Splicing Community Patterns and Smells: A Preliminary Study

Manuel De Stefano
m.destefano36@studenti.unisa.it
SeSa Lab
University of Salerno, Italy

Fabiano Pecorelli
fpecorelli@unisa.it
SeSa Lab
University of Salerno, Italy

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

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

Andrea De Lucia
adelucia@unisa.it
SeSa Lab
University of Salerno, Italy

ABSTRACT

Software engineering projects are now more than ever a community effort. In the recent past, researchers have shown that their success may not only depend on source code quality, but also on other aspects like the balance of distance, culture, global engineering practices, and more. In such a scenario, understanding the characteristics of the community around a project and foresee possible problems may be the key to develop successful systems. In this paper, we focus on this research problem and propose an exploratory study on the relation between community patterns, *i.e.*, recurrent mixes of organizational or social structure types, and smells, *i.e.*, sub-optimal patterns across the organizational structure of a software development community that may be precursors of some sort of social debt. We exploit association rule mining to discover frequent relations between them. Our findings show that different organizational patterns are connected to different forms of socio-technical problems, possibly suggesting that practitioners should put in place specific preventive actions aimed at avoiding the emergence of community smells depending on the organization of the project.

CCS CONCEPTS

• **Software and its engineering → Programming teams**;

KEYWORDS

Community patterns; Community smells; Empirical studies.

ACM Reference Format:
Manuel De Stefano, Fabiano Pecorelli, Damian A. Tamburri, Fabio Palomba, and Andrea De Lucia. 2018. Splicing Community Patterns and Smells: A Preliminary Study. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/1122445.1122456

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Software is increasingly being developed by globally-distributed communities having complex social networks of software development [42]. Over the last years, researchers have been investigating the impact of such complex social networks on the sustainability of open- and closed-source communities as well as source code quality, finding them to be a highly relevant factor for the success of software systems [7, 27, 32, 37, 44]. As an example, Kwan *et al.* [27] showed that the alignment between social and technical structure of the community, *i.e.*, the so-called socio-technical congruence [8], has an effect on the build success, while Palomba *et al.* [32] found that community-related factors can increase the criticality of source code quality issues. As such, studying software communities does not only represent a way to understand and learn how to reduce social debt, *i.e.*, the unforeseen cost given by a wrong management of the communication/coordination between developers [38], but also to possibly improve the overall quality of the technical products being developed [27, 32].

In recent work, Tamburri *et al.* [42] elicited a set of *community patterns*, namely common governance mechanisms adopted by open-source practitioners to manage the community, showing that each pattern has its own characteristics and peculiarities [10]. On the other hand, Tamburri *et al.* [37, 44] also explored the dark-side of software communities and described a set of sub-optimal organizational structures that lead to the emergence of both social and technical debt, which have been named as *community smells*.

While researchers are studying community patterns and smells in isolation, for instance by assessing how community smells manifest themselves and can be mitigated [11, 12], there is still a lack of knowledge on the joint relationship between community patterns and smells. An improved understanding of such a relation is important to reveal whether and to what extent certain governance mechanisms are more incline to incite the emergence of community smells. Such understanding would allow researchers to further investigate the problem, possibly proposing monitoring and/or remediation strategies.

In this paper, we start tackling this lack of knowledge and propose an exploratory empirical study on how community patterns relate to community smells. By mining data from 25 open-source communities, we first exploit association rule learning [1] with the aim of discovering frequent co-occurrences between the two phenomena of interest, and then reason on the rationale behind the observed relations. Key findings of our study show that different

community patterns relate to different smells, highlighting that the governance mechanisms which are put in place may potentially have consequences in terms of social debt.

Our results have implications for both researchers and practitioners. Based on our findings, the former can further analyze the dynamics behind community patterns and how various known and established governance mechanisms lead to socio-technical issues or whether novel mechanisms are required for specific operational conditions. The latter, instead, can exploit our results to understand what are the risks associated with the community pattern(s) currently in place in their projects and take preventive actions for business continuity perhaps focusing on working out the aforementioned novel governance patterns and practices.

Structure of the paper. Section 2 overviews the design of the exploratory study, while Section 3 presents the achieved results and discusses them. The potential limitations of the study are reported in Section 4. Section 5 discusses the literature related to community patterns and smells. Finally, Section 6 concludes the paper and presents our future research agenda.

2 EMPIRICAL STUDY DESIGN

The *goal* of our empirical study is to investigate the relation between community patterns and smells, with the *purpose* of understanding whether the structural organization of a community may potentially lead to some sort of social debt. The *perspective* is of both researchers and practitioners, who are interested in discovering the potential impact of organizational patterns on the emergence of community smells.

More specifically, the empirical study is driven by the following research question (**RQ**):

> **RQ.** *What is the relation between community patterns and community smells?*

In the following sections, we describe the context of the study as well as the data collection and analysis required to address our **RQ**.

2.1 Context of the Study

The *context* of our empirical study was composed of (1) projects, (2) community patterns, and (3) community smells.

Projects. We considered 25 open source software communities coming from the GitHub repository, sampled according to guidelines from the state of the art [15] and refined applying best-practice sampling criteria [26]. In particular, from the initial list of 81,327,803 open-source projects available in the repository, we first excluded systems having less that 10 contributors: such a filter was required to gather projects built by an actual community of developers. Then, we excluded systems with less that 100 commits, so that we could rely on a decent amount of information to study how developers collaborate with each other—this is required to accurately detect community smells, as explained later in the paper. Finally, we took into account systems having at least 50 KLOCs with the aim of studying medium to large projects. Applying these filters, we came up with a total of 44,387,266 projects. For computational constraints, we randomly selected 25 of them.

Table 1: Basic characteristics of the software project communities considered in our study—domain taxonomy tailored from literature [6].

Name	# Rel.	# Commits	# Contributors	#KLOC	Domain
Netty	164	8,123	258	438	Software Tools
Android	3	132	14	382	Library
Arduino	74	6,516	210	192	Electronics prototyping platform
Bootstrap	55	2,067	389	378	Web libraries and frameworks
Boto	86	7,111	495	56	Web libraries and frameworks
Bundler	251	8,464	549	112	Web libraries and frameworks
Cloud9	97	9,485	64	293	Application software
Composer	35	7,363	629	254	Software Tools
Cucumber	8	566	15	382	Software Tools
Ember-JS	129	5,151	407	272	Web libraries and frameworks
Gollum	76	1,921	143	182	Non-web libraries and frameworks
Hammer	25	1,193	84	199	Web libraries and frameworks
BoilerPlate	12	469	48	266	Web libraries and frameworks
Heroku	52	353	10	292	Software Tools
Modernizr	27	2,392	220	382	Web libraries and frameworks
Mongoid	253	6,223	317	187	Non-web libraries and frameworks
Monodroid	2	1,462	61	391	Non-web libraries and frameworks
PDF-JS	43	9,663	228	398	Web libraries and frameworks
Scrapy	78	6,315	242	287	Non-web libraries and frameworks
Refinery	162	9,886	385	188	Software Tools
Salt	146	81,143	1,781	278	Software Tools
Scikit-Learn	2	4,456	17	344	Non-web libraries and frameworks
SimpleCV	5	2,625	69	389	Non-web libraries and frameworks
Hawkthorne	116	5,537	62	211	Software Tools
SocketRocket	10	494	67	198	Non-web libraries and frameworks

Table 1 summarizes the characteristics of the extracted software projects in terms of (i) size, measured as number of public releases issued and number of commits performed over their history, (ii) contributors, and (iii) application domain, according to the taxonomy proposed by Borges *et al.* [6].

Community Patterns. Table 2 overviews the community patterns under investigation, along with a short description. They come from existing literature and include various forms of organizational structures. The choice of focusing on these patterns come from the availability of an automated tool enabling their detection, *i.e.*, Yoshi [42]. In particular, this tool implements a two-step approach: given a Github repository, it mines commit history, issue tracker, and contributor's data in order to compute metrics that characterize the structure of the community. For example, Yoshi computes the overall engagement of developers, *i.e.*, the amount of time that the contributors actively spend in community-related actions, or the level of formality of the decisional process exercised or self-imposed on the community. In the second step, the tool implements a decision tree that, on the basis of the measurements previously computed, is able to classify the organizational pattern implemented by a community, *e.g.*, a formal or informal group.

It is important to highlight that the performance of Yoshi has been empirically assessed [42], showing an accuracy close to 100%. As such, the tool represents the ideal way to detect community patterns in our study.

Community Smells. Regarding community smells, we focused on four specific types such as:

Black Cloud. This smell arises when the community presents an information overload due to lack of structured communications or cooperation governance;

Bottleneck. In this case, one member interposes herself into every formal interaction across two or more sub-communities with little or no flexibility to introduce other parallel channels.

Table 2: Organizational structure types considered in our empirical study.

Name	Description
Communities of practice (CoP)	A CoP consists of collocated groups of people who share a concern, a set of problems, or a passion about a practice. Interactions are frequent, face-to-face, collaborative (to help each other) and constructive (to increase mutual knowledge). This set of social processes and conditions is called situatedness [17]. An example is the SRII community[1] which gathers multiple CoPs (corporate and academic) into a single one, meeting physically to informally exchange best practices in services science.
Informal Networks (IN)	INs are loose networks of ties between individuals that happen to come informally in contact in the same context. Primary indicator is the high strength of informal member ties. Finally, IN do not use governance practices [13]. An example in academia, is the informal and loosely coupled set of research communities around a single topic (e.g., computer science) is a world-wide informal network.
Formal Networks (FN)	FNs rigorously select and prescribe memberships, which are created and acknowledged by FN management. Direction is carried out according to corporate strategy and its mission is to follow this strategy [40]. An example in software engineering is the OMG (Object Management Group): it is a formal network, since the interaction dynamics and status of the members (i.e. the organizations which are part of OMG) are formal; also, the meeting participants (i.e. the people that corporations send as representatives) are acknowledged formally by their corporate sponsors.
Informal Communities (IC)	ICs reflect sets of people part of highly-dispersed organisation, with a common interest, often closely dependent on their practice. They interact informally across unbound distances, frequently over a common history or culture (e.g. shared ideas, experience etc). The main difference they have with all communities (with the exception of NoPs) is that their localisation is necessarily dispersed (e.g., contrarily to INs where networked interactions can also be in the same timezone or physical location) so that the community can reach a wider audience [40]. Loosely-affiliated political movements (such as green-peace) are examples of ICs: their members disseminate their vision (based on a common idea, which is the goal of the IC).
Networks of Practice (NoP)	A NoP is a networked system of communication and collaboration that connects CoPs (which are localised). In principle anyone can join it without selection of candidates (e.g. Open-Source forges are an instance of NoP). NoPs have the highest geodispersion. An unspoken requirement is expected IT literacy [35]. For example, previous literature [4] discusses Socio-technical Networks in software engineering using the exact terms with which NoPs are defined in literature.
Workgroups (WG)	WG are made of technical experts whose goals span a specific business area. WGs are always accompanied by a number of organisational sponsors and are expected to generate tangible assets and benefits (i.e., Return-On-Investment). Fundamental attributes of WGs are collocation and the highest cohesion of their members (e.g., long-time collaborators). For example, in software engineering, the IFIP WG 2.10 on software architecture[2] is obviously a WG, since its effort is planned and steady, with highly cohesive action of its members, as well as focused on pursuing the benefits of certain organisational sponsors (e.g. UNESCO for IFIP).
Project-Teams (PT)	PTs are fixed-term, problem-specific aggregations of people with complementary skills who work together to achieve a common purpose for which they are accountable. They are enforced by their organisation and follow specific strategies or organisational guidelines (e.g. time-to-market, effectiveness, low-cost, etc.). Their final goal is delivery of a product or service [40].
Formal Groups (FG)	FGs are comprised of people which are explicitly grouped by corporations to act on (or by means of) them (e.g. governing employees or ease their job or practice by grouping them in areas of interest). Each group has a single organisational goal, called mission (governing boards are groups of executives whose mission is to devise and apply governance practices successfully). In comparison to Formal Networks, they seldom rely on networking technologies, on the contrary, they are local in nature and are less formal since there are no explicit governance protocols employed other than the grouping mechanism and the common goal. Examples of formal groups in software engineering are software taskforces, e.g. IEEE Open-Source Software Task Force[3].
Social Networks (SN)	SNs represent the emergent network of social ties spontaneously arising between individuals who share, either willingly or not, a practice or common interest. Conversely, an unstructured network is (often by-design) not constrained by any design or structural tie (e.g., a common social practice) [47]. SNs act as a gateway to communicating communities [13].

Organizational Silo. This smell appears when there are siloed areas of the developer community that do not communicate, except through one or two of their respective members;

Lone Wolf. Instances of this smell arise when the developer community has unsanctioned or defiant contributors who carry out their work with little consideration of their peers, their decisions and/or communication.

We focused on these community smells for multiple reasons. In the first place, these specific smells have been shown by previous research [32, 39] to be (1) among the most problematic community-related issues to deal with and (2) a potential threat to the emergence of technical debt. Secondly, these smells can be detected exploiting an automated tool named CodeFace4Smells [44]. This is a fork of CodeFace [24], a tool originally designed to extract coordination and communication graphs mapping the developer's relations within a community. CodeFace4Smells augments these graphs with detection rules able to identify the four community smells taken into account. As an example, the identification pattern for Lone Wolf is based on the detection of development collaborations between two community members that have intermittent communication counterparts or feature communication by means of an external "intruder", *i.e.*, not involved in the collaboration.

Also for this tool, it is important to comment on its accuracy. CodeFace4Smells has been empirically evaluated [39, 44] by means of surveys and/or semi-structured interviews with both the original industrial and open-source practitioners belonging to 60 communities. In particular, the authors of the tool showed practitioners the results obtained when running CodeFace4Smells on their communities, asking for confirmation. As an outcome, they all reported the validity and usefulness of the tool, without pointing out additional problematic situations occurred in their communities. In other words, according to developers, the community smells output by the tool *are all true positives*; as for false negatives, if they exist, these *were not pointed out by original developers*. The results achieved by the tool in previous studies [39, 44] make us confident of the high reliability of the tool and its suitability for our study.

Data Collection and Analysis. To address our **RQ** and evaluate the relationship between community patterns and smells we performed a three step data collection and analysis:

(1) Identification of community patterns;
(2) Identification of community smells;
(3) Association rule discovery;

To achieve the first step, we exploited Yoshi to discover the community patterns that affect the considered software projects. This was conducted as a release-level analysis. The releases were directly extracted from the Github repository of each considered project. We performed such an evolutionary analysis because in this way we could have a much richer and meaningful dataset if compared to the only 25 data points represented by the most recent snapshots of the communities considered.

Then, we detected the community smells that affect the considered software projects, exploiting the aforementioned CodeFace4Smell. Once we have extracted the list of community smells

contained in each of the releases of the projects considered, we mined association rules [1] for detecting which community patterns and smells co-occur. In particular, association rule discovery is an unsupervised learning technique used for local pattern detection highlighting attribute value conditions that occur together in a given dataset [1]—in our case, the dataset contained the set of community patterns and smells discovered in each release of the considered projects. An association rule $R_{left} \rightarrow R_{right}$ implies that, if a certain community pattern occurs in a project, then a community smell should occur as well. The strength of an association rule is determined by two metrics, *i.e.*, support and confidence [1]:

$$support = \frac{|R_{left} \cup R_{right}|}{T} \quad (1)$$

$$confidence = \frac{|R_{left} \cup R_{right}|}{R_{left}} \quad (2)$$

where T is the total number of co-occurrences between community patterns and smells in our dataset. To implement association rules, we exploited the well-known aPriori algorithm [1], which is available in the `R` toolkit.[4] In Section 3 we report and discuss the association rules having a support higher than 0.6 and confidence higher than 0.8 [1]. This focus is necessary to produce and report only the association rules having the highest strength.

Furthermore, we computed the lift metric, which measures the ability of a rule to correctly identify a relationship with respect to a random choice model [1]. A lift value higher than 1 indicates that the left-hand and right-hand operators of an association rule appear together more often than expected, thus meaning that the occurrence of the left-hand operator often implies the co-presence of the right-hand operator. To understand the statistically significance of the rules found, we employed Fisher's exact test [16] on the lift value achieved by the mined association rules: specifically, the test measures the significance of the deviation between the association rule model and the random choice models compared when computing the lift. The statistical significance is obtained in case of ρ-value lower than 0.05.

3 RESULTS AND DISCUSSION

Table 3 reports the association rules, grouped by community pattern, extracted after the application of the aPriori algorithm [1]. In this section, we also provide some qualitative analysis of the association rules aimed at further investigating the results and possibly understanding whether the relation between community patterns and smells is causal: to this aim, however, we deeper analyzed only a subset of the systems contained in our dataset. Specifically, we focused on the two largest projects, namely Arduino and Salt.

A first consideration is related to the fact that, when not filtering association rules by support and confidence, we found relationships between smells and all the communities identified by Yoshi, with the exception of Formal Networks. Likely, this is due to the rigorousness used to select members in this community type: indeed, only certified and acknowledged developers can become members of these types of communities, which typically operate under strict regulatory contribution policies and codes of conduct [45]. As a consequence, developers within the community need to follow strict code of conducts to continue contributing and this is known to address a number of known organisational issues but also manifesting unexpected ones such as higher turnover [45]. In our case, think of the release 1.6.0 of Arduino, where Yoshi identifies a formal network; in this context, the developers adopted a code of conduct[5] to avoid unfriendly behavior among members. This result seems to confirm previous findings indicating that the usage of code of conducts actually supports the activity of software communities by creating a friendly and inclusive environment [45].

On the other hand, other community types are quite prone to be smelly. In our dataset, Formal Groups are strictly connected with two types of community smells, i.e., Bottleneck and Lone Wolf. To discover the reasons behind the relationship, we manually analyzed the sources of communications the systems rely on, i.e., the GitHub issue trackers and the mailing lists. Basically, formal groups are formed by people which are explicitly grouped to reach single specific missions. The problems arise in case two groups of the network need to communicate: in these cases it is usual that such groups communicate by means of a representative member, thus naturally leading to the introduction of a Bottleneck smell instance, which appears when there is an overhead due to members interposing themselves into every formal interaction between two groups. An interesting example appeared in the Arduino community, where the communications related to programming questions[6] are often conducted by two specific members, i.e., `Member A`[7] (present in 2,675 forum posts over the total 3,410) and `Member B` (present in 2,155 forum posts over the total 3,410). At the same time, the peculiarities of the community type makes it more prone to be affected by the Lone Wolf smell, which represents an extreme case of formal group, i.e., when a small set of developers perform their tasks without caring the decisions made within the group.

Besides having a high support and confidence values, the rules found also have a lift higher than 1, confirming that the two community smells often appear within formal groups. Note that the high lift values are also statistically significant as the Fisher's exact test quantified the $p - values$ as lower than 0.05.

The relationships between the Informal Communities and the Organizational Silo Effect and Lone Wolf smells were also quite expected. In this case, an informal community refers to highly-dispersed organizations having a common goal. The high dispersion of developers makes the community intrinsically more prone to be affected by the Organizational Silo Effect, since it may happen that some of the developers do not communicate with others causing poor social connections among the community members.

The high dispersion characteristic also explains the relationship with the Lone Wolf smell: as previously explained, this smell appears when a single developer or a small group of community members start working in isolation without considering the decisions made within the community. Of course, a dispersed environment without a well defined structure is more prone to such behavior because developers cannot physically meet each other daily. The results

[4] https://www.rdocumentation.org/packages/arules/versions/1.6-4/topics/apriori.

[5] http://forum.arduino.cc/index.php?topic=148996.0

[6] http://forum.arduino.cc/index.php?board=4.0

[7] Names of developers are anonymised to preserve their privacy

Table 3: Association rules between community patterns and smells.

Rule	Support	Confidence	Lift	p-value
Formal Group → Bottleneck	0.77	0.91	1.54	0.033
Formal Group → Lone Wolf	0.73	0.88	1.26	0.013
Informal Community → Organisational Silo Effect	0.74	0.94	1.69	0.011
Informal Community → Lone Wolf	0.68	0.82	1.63	0.022
Informal Network → Organisational Silo Effect	0.71	0.85	1.46	0.024
Informal Network → Black Cloud	0.69	0.83	1.55	0.043
Network of Practice → Bottleneck	0.76	0.89	1.59	0.032

were also confirmed when looking at the lift value which was higher than 1, being statistically significant (p-values lower than 0.05).

As for Informal Networks, it is worth remarking that this community type does not use governance practices. As such, it is naturally prone to the appearance of a Black-cloud instance, i.e., information overload caused by the lack of structured communication. At the same time, lack of governance also tends to make developers more independent from the community, possibly leading to the introduction of a Lone Wolf smell instance. Also in these cases, the lift values were higher than one while the p-values lower than 0.05: for this reason, we can conclude that the relationships discovered are meaningful and statistically significant.

Finally, we found an unexpected connection between Networks of Practice and the Bottleneck smell. By definition, a network of practice is a community that connects communities of practice, i.e., collocated groups in which interactions are frequent and collaborative. While such communities should theoretically be effective in communication, they are often affected by a Bottleneck due to developers who act as middleman between two groups. For example, in the version $v2015.5.0$ of the Salt project a single developer managed most of the communications performed by developers, becoming in practice a bottleneck. Interestingly, the lift value reached 1.59 with p-value = 0.032, confirming that the relationship is strong and statistically significant.

To broaden the scope of the discussion, the results achieved show that different community patterns are more prone to be affected by different community smells: this practically means that the information extracted by Yoshi about the community pattern can be exploited by practitioners as a useful source to diagnose and understand underlying social, socio-technical as well as organizational issues across their community.

> ***Finding 1***. *Different community patterns relate to different community smells. The reasons behind the presence of smells are strongly related to the characteristics and peculiarities of the community patterns.*

4 THREATS TO VALIDITY

In this section we discuss factors that might have influenced our study and be a threat to its validity, focusing on threats to *construct*, *conclusion* and *external* validity.

4.1 Threats to Construct Validity

Threats to construct validity are related to the relationships between theory and observation. Generally, this threat refers to imprecision in the measurements performed. In our case, this impacts all the gathered data, as it might be "biased" by the imprecision of the exploited tools. However, both Yoshi and CodeFace4Smells were previously validated [24, 37, 41, 42], showing good detection capabilities. This increases the reliability of our study and make us confident of the accuracy of the collected data.

4.2 Threats to Conclusion Validity

The main threat in this category is the use of the aPriori algorithm to discover the relationships between the observed phenomena. On the one hand, this technique has been widely adopted by researchers to study hidden relations between two phenomena (*e.g.*, [30, 31, 48]). On the other hand, we only considered and discussed the strongest rules, namely the most reliable ones which had high support and confidence. In addition, we also closely looked at two systems of the dataset with the aim of providing qualitative examples and a rationale that explains the rules discovered.

4.3 Threats to External Validity

The main issue concerned with the generalization of the results is the number of software communities analyzed in the study. While a set of 25 systems is not a statistically significant sample of the most active projects present in Github, it is important to remark that the main goal of our paper was to discover a relation between community patterns and smells, and not a large-scale study of open source projects properties. Furthermore, we were able to perform finer observations looking at some specific projects of our dataset, which were studied closely. For this reason, we believe that the dataset can be considered large enough for answering our research question. In addition, to make our findings as generalizable as possible we took into account a variety of communities having different characteristics, scope, size, and coming from different application domains. We plan to extend our investigation on a larger set of communities.

The choice of focusing on certain community patterns and smells might be a threat to the generalization of the results as well. While further replications of our work would be desirable and already part of our future research agenda, in our context we had to limit the analysis to those patterns and smells because of the tools available.

5 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) [33]. 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 [2, 3], awareness [5, 29], distance and coordination [20, 21], etc.) with respect to software quality [43].

Jansen [23] 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 [14] 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 [34, 36], as well as (a few) best practices to address them [22, 46], with even fewer exceptions for open-source software communities [39]. For example, Giatsidis *et al.* [19] 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 [18]. 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 situations; thus might lead to achieving more practical insights.

Research on community smells. In the last years, community smells have begun to receive particular attention [32, 44]; one of the motivations resides in the development of the tool able to detect them called CodeFace by Joblin *et al.* [25]. Indeed, the aforementioned tool was first augmented with heuristics capable of detecting community smells [44] and then adopted to investigate the impact of community smells over code smells [32]. In the first place, Tamburri *et al.* [44] 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.* [32] 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 [28] and organization structure types [43]).

On another note, Catolino *et al.* [12] analyzed that in certain cases the emergence of community smells may be potentially reduced by increasing gender diversity. In their extended work [9], 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.* [44] 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.* [44] and potentially show preliminary insights into the awareness of practitioners with respect to community smells.

6 CONCLUSION

In this paper, we studied the relation between community patterns and community smells in open-source software projects. We identified community patterns in a set of 25 open source projects, hosted on Github, exploiting Yoshi, while we detected community smells that affect the same set of projects using CodeFace4Smells. Finally, we exploit association rule mining, particularly the aPriori algorithm, to discover relations between them.

The key findings of the study show that specific communities smells may arise depending on the peculiarities of the community organization. Our results may be useful to prevent the occurrence of these smells, if the community is aware that it is following a certain pattern. Our future research agenda includes a replication of our experiments on a larger dataset. Moreover, we plan to further analyze the properties of community patterns and the possible impact that organizational decisions have on social debt.

REFERENCES

[1] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining Association Rules Between Sets of Items in Large Databases. *SIGMOD Rec.* 22, 2 (June 1993), 207–216. https://doi.org/10.1145/170036.170072

[2] 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.

[3] David P Baron and David Besanko. 1992. Information, control, and organizational structure. *Journal of Economics & Management Strategy* 1, 2 (1992), 237–275.

[4] Christian Bird, Nachiappan Nagappan, Harald Gall, Brendan Murphy, and Premkumar Devanbu. 2009. Putting It All Together: Using Socio-technical Networks to Predict Failures. In *Proceedings of the 2009 20th International Symposium on Software Reliability Engineering (ISSRE '09)*. IEEE Computer Society, Washington, DC, USA, 109–119. https://doi.org/10.1109/ISSRE.2009.17

[5] 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.

[6] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors that Impact the Popularity of GitHub Repositories.. In *IEEE International Conference on Software Maintenance and Evolution*. IEEE, –, 334–344.

[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* (Kaiserslautern, Germany). ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/1414004.1414008

[8] 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 *ESEM '08: Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement* (Kaiserslautern, Germany). ACM, New York, NY, USA, 2–11. https://doi.org/10.1145/1414004.1414008

[9] Gemma Catolino, Fabio Palomba, Damian Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Gender Diversity and Community Smells: Insights from the Trenches. *IEEE Software* (2019).

[10] Gemma Catolino, Fabio Palomba, and Damian A Tamburri. [n.d.]. The Secret Life of Software Communities: What we know and What we Don't know. ([n. d.]).

[11] Gemma Catolino, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Filomena Ferrucci. 2019. Gender diversity and community smells: insights from the trenches. *IEEE Software* 37, 1 (2019), 10–16.

[12] 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.

[13] Rob Cross, Jeanne Liedtka, and Leigh Weiss. 2005. A Practical Guide to Social Networks. *Harvard Business Review* (2005), –.

[14] Kevin Crowston and James Howison. 2005. The social structure of free and open source software development. *First Monday* 10, 2 (2005).

[15] Davide Falessi, Wyatt Smith, and Alexander Serebrenik. 2017. STRESS: A Semi-Automated, Fully Replicable Approach for Project Selection.. In *ESEM*. IEEE, 151–156. http://dblp.uni-trier.de/db/conf/esem/esem2017.html#FalessiSS17

[16] Ronald Aylmer Fisher. 1922. On the Interpretation of chi^2 from Contingency Tables, and the Calculation of P. *Journal of the Royal Statistical Society* 85, 1 (Jan. 1922), 87–94. https://doi.org/10.2307/2340521

[17] Shaun Gallagher. 2006. Introduction: The Arts and Sciences of the Situated Body. *Janus Head* 9, 2 (2006), 1–2.

[18] 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.

[19] 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.

[20] 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.

[21] James D Herbsleb and Rebecca E Grinter. 1999. Architectures, coordination, and distance: Conway's law and beyond. *IEEE software* 16, 5 (1999), 63–70.

[22] 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*.

[23] 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.

[24] 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 (ICSE 2015)*. ACM Press, Piscataway (NY), US, 563–573. https://doi.org/10.1109/ICSE.2015.73

[25] 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* (Florence, Italy) *(ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 563–573. http://dl.acm.org/citation.cfm?id=2818754.2818824

[26] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. Germán, and Daniela E. Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), 2035–2071. http://dblp.uni-trier.de/db/journals/ese/ese21.html#KalliamvakouGBS16

[27] Irwin Kwan, Adrian Schroter, and Daniela Damian. 2011. Does Socio-Technical Congruence Have an Effect on Software Build Success? A Study of Coordination in a Software Project. *IEEE Trans. Softw. Eng.* 37, 3 (May 2011), 307–324. https://doi.org/10.1109/TSE.2011.29

[28] 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

[29] AHJ Oomes. 2004. Organization awareness in crisis management. In *Proceedings of the international workshop on information systems on crisis response and management (ISCRAM)*.

[30] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Fausto Fasano, Rocco Oliveto, and Andrea De Lucia. 2018. A large-scale empirical study on the lifecycle of code smell co-occurrences. *Information and Software Technology* 99 (2018), 1–10.

[31] F. Palomba, G. Bavota, M. D. Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia. 2015. Mining Version Histories for Detecting Code Smells. *IEEE Transactions on Software Engineering* 41, 5 (May 2015), 462–489. https://doi.org/10.1109/TSE.2014.2372760

[32] 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).

[33] 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.

[34] 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.

[35] K. Ruikar, L. Koskela, and M. Sexton. 2009. Communities of practice in construction case study organisations: Questions and insights. *Construction Innovation* 9, 4 (2009), 434–. http://proquest.umi.com/pqdweb?did=1920022811&Fmt=7&clientId=4574&RQT=309&VName=PQD

[36] 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.

[37] Damian A. Tamburri, Rick Kazman, and Hamed Fahimi. 2016. The Architect's Role in Community Shepherding. *IEEE Software* 33, 6 (2016), 70–79.

[38] 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

[39] 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

[40] Damian Andrew Tamburri, Patricia Lago, and Hans van Vliet. 2013. Organizational social structures for software engineering. *ACM Comput. Surv.* 46, 1 (2013), 3.

[41] D. A. Tamburri and E. D. Nitto. 2015. When Software Architecture Leads to Social Debt. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*. ACM Press, Piscataway (NY), US, 61–64. https://doi.org/10.1109/WICSA.2015.16

[42] Damian Andrew Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. 2018. Discovering community patterns in open-source: a systematic approach and its evaluation. *Empirical Software Engineering* 24 (2018), 1369–1417.

[43] 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.

[44] 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).

[45] P. Tourani, B. Adams, and A. Serebrenik. 2017. Code of conduct in open source projects. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. ACM, Piscataway (NY), US., 24–33. https://doi.org/10.1109/SANER.2017.7884606

[46] Ariel Tseitlin. 2013. The Antifragile Organization. *Commun. ACM* 56, 8 (2013), 40–44.

[47] Jan Zich, Yoshiharu Kohayakawa, Vojtech Rödl, and V. Sunderam. 2008. JumpNet: Improving Connectivity and Robustness in Unstructured P2P Networks by Randomness. *Internet Mathematics* 5, 3 (2008), 227–250. http://dblp.uni-trier.de/db/journals/im/im5.html#ZichKRS08

[48] Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. 2005. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering* 31, 6 (2005), 429–445.