health or status of software engineering communities and (ii) empirically assessing the effects of community smells on social and technical aspects of source code.

Software communities health. On the software engineering side of the topic spectrum, several works provided fundamental insights into the problem, including the widely known socio-technical congruence [7] research, but without ever offering a theoretically- and empirically-established quality model. For instance,research community concentrated on establishing the link between several organizational structure qualities (*e.g.*, hidden-subcontractors in the organizational structure [1, 3], awareness [4, 39], distance and coordination [23, 26], etc.) with respect to software quality [57].

Jansen [31] proposed a framework for open-source ecosystems health, based on the study of the literature; in particular, the proposed framework was focused on parameters for ecosystem health without considers organizational structures or anti-patterns emerging thereto. Similarly, the work of Crowston and Howison [13] offered anecdotal evidence of the need for empirically-proven quality models for open-source communities. They argued that informal open-source communities are healthier since they are more engaged. Our work could be seen as a second step of their proposals, since we propose an empirically-grounded catalog of strategies that practitioners can be use successfully to mitigate their encounters with specific community smells.

At the other end of the spectrum, organization and social-network research proposed a plethora of organizational anti-patterns [43, 51], as well as (a few) best practices to address them [29, 60], with even fewer exceptions for open-source software communities [55]. For example, Giatsidis *et al.* [20] elaborated on collaboration structures with high-edge social network analysis. They concluded that organizationally-specific k-structured networks are more efficient than others, so there exists an organizational structure which best fits a pre-specified purpose. Similarly, the same authors investigated on the impact of communication, collaboration and cooperation over community structure qualities [19]. Insights from both papers would offer a valuable basis for argument over organizational structure research in software engineering. However, in our work, we face the problem asking practitioners to share us their knowledge and experience about sub-optimal situation; thus might lead to achieving more practical insights.

Research on community smells. In the last years, community smells have begun to receive particular attention [41, 58]; one of the motivation resides in the development of the tool able to detect them called CodeFace by Joblin *et al.* [32]. Indeed, the aforementioned tool was first augmented with heuristics capable of detecting community smells [58] and then adopted to investigate the impact of community smells over code smells [41]. In the first place, Tamburri *et al.* [58] assessed the detection capabilities of the proposed augmented tool, named CodeFace4Smells by surveying practitioners, who confirmed that the results given by the tool are accurate and meaningful. Also, the authors investigated (i) the diffuseness of four community smells in open-source and (ii) their relation with known socio-technical factors: their results provided evidence that smells are highly diffused and can be foreseen by taking certain socio-technical indicators under control. At the same time, Palomba *et al.* [41] discovered that community smells represent top factors preventing from refactoring; moreover, they are key features when it comes to predicting the severity of specific code smells. Similar works have concentrated on establishing the impact of community smells on other dimensions of software engineering (*e.g.*, architecture debt [35] and organization structure types [57]).

On another note, Catolino *et al.* [10] analyzed that in certain cases the emergence of community smells may be potentially reduced by increasing gender diversity. In their extended work [8], however, they found how practitioners do not perceive gender diversity and presence of women in software teams as relevant factors to avoid community smells, while they believe that other aspects, like developer's experience or team size, may make a community more prone to be affected by smells.

The works presented in this section is complementary to those discussed above, as it does not focus on the emergence of community smells or their impact, but rather on how practitioners deal with them and, particularly, on the strategies employed in practice to get rid of community smells. Nevertheless, it is important to point out that Tamburri *et al.* [58] have developed a mining study in which they assessed the diffuseness of community smells in open-source projects; our analysis of the perceived relevance of community smells can nicely triangulate the findings of Tamburri *et al.* [58] and potentially show preliminary insights into the awareness of practitioners with respect to community smells.

7 CONCLUSION

Community smells are critical socio-technical situations that may lead to increase social debt and overall project's costs [37, 54]. While the research community has mainly focused on understanding their properties [8, 10, 57, 58] and detecting them [41, 58], a few studies have investigated how practitioners remove them and increase the health status of software communities.

In this paper, we have done the first step toward the definition of refactoring strategies for four well-known community smells such as Organizational Silo, Black Cloud, Lone Wolf, and Radio Silence. We have designed a survey study and recruited 76 experienced professionals, with most of them who observed community smells in their career, to address two research questions related to (i) the perceived relevance of community smells and (ii) the corrective actions performed. On the one hand, key results of our study confirm that community smells are relevant problems for practitioners; in particular Lone Wolf and Black Cloud are the smells that more frequently arise in industry. On the other hand, we devised and discussed a brand new taxonomy of refactoring strategies that can be used by practitioners as a field manual to deal with community smells in practice.

Our research agenda includes first to complement the findings with semi-structured interviews and longitudinal studies. Then, we want to replicate the study aimed at considering a larger amount of practitioners and confirming the results achieved so far.

ACKNOWLEDGMENTS

Fabio gratefully acknowledges the help and financial support of the Swiss National Science Foundation through the SNF Project No. PZ00P2_186090 (TED). Furthermore, Damian's work is supported by the European Commission grants no. 787061 (H2020), ANITA, no. 825040 (H2020), RADON, and no. 825480 (H2020), SODALITE.

REFERENCES

[1] Vito Albino and A Claudio Garavelli. 1998. A neural network application to subcontractor rating in construction firms. *International Journal of Project Management* 16, 1 (1998), 9–14.
[2] Mousumi Banerjee, Michelle Capozzoli, Laura McSweeney, and Debajyoti Sinha. 1999. Beyond kappa: A review of interrater agreement measures. *Canadian journal of statistics* 27, 1 (1999), 3–23.
[3] David P Baron and David Besanko. 1992. Information, control, and organizational structure. *Journal of Economics & Management Strategy* 1, 2 (1992), 237–275.
[4] James M Bloodgood and JL Morrow Jr. 2003. Strategic organizational change: exploring the roles of environmental structure, internal conscious awareness and knowledge. *Journal of Management Studies* 40, 7 (2003), 1761–1782.
[5] Sarah Lynne Bowman. 2010. *The functions of role-playing games: How participants create community, solve problems and explore identity.* McFarland.
[6] Ronald S. Burt. 1987. Social contagion and innovation: Cohesion versus structural equivalence. *The American Journal of Sociology* 92, 6 (1987), 1287–1335. https://doi.org/doi:10.1086/228667
[7] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. 2008. Socio-technical congruence: a framework for assessing the impact of technical and work dependencies on software development productivity. In *Empirical software engineering and measurement.* ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/1414004.1414008
[8] Gemma Catolino, Fabio Palomba, Damian Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Gender Diversity and Community Smells: Insights from the Trenches. *IEEE Software* (2019).
[9] Gemma Catolino, Fabio Palomba, Damian Andrew Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Gender Diversity and Community Smells: Insights from the Trenches. *IEEE Software* (2019), to appear.
[10] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Gender diversity and women in software teams: How do they affect community smells?. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Society.* IEEE Press, 11–20.
[11] Gemma Catolino, Fabio Palomba, Damian Andrew Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Refactoring of Community Smells: The Practitioner's Strategies - Online Appendix - https://doi.org/10.6084/m9.figshare.10075406.
[12] Juliet M Corbin and Anselm Strauss. 1990. Grounded theory research: Procedures, canons, and evaluative criteria. *Qualitative sociology* 13, 1 (1990), 3–21.
[13] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2 (2005).
[14] Janet Finch. 1987. The vignette technique in survey research. *Sociology* 21, 1 (1987), 105–114.
[15] Timothy S Flanigan, Emily McFarlane, and Sarah Cook. 2008. Conducting survey research among physicians and other medical professionals: a review of current literature. In *Proceedings of the Survey Research Methods Section, American Statistical Association*, Vol. 1. 4136–47.

[16] Martin Fowler. 2018. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.
[17] Christian Fuchs. 2017. Sustainability and community networks. *Telematics and Informatics* 34, 2 (2017), 628–639. http://dblp.uni-trier.de/db/journals/tele/tele34.html#Fuchs17
[18] Jonas Gamalielsson and Björn Lundell. 2014. Sustainability of Open Source software communities beyond a fork: How and why has the LibreOffice project evolved? *Journal of Systems and Software* 89 (2014), 128–145.
[19] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2011. Evaluating cooperation in communities with the k-core structure. In *2011 International conference on advances in social networks analysis and mining*. IEEE, 87–93.
[20] Christos Giatsidis, Dimitrios M Thilikos, and Michalis Vazirgiannis. 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. *Knowledge and information systems* 35, 2 (2013), 311–343.
[21] Daniel J. Greenhoe. 2016. Properties of distance spaces with power triangle inequalities. *PeerJ PrePrints* 4 (2016), e2055. http://dblp.uni-trier.de/db/journals/peerjpre/peerjpre4.html#Greenhoe16a
[22] Lucas Gren. 2019. On Gender, Ethnicity, and Culture in Empirical Software Engineering Research. *CoRR* abs/1904.09820 (2019). http://dblp.uni-trier.de/db/journals/corr/corr1904.html#abs-1904-09820
[23] Rebecca E Grinter, James D Herbsleb, and Dewayne E Perry. 1999. The geography of coordination: dealing with distance in R&D work. In *Proceedings of the international ACM SIGGROUP conference on Supporting group work*. ACM, 306–315.
[24] Anna Hannemann, Hans-Jorg Happel, Matthias Jarke, Ralf Klamma, Steffen Lohmann, Walid Maalej, and Volker Wulf. 2009. Social Aspects in Software Engineering.. In *Software Engineering (Workshops) (LNI)*, Vol. 150. GI, 239–242. http://dblp.uni-trier.de/db/conf/se/se2009w.html#HannemannHJKLMW09
[25] Sébastien Harispe, Sylvie Ranwez, Stefan Janaqi, and Jacky Montmain. 2015. *Semantic Similarity from Natural Language and Ontology Analysis*. Synthesis Lectures on Human Language Technologies, Vol. 8. Morgan & Claypool Publisers.
[26] James D Herbsleb and Rebecca E Grinter. 1999. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software* 16, 5 (1999), 63–70.
[27] B Herrmann, C Thoni, and S Gachter. 2008. Antisocial Punishment Across Societies. *Science* 319, 5868 (2008), 1362 – 1367.
[28] Gert Jan Hofstede, Catholijn M. Jonker, and Tim Verwaart. 2008. Modeling Power Distance in Trade.. In *MABS (Lecture Notes in Computer Science)*, Nuno David and Jaime Simao Sichman (Eds.), Vol. 5269. Springer, 1–16. http://dblp.uni-trier.de/db/conf/mabs/mabs2008.html#HofstedeJV08
[29] Kei Ito, Hironori Washizaki, and Yoshiaki Fukazawa. 2016. Handover anti-patterns. In *Proceedings of the 5th Asian Conference on Pattern Language of Programs (Asian PLoP 2016), Taipei, Taiwan*.
[30] Hannu Jaakkola. 2012. Culture Sensitive Aspects in Software Engineering.. In *Conceptual Modelling and Its Theoretical Foundations (Lecture Notes in Computer Science)*, Antje Dusterhoft, Meike Klettke, and Klaus-Dieter Schewe (Eds.), Vol. 7260. Springer, 291–315. http://dblp.uni-trier.de/db/conf/birthday/thalheim2012.html#Jaakkola12
[31] Slinger Jansen. 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. *Information and Software Technology* 56, 11 (2014), 1508–1519.
[32] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. 2015. From Developer Networks to Verified Communities: A Fine-grained Approach. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1 (ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 563–573. http://dl.acm.org/citation.cfm?id=2818754.2818824
[33] Harry F. Joiner. 1992. Management Barriers to Software Reuse.. In *TRI-Ada*, Charles B. Engle Jr. (Ed.). ACM, 295–298. http://dblp.uni-trier.de/db/conf/sigada/triada92.html#Joiner92
[34] Sergi Lozano, Javier Borge, Alex Arenas, and Jose Luis Molina. 2008. Beyond Nadel's Paradox. A computational approach to structural and cultural dimensions of social cohesion. http://arxiv.org/abs/0807.2880
[35] Antonio Martini and Jan Bosch. 2017. Revealing Social Debt with the CAFFEA Framework: An Antidote to Architectural Debt.. In *ICSA Workshops*. IEEE Computer Society, 179–181. http://dblp.uni-trier.de/db/conf/icsa/icsaw2017.html#MartiniB17
[36] Mary L McHugh. 2013. The chi-square test of independence. *Biochemia medica: Biochemia medica* 23, 2 (2013), 143–149.
[37] Nils Brede Moe and Darja Šmite. 2008. Understanding a lack of trust in Global Software Teams: a multiple-case study. *Software Process: Improvement and Practice* 13, 3 (2008), 217–231.
[38] John Noll, Sarah Beecham, and Ita Richardson. 2010. Global software development and collaboration: barriers and solutions. *ACM Inroads* 1 (August 2010), 66–78. Issue 3. https://doi.org/10.1145/1835428.1835445
[39] AHJ Oomes. 2004. Organization awareness in crisis management. In *Proceedings of the international workshop on information systems on crisis response and management (ISCRAM)*.
[40] Premalatha Packirisamy. 2017. Managing Power Distance to Retain Talent: Evidence from India. *IJHCITP* 8, 3 (2017), 49–67. http://dblp.uni-trier.de/db/journals/ijhcitp/ijhcitp8.html#Packirisamy17
[41] Fabio Palomba, Damian Andrew Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. 2018. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Transactions on Software Engineering* (2018).
[42] Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. 2016. Smells like teen spirit: Improving bug prediction performance using the intensity of code smells. In *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 244–255.
[43] Anne Persson and Janis Stirna. 2006. How to transfer a knowledge management approach to an organization–a set of patterns and anti-patterns. In *International Conference on Practical Aspects of Knowledge Management*. Springer, 243–252.
[44] Paul Ralph, Mike Chiasson, and Helen Kelley. 2016. Social theory for software engineering research.. In *EASE*, Sarah Beecham, Barbara A. Kitchenham, and Stephen G. MacDonell (Eds.). ACM, 44:1–44:11.
[45] Oliver C Robinson. 2014. Sampling in interview-based qualitative research: A theoretical and practical guide. *Qualitative research in psychology* 11, 1 (2014), 25–41.
[46] Gregorio Robles and Jesus M Gonzalez-Barahona. 2006. Contributor turnover in libre software projects. In *IFIP International Conference on Open Source Systems*. Springer, 273–286.
[47] Victor Sanchez-Anguix, Tinglong Dai, Zhaleh Semnani-Azad, Katia P. Sycara, and Vicente J. Botti. 2012. Modeling Power Distance and Individualism/Collectivism in Negotiation Team Dynamics.. In *HICSS*. IEEE Computer Society, 628–637. http://dblp.uni-trier.de/db/conf/hicss/hicss2012.html#Sanchez-AnguixDSSB12
[48] Helen Sharp, Hugh Robinson, and Mark Woodman. 2000. Software Engineering: Community and Culture. *IEEE Software* 17, 1 (2000), 40–47. http://dblp.uni-trier.de/db/journals/software/software17.html#SharpRW00
[49] Karl Sigmund, Christoph Hauert, Arne Traulsen, and Hannelore De Silva. 2011. Social Control and the Social Contract: The Emergence of Sanctioning Systems for Collective Action. *Dynamic Games and Applications* 1, 1 (2011), 149–171. http://dblp.uni-trier.de/db/journals/dga/dga1.html#SigmundHTS11
[50] R Sommerfeld, H Krambeck, D Semmann, and M Milinski. 2007. Gossip as an alternative for direct observation in games of indirect reciprocity. *P Natl Acad Sci USA* 104, 44 (2007), 17435–17440.
[51] Janis Stirna and Anne Persson. 2009. Anti-patterns as a means of focusing on critical quality aspects in enterprise modeling. In *Enterprise, Business-Process and Information Systems Modeling*. Springer, 407–418.
[52] D. A. Tamburri. 2019. Software Architecture Social Debt: Managing the Incommunicability Factor. *IEEE Transactions on Computational Social Systems* 6, 1 (Feb 2019), 20–37. https://doi.org/10.1109/TCSS.2018.2886433
[53] Damian A. Tamburri, Rick Kazman, and Hamed Fahimi. 2016. The Architect's Role in Community Shepherding. *IEEE Software* 33, 6 (2016), 70–79.
[54] Damian A Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2013. What is social debt in software engineering?. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on*. 93–96. https://doi.org/10.1109/CHASE.2013.6614739
[55] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2015. Social debt in software engineering: insights from industry. *J. Internet Services and Applications* 6, 1 (2015), 10:1–10:17. http://dblp.uni-trier.de/db/journals/jisa/jisa6.html#TamburriKLV15
[56] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. 2015. Social debt in software engineering: insights from industry. *J. Internet Services and Applications* 6, 1 (2015), 10:1–10:17.
[57] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2019. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering* 24, 3 (2019), 1369–1417.
[58] Damian Andrew Andrew Tamburri, Fabio Palomba, and Rick Kazman. 2019. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* (2019).
[59] Jordi Torrents and Fabrizio Ferraro. 2015. Structural Cohesion: Visualization and Heuristics for Fast Computation. *Journal of Social Structure* 16 (2015), 8. http://dblp.uni-trier.de/db/journals/joss/joss16.html#TorrentsF15
[60] Ariel Tseitlin. 2013. The Antifragile Organization. *Commun. ACM* 56, 8 (2013), 40–44.
[61] Daniel Villatoro, Sandip Sen, and Jordi Sabater-Mir. 2010. Of Social Norms and Sanctioning: A Game Theoretical Overview. *IJATS* 2, 1 (2010), 1–15. http://dblp.uni-trier.de/db/journals/ijats/ijats2.html#VillatoroSS10
[62] Janiece L Walker. 2012. Research column. The Use of Saturation in Qualitative Research. *Canadian Journal of Cardiovascular Nursing* 22, 2 (2012).
[63] Stanley Wasserman and Katherine Faust. 1994. *Social Network Analysis. Methods and Applications*. Cambridge University Press.
[64] Lynn Zimmer. 1989. Solving women's employment problems in corrections: Shifting the burden to administrators. *Women & Criminal Justice* 1, 1 (1989), 55–79.