Impacts of Software Community Patterns on Process and Product: An Empirical Study

Manuel De Stefano,[1] Emanuele Iannone,[1] Fabiano Pecorelli,[1] Damian Andrew Tamburri[2]

[1]*SeSa Lab, University of Salerno (It) — [2]JADE Lab, Eindhoven University of Technology - Jheronimus Academy of Data Science (NL)*
madestefano@unisa.it, eiannone@unisa.it, fpecorelli@unisa.it, d.a.tamburri@tue.nl

Abstract

Software engineering projects are now more than ever a community effort. In the recent past, researchers have shown that their success not only depends on source code quality, but also on other aspects like the balance of power distance, culture, and 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 aspects related to the quality of software products and processes by mining open-source software repositories hosted on GitHub. We first exploit association rule mining to discover frequent relations between community pattern and community smells, *i.e.,* sub-optimal patterns across the organizational structure of a software development community that may be precursors of some form of social debt. Further on, we use statistical analyses to understand their impact on software maintainability and on the community engagement, in terms of contributions and issues. Our findings show that different organizational patterns are connected to different forms of socio-technical problems; further on, specific combinations are set in equally specific contextual conditions. Findings support two possible conclusions: (1) practitioners should put in place specific preventive actions aimed at avoiding the emergence of community smells and (2) such actions should be drawn according to the contextual conditions of the organization and the project.

Keywords: Community patterns, Community smells, Empirical studies.

1. Introduction

Software is increasingly being developed by globally-distributed communities having complex social networks of software development [76]. Over the last few 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 [69, 10, 46, 61, 78]. As an example, Kwan *et al.* [46] showed that the alignment between social and technical structure of the community, *i.e.,* the so-called socio-technical congruence [11], has an effect on the build success, while Palomba *et al.* [61] 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 [71], but also to possibly improve the overall quality of the technical products being developed [46, 61].

In recent work, Tamburri *et al.* [76] elicited a set of *community patterns*—common governance mechanisms adopted by open-source practitioners to manage the community—showing that each pattern has its own characteristics and peculiarities [13]. On the one hand, Tamburri *et al.* [69, 78] 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.* On the other hand, researchers have been studying community patterns and smells in isolation, for instance by assessing how community smells manifest themselves and can be mitigated [15, 14]. What is more, research is scarce in understanding the influence between such patterns, smells, and the software processes and products around them. An improved understanding of such relations is important to reveal whether and to what extent certain governance mechanisms are more incline to which improvement or pejoration over the software community's organizational conditions (*e.g.,* whether they incite the emergence of community smells). Furthermore, gaining such understanding would allow researchers to further investigate the problem, possibly proposing monitoring and/or remediation strategies.

In our previous work [24], we shed light on the aforementioned relations by studying how community patterns relate to community smells through association rule min-

Table 1: Overview of the software project communities considered in our study. The domain taxonomy is tailed from literature [9]. Projects marked with an asterisk (*) were only available for the first of three parts of the study (Section 4). The reported numbers refer up to 30th April 2017.

Name	#COMMITS	#CONTRIBUTORS	Main Lang.	Domain
Android*	314790	759	Java	Library
Arduino	6596	262	Java	Eletronics prototype platform
BoilerPlate*	469	48	JS	Web libraries and frameworks
Bootstrap	16665	1064	JS	Web libraries and frameworks
Boto	7282	682	Python	Web libraries and frameworks
Bundler	39236	761	Ruby	Web libraries and frameworks
Cloud9	9160	82	JS	Application Software
Composer	7409	774	PHP	Software Tools
Cucumber	600	23	Java	Software Tools
Ember.Js	17594	868	JS	Web libraries and frameworks
Gollum	1970	186	Ruby	Non-Web libraries and frameworks
Hammer.Js	1282	101	JS	Web libraries and frameworks
Hawkthorne	5568	82	Lua	Software Tools
Heroku*	353	48	Ruby	Software Tools
Modernizr	2412	260	JS	Non-Web libraries and frameworks
Mongoid	6932	497	Ruby	Non-Web libraries and frameworks
Monodroid*	1462	61	PHP	Non-Web libraries and frameworks
Netty	13308	419	Java	Software Tools
PDF.Js	9735	307	JS	Web libraries and frameworks
Refinery	14003	542	Ruby	Software Tools
Salt	86637	2599	Python	Software Tools
Scikit-Learn	23110	1146	Python	Non-Web libraries and frameworks
Scrapy	7063	298	Python	Non-Web libraries and frameworks
SimpleCV	2649	108	Python	Non-Web libraries and frameworks
SocketRocket	537	81	Objective-C	Non-Web libraries and frameworks

ing. By gathering data from 25 open-source communities, we exploited 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.

In this paper, we expand exploratively on the aforementioned work including how common software process and product characteristics relate to community patterns. We analyze the chosen systems relying on various third-party tools—namely YOSHI and CODEFACE4SMELLS (that we have already employed in our previous study) to detect community patterns and smells, respectively, while CLOC, MULTIMETRICS and GRIMOIRELAB for extracting product and process metrics. To account for an exploratory perspective, we adopt a statistical hypothesis-testing approach with the goal of pinpointing the relations (if any) among community patterns, software processes, and product metrics in our sample. We find that specific community patterns relate to equally specific organisational conditions; at the same time, we reveal that the current understanding over software community structures and their implicit/explicit adoption of community patterns is still limited and deserves further attention.

Our results have implications for both researchers and practitioners. Based on our findings, the former can further analyse 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.

To sum up, the contributions of this paper are:

1. A list of association rules between community patterns and community smells to see how they relate each other.
2. An empirical analysis on how community patterns affect the product quality.
3. An empirical analysis on how community patterns affect the community engagement in open-source projects.
4. A replication package [23] containing all the materials to reproduce and extend our study.

Structure of the paper. Section 2 discusses the literature related to community patterns and smells. Section 3 presents the research questions driving our study and the dataset exploited, while Section 4 and 5 describe the methodological detail and results of the research questions. In Section 6 we further discuss the main findings and their implications. The potential limitations of the study are reported in Section 7. Finally, Section 8 concludes the paper and presents our future research agenda.

2. 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) [62]. 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 [10] research, but without ever offering a theoretically- and empirically-established quality model. For instance, the research community concentrated on establishing the link between several organizational structure qualities (*e.g.,* hidden-subcontractors in the organizational structure [4, 6], awareness [8, 58], distance and coordination [36, 32], etc.) with respect to software quality [77]. Nagappan *et al.* [51] found empirical evidence of the existence of a relation between the development process and the product failure-proneness. Later on, Meneely and Williams [49] conducted an online survey to study the relationships between developer networks and SNA metrics. Their results have shown that developer networks represent the real-world socio-technical aspects described by those metrics. Nordio *et al.* [54] studied the impacts of distributed networks on the quality of the development process from the communication perspective. They conducted an experiment comparing the amount of communication in two-location and three-location projects showing that there is no statistical significant difference between the two network configurations. Jansen [40] 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 considering organizational structures or anti-patterns emerging thereto. Similarly, the work by Crowston and Howison [21] 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 use to avoid running into specific community smells.

At the other end of the spectrum, organization and social-network researchers proposed a plethora of organizational anti-patterns [63, 68], as well as (a few) best practices to address them [39, 80], with even fewer exceptions for open-source software communities [72]. For example, Giatsidis *et al.* [30] 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 [29]. 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; this might lead to achieving more practical insights.

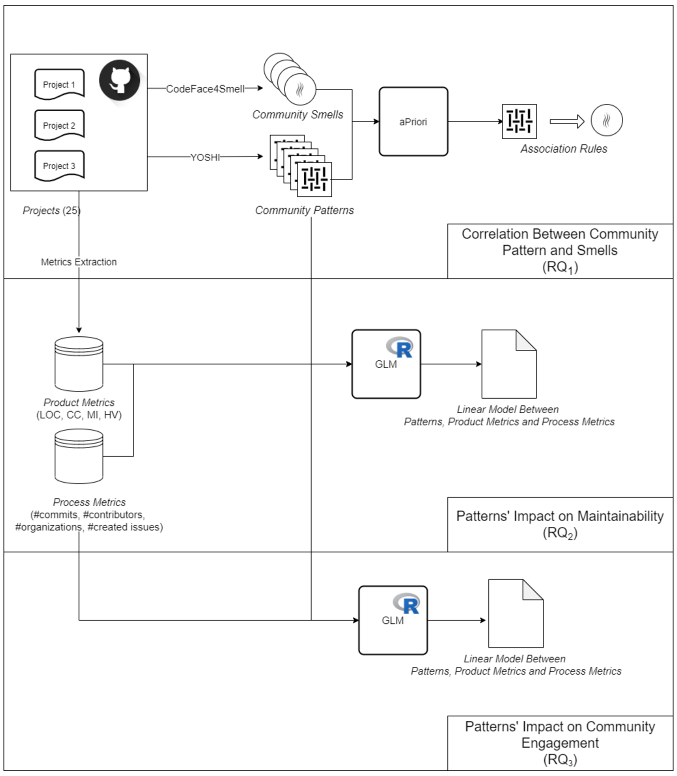
Research on community smells. In the last years, community smells have begun to receive particular attention [61, 78, 5]; one of the motivations resides in the development of the tool able to detect them called CODEFACE by Joblin *et al.* [42]. Indeed, the aforementioned tool was first augmented with heuristics capable to detect community smells [78] and then adopted to investigate the impact of community smells over code smells [61]. In the first place, Tamburri *et al.* [78] 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.* [61] 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 [47] and organization structure types [77]).

On another note, Catolino *et al.* [15] analysed that in certain cases the emergence of community smells may be potentially reduced by increasing gender diversity. In their extended work [12], however, they found that practitioners do not perceive gender diversity and the 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. Finally, more recently, Catolino *et al.* [16] conducted an empirical investigation on the prominence of four community smells in open source projects and on the methods adopted to refactor them. Results revealed that community smells manifest in software projects with high frequency and the evidence that developers adopt specific practices to refactor them.

Our work 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.* [78] have assessed the diffuseness of community smells in open-source projects; our analysis of the perceived relevance of community smells can nicely triangulate the findings by Tamburri *et al.* [78] and potentially show preliminary insights into the awareness of practitioners with respect to community smells.

3. Research Questions and Context Selection

The *goal* of this empirical study is to investigate the relations of community patterns with community smells and their effect on both a software system's maintainability and community engagement, with the *purpose* of understanding whether the structural organization of a community may potentially affect other community-, product- or process-related factors, from the *perspective* of both researchers and practitioners, who are interested in discovering the potential impact that a community structure has on the entire project.



Project 1
Project 2
Project 3
Projects (25)
CodeFace4Smell
YOSHI
Community Smells
Community Patterns
aPriori
Association Rules
Correlation Between Community Pattern and Smells (RQ_1)
Metrics Extraction
Product Metrics (LOC, CC, MI, HV)
Process Metrics (#commits, #contributors, #organizations, #created issues)
GLM
Linear Model Between Patterns, Product Metrics and Process Metrics
Patterns' Impact on Maintainability (RQ_2)
GLM
Linear Model Between Patterns, Product Metrics and Process Metrics
Patterns' Impact on Community Engagement (RQ_3)

Figure 1: Graphical representation of the applied methodology for each **RQ**. In the upper part it is shown the repository mining process with the used tools (YOSHI, CODEFACE4SMELLS, etc.) and the association rule mining between community patterns and smell, to answer **RQ**$_1$. In the middle and bottom portion it is represented how the extracted metrics from the same repositories were correlated with community patterns, to answer **RQ**$_2$ and **RQ**$_3$.

Our study uses the presence of community patterns as the *cause* construct and see which are their *effects* on some project aspects that we suspect may be influenced. On the one hand, given its exploratory nature, our study therefore makes use of metrics and constructs we conjecture may be influenced by the community dimensions under investigation. On the other hand, in this section we provide an overview of the research questions and conjectures driving our study and present some relevant information on the dataset employed to address them. It should be noted that, however, the selection of the constructs to examine reflects the exploratory nature of our study; at the same time, we recognise the need for future work invested in coherently and systematically identify and analyse additional metrics from both the process and product front.

3.1. Research Questions

Starting from the general goal, we structured our study around three main research questions (**RQ**s). The first investigates the relation between community patterns and community smells to understand whether certain smells tend to occur more often under specific community types.

> **RQ**$_1$ *What is the relation between community patterns and community smells?*

Once having assessed the links between community patterns and smells, we kept going forward investigating the potential impact of the patterns on the maintainability.

> **RQ**$_2$ *To what extent do community patterns affect the maintainability of a system?*

Finally, we studied the relations between community patterns and some aspects related to the engagement of the communities under analysis. Specifically, we considered their impact on the number of commits and the number of created issues.

> **RQ**$_3$ *To what extent do community patterns affect the engagement of a community?*

Figure 1 reports the methodology adopted to answer our three **RQ**s. To answer **RQ**$_1$, we first selected a set of 25 open-source software systems, then we extracted the patterns and smells of their communities, and finally we run the APRIORI algorithm [2] to mine associations between them. After that, to answer **RQ**$_2$ and **RQ**$_3$, from the same set of projects, we extracted several product and process related metrics, and we used them to build a Generalized Linear Model (GLM) [53] to find possible relations between community patterns and (i) the system's maintainability, (ii) the community engagement in terms of the number of commits and number of created issues.

3.2. Context of the Study

Projects. We considered 25 open-source software communities coming from the GITHUB software community management platform, sampled according to guidelines from the state of the art [25] and refined applying best-practice sampling criteria [44]. In particular, from the initial list of 81,327,803 open-source projects available, we first excluded systems having less than 10 contributors: such a filter was required to gather projects built by an actual community of developers. Then, we excluded systems with less than 100 commits—since they represent the most basic organisational manifestation of any software community—so that we could rely on a sufficient 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). Applying these filters, we came up with a total of 44,387,266 projects. Starting from this very large number of projects, we set out additional filters to restrict the size of the sample: we considered projects having at least (i) 1000 stars, (ii) 500 commits and (iii) 50 contributors. Applying these additional filters we came up with 3558 projects. Due to computational constraints, we randomly selected 25 of them, which we used to run the previous study (*i.e.,* $\mathbf{RQ}_1$); however, during the investigations of $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, 4 out of 25 projects' GitHub repositories were removed by their authors, preventing us to extract the product and process metrics needed to answer these two research questions. We decided not to replace these projects and to continue the study restricting to the remaining 21.

Table 1 summarizes the main characteristics of the extracted software projects in terms of (i) size, measured as the number of commits performed over their history, (ii) contributors, (iii) main programming language, and (iv) application domain, according to the taxonomy proposed by Borges *et al.* [9].

Community Patterns. In the following we report the list of types appearing in each community pattern emerged in our study. On the one hand, each type is accompanied by a brief description but reflects one of four meta-types [74], namely: (a) *communities*, which are social constructs made for sharing (*e.g.,* of values, norms, practices, etc.); (b) *networks*, which suggest the presence of digital or technological support tools to account for (physical, cultural or otherwise) distance of some kind; (c) *groups*, which are tightly knit sets of people or agencies that pursue an organizational goal; (d) *teams*, which emerge as specifically assembled sets of people with a diversified and complementary set of skills. On the other hand, community types manifest themselves into a *pattern*, made up of two or more types. In the scope of this study we start exploratively focusing on community patterns as manifestations of multiple types at once but, to give concrete inputs to the research and practitioner communities, also discuss the influence of each individual type over the observed characteristics. More concretely, types part of our study context are reported below:

– **Informal Communities (IC).** ICs reflect sets of people part of a 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 localization 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 [74]. Loosely-affiliated political movements (such as Greenpeace) are examples of ICs: their members disseminate their vision (based on a common idea, which is the goal of the IC).

– **Formal Networks (FN).** FNs rigorously select and prescribe memberships, which are created and acknowledged by FN management. The direction is carried out according to corporate strategy and its mission is to follow this strategy [74]. 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.

– **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 organizational 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.[1]

– **Informal Networks (IN).** INs are loose networks of ties between individuals that happen to come informally in contact in the same context. The primary indicator is the high strength of informal member ties. Finally, IN does not use governance practices [20]. 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.

– **Networks of Practice (NoP).** A NoP is a networked system of communication and collaboration that connects CoPs (which are localized). In principle, anyone can join it without the selection of candidates (e.g. Open-Source forges are an instance of NoP). NoPs have the highest geodispersion. An unspoken requirement is

[1] `http://ewh.ieee.org/cmte/psace/CAMS_taskforce/index.htm`

expected IT literacy [64]. For example, previous literature [7] discusses Socio-technical Networks in software engineering using the exact terms with which NoPs are defined in the literature.

- **Workgroups (WG).** WGs are made of technical experts whose goals span a specific business area. WGs are always accompanied by a number of organizational 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 organizational sponsors (e.g. UNESCO for IFIP).
- **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 [27]. An example is the SRII community[3] which gathers multiple CoPs (corporate and academic) into a single one, meeting physically to informally exchange best practices in services science.
- **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 organization and follow specific strategies or organizational guidelines (e.g. time-to-market, effectiveness, low-cost, etc.). Their final goal is the delivery of a product or service [74].
- **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) [82]. SNs act as a gateway to communicating communities [20].

All of them come from existing literature and include various forms of organizational structures. The choice of focusing on these types comes from the availability of an automated tool enabling their detection, *i.e.,* YOSHI [76]. 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 decision making process exercised or self-imposed on the community. In the second step, the tool implements a validated decision-tree that—on the basis of the measurements previously computed—is able to classify the organizational pattern exhibited by a community, intended as the set of community types matching the characteristics that YOSHI observes (*e.g.,* a formal or informal group mixed with a working group, as matched to high formality and collocation and high-cohesion of workers). It is important to highlight that the performance of YOSHI has been empirically assessed [76], showing an accuracy close to 100%. As such, the tool represents the ideal way to detect community patterns in our study.

From a practical point of view, each project community complies to one or more community patterns, so the best way to encode this aspect is by the means of multiple boolean variables, one per each of the nine types detected by YOSHI, indicating their presence (`True`) or absence (`False`). Since three of the patterns we considered—namely, Community of Practice (CoP), Projects-Teams (PT), and Social Networks (SN)—never occur in the selected 25 communities, we decided to remove their variables due to their little use in the context of this study. Thus, we make use of 6 different boolean variables. At the same time, a newer version of the tool (available online: `https://github.com/maelstromdat/YOSHI`) provides further validated typification facilities and allows replication and further study of the materials in this paper.

Community Smells. As another crucial part of our study context, community smells represent one of the possible manifestations emerging in connection to specific patterns. As such manifestations are established already in software engineering literature (*e.g.,* see Palomba et al. [61]) we chose to omit their elaboration and refer the reader to the background and related work section (see Section 2).

4. RQ$_1$ - On the Relation Between Community Patterns and Community Smells

4.1. Research Methodology

Community smells are the "social" counterpart of the well-known code smells affecting communities of developers. Just like code smells, they are indicators of the presence of social debt, and represent the result of accumulated bad or sub-optimal decisions in terms of people and organizations [70]. In the context of our study, we focused on some particular instances of smells:

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

[2]`http://www.softwarearchitectureportal.org/`
[3]`www.thesrii.org`

- **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.

- **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.

The reasons of this choice are as follows. In the first place, these specific smells have been shown by previous research [72, 61, 56, 41, 37] to be (1) among the most problematic community-related issues to deal with and (2) a potential threat to the emergence of technical debt and other nasty code-quality related phenomena [43, 78]. Secondly, these smells can be detected exploiting an automated tool named CodeFace4Smells [78]. This is a fork of CodeFace [43], 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. More specifically, the tool jointly uses (1) the coordination and communication graphs, (2) the dependency relations connecting these two graphs to discover antipatterns reflecting the community smells, e.g., so-called *cliques* [3] on the resulting graph which reflect *organizational siloes.*

Also for this tool, it is important to comment on its accuracy. CodeFace4Smells has been empirically evaluated [78, 72] 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 [78, 72] make us confident of the high reliability of the tool and its suitability for our study. What is more, the tool is currently undergoing re-implementation featuring an R-package based implementation which is bound to make the execution—and therefore the replication of the material in this paper—of the tool more straightforward. This new version, called Kaiaulu, is still under development and evaluation, but its repository is already available at: `http://itm0.shidler.hawaii.edu/kaiaulu/`.

To address $\mathbf{RQ}_1$ 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 and (3) association rule discovery.

To achieve the first step we exploited both Yoshi's and CodeFace4Smells' capabilities. First, we mined the repositories of the selected projects (see Section 3.2) with Yoshi to get the list of community patterns holding on those communities. Then we used CodeFace4Smells to detect the community smells affecting such communities. In order to properly analyze the communities and not to consider too much (*i.e.,* outdated) or insufficient data, we conducted the community analysis by covering a three months wide time window, as done in a previous study [77]. Once we gained the list of patterns and smells affecting the considered communities, we mined associations rules [1] to discover which community patterns and smells might 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 for each 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]:

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

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

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 [2], which is available in the `R` toolkit.[4] In Section 4.2 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

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

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 [26] 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 p-value lower than 0.05.

4.2. Results Analysis

Table 2 reports the association rules, grouped by community pattern, extracted after the application of the aPriori algorithm. 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 deeply analysed only a subset of the systems contained in our dataset. Specifically, we focused on the two projects, namely Arduino and SaltStack (*a.k.a.* Salt in our sample).

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 (FNs). 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 [79]. 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 [79]. In our case, think 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 [79].

On the other hand, other community types are quite prone to be smelly. In our dataset, Formal Groups (FGs) 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 fact that this community type is characterized by a excessively formal communication process, could reasonably lead to the rising of a 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 (ICs) 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 work 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, since developers cannot physically meet each other daily. The results were also confirmed looking at the lift value, which was higher than 1, and thus statistically significant (p-values lower than 0.05).

As for Informal Networks (INs), 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. In these cases, the lift values were higher than 1, while the p-values lower than 0.05, as well: for this reason, we can conclude that the relationships discovered are meaningful and statistically significant.

[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 2: 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	0.74	0.94	1.69	0.011
Informal Community → Lone Wolf	0.68	0.82	1.63	0.022
Informal Network → Lone Wolf	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

Finally, we found an unexpected connection between Networks of Practice (NoPs) 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 SALT project a single developer managed most of the communications performed by developers, becoming *de facto* a bottleneck. Interestingly, the lift value reached 1.59, with p-value equal to 0.032, confirming the relationship's significance.

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.*

5. RQ$_2$, RQ$_3$ - On the Impact of Community Patterns on Process and Product

In this section we discuss the research methodology and the results achieved when investigating our second and third research questions, the core of this extension.

5.1. Research Methodology

The *goal* of the two research questions is to analyze the impact that community patterns—along with other product and process characteristics—have (i) on the maintainability of a system (**RQ**$_2$) and (ii) on the community engagement, in terms of contributions and issues created in the project's repository (**RQ**$_3$).

Dependent Variable(s). For **RQ**$_2$, we measured the maintainability through the use of *Maintainability Index* (MI) [81] which is an aggregate of traditional source code metrics, used to indicate the degree of maintainability of a given system. This metric is able to capture different source code aspects which are related to maintainability, i.e., size, volume and complexity, irrespective of the programming language and paradigm—since we considered systems developed in different languages. Since its original formulation by Oman and Hagemeister [57], there have been many variants of such a metric, all of them combining the Lines Of Code (LOC), the McCabe's Cyclomatic Compexity (CC) [48] and one of the Halstead metrics (*i.e.,* Volume or Effort) [33] into a polynomial equation. In this paper we consider the *modified 3-metrics* variant, whose formula is reported below:

$$\begin{aligned} MI = 171 &- 5.2 \cdot ln(avgHV) \\ &- 0.23 \cdot avgCC \\ &- 16.2 \cdot ln(avgLOC) \end{aligned} \tag{3}$$

where *avgHV*, *avgCC* and *avgLOC* are, respectively, the average Halstead Volume, Cyclomatic Complexity, and Lines Of Code, computed over all the considered source code files. We exploited MULTIMETRICS,[8] an open-source automatic metrics extraction tool to get the average values of MI, over all the considered source code files, of the selected projects. Due to computational constraints, we run MULTIMETRICS only on the source files written in the main project language (*e.g.,* for SCIKIT-LEARN, only the `.py` files were considered). To avoid dissimilarity with the investigation of **RQ**$_1$, the metrics were computed for the state of each project as of 30th April 2017 (inclusive).

The investigation of **RQ**$_3$ focused on two different aspects explaining the engagement in open-source communities, *i.e.,* the number of contributions and the created issues, so we used two different dependent variables to separately analyze how they are affected by the presence of community patterns; in particular, we used, respectively, the number of commits (*i.e.,* #COMMITS) and the number of created issues (*i.e.,* #CREATED_ISSUES) until 30th April 2017 (inclusive), just as the investigation of **RQ**$_1$. Furthermore, considering that these two metrics are

[8]`https://pypi.org/project/multimetric/`

strongly influenced by the project sizes, *i.e.,* larger project tend to have a higher number of commits and issues, we divided them by the number of lines of code (LOC). As a result, we considered the dependent variables explained in the following equations:

$$commits\_intensity = \frac{\#commits}{LOC} \tag{4}$$

$$issues\_intensity = \frac{\#created\_issues}{LOC} \tag{5}$$

The extraction of such metrics was supported by the open-source tool GrimoireLab.[9]

Independent Variable(s). We used the presence/absence of the 6 considered community patterns, represented as 6 different boolean variables, as independent variables for both **RQ**$_2$ and **RQ**$_3$ (as explained in Section 3.2).

Confounding Factors. We suspected that the presence of community patterns could have a significant impact on both maintainability (**RQ**$_2$) and community engagement (**RQ**$_3$) only if they are considered together with other product- and process-related factors. For this reason, we considered additional confounding factors, collected with the support of different automatic analysis tools. Since the selected projects are implemented using different programming languages and paradigms (*i.e.,* procedural and object-oriented), we could not capture certain paradigm-specific aspects, such as cohesion and coupling [17], limiting our choice for the selection of the metrics. Thus, the extraction tools we exploited work with different languages and ignore paradigm-specific aspects. We used these three open-source tools to fulfill our needs:

- Cloc,[10] for extracting size metrics.
- Multimetrics (already used for MI) for extracting complexity metrics.
- GrimoireLab (already used for #commits and #created_issues), for extracting process metrics.

The complete list of selected confounding factors is reported in Table 3. Please, note that some of the confounding factors were used as dependant variables in some models: when doing so, they were not considered as confounding factors.

Statistical Modeling. For **RQ**$_2$ we defined the following set of null (Hn) and alternative (Ha) hypotheses, containing a pair of hypotheses per each involved community pattern:

Table 3: List of confounding factors considered in this study.

Group	Name	Description
Product	LOC	Number of Lines Of Code
	CC	Cyclomatic Complexity
	HV	Halstead Volume
	MI	Maintainability Index
Process	#commits	Number of Commits
	#contributors	Number of developers who committed at least once
	#organizations	Number of organizations that contributed with at least one commit[11]
	#created_issues	Number of created issues

$Hn_1(p)$: *The community pattern $p \in P$ has no impact on the maintainability index.*
$Ha_1(p)$: *The community pattern $p \in P$ has an impact on the maintainability index.*

where $P = \{IC, FN, FG, IN, NOP, WG\}$.

Since **RQ**$_3$ involves two different aspects describing the engagement of a community, we defined other two sets of hypotheses:

$Hn_2(p)$: *The community pattern $p \in P$ has no impact on the commits intensity.*
$Ha_2(p)$: *The community pattern $p \in P$ has an impact on the commits intensity.*
$Hn_3(p)$: *The community pattern $p \in P$ has no impact on the issues intensity.*
$Ha_3(p)$: *The community pattern $p \in P$ has an impact on the issues intensity.*

To test these hypotheses, we devise a Generalized Linear Model (GLM) [53] with the aim of relating the independent variables, along with the confounding factors, to the dependent variable(s). This statistical technique is used to fit a function describing the continuous response variable (*i.e.,* the maintainability index, the commits intensity and issues intensity in our case) relying on a set of categorical and/or numerical variables (in our case, the community patterns and the confounding factors). We used this statistical modeling approach for two reasons. On the one hand, it is able to analyze the simultaneous effects of both independent variables and confounding factors on the response variable [38]; on the other hand, it does not assume the underlying distribution of data to be normal: in our case, we have verified the normality of the distribution exploiting the Shapiro-Wilk test [65], which fails to reject its null hypothesis (*i.e.,* the data are not normally distributed) and, as such, we could rely on GLM [53].

Since the predictors of the model (*i.e.,* independent variables and confounding factors) might be linearly correlated to each other, which could negatively affect the interpretation of our linear modelling results [55, 50], we

[9]`https://chaoss.github.io/grimoirelab/`
[10]`https://github.com/AlDanial/cloc`

[11]We discuss the threats related to the detection of organizations in Section 7.

Table 4: Pairs of correlated confounding factors ($\rho^2 > 0.5$) for the models of $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. The discarded factors are reported in the 'Factor #2' column.

Dep. Variable	Factor #1	Factor #2	ρ^2
MI	CC	HV	0.53
	LOC	#COMMITS	0.62
	#CONTRIBUTORS	#CREATED_ISSUES	0.68
COMMITS_INTENSITY	CC	HV	0.51
	CC	MI	0.59
	#CONTRIBUTORS	#CREATED_ISSUES	0.72
ISSUES_INTENSITY	CC	HV	0.51
	CC	MI	0.59
	#CONTRIBUTORS	#COMMITS	0.71

made a *multicollinearity analysis* to exclude possible correlations among predictors. In particular, we exploited a hierarchical clustering method based on the Spearman's rank correlation coefficient [67] of the considered confounding factors (using the `varclus` function available in the R statistical toolkit[12]), and so, excluding from the model one of the two factors that showed a correlation value (ρ^2) higher than 0.5.

Among all the conflicting factors (*i.e.,* with $\rho^2 > 0.5$), we selected:

- **CC** over HV and MI because of its significant linear relation with MI in Equation 3. It should also be noted that similar relations have already been highlighted as problematic in software engineering [35, 66];
- **LOC** over #COMMITS because the Lines of Code are known to be negatively correlated with maintainability (see Equation 3) and they are easier to understand than number of commits;
- **#CONTRIBUTORS** over #CREATED_ISSUES because issues can be—and, for most medium to large-sized open-source projects in fact are [22]—created by non-contributing users [52] as well and who do not propose changes to the project code, so never directly affecting its maintainability. Given this selection, we discard #COMMITS, as well.

The main results of the multicollinearity analysis are reported in Table 4.

The models were built in a progressive manner to measure the explanatory power of different factors in a step-by-step fashion. In particular, three models were defined for each of the three analyses: (i) the first only considering the effects of the presence of community patterns, (ii) the second considering both the patterns and the selected product-related metrics, and (iii) a full model that involves both product and process metrics as confounding factors.

To sum up, the model's regressors consist of 6 independent variables (*i.e.,* the six different community patterns) and 4 confounding factors, namely LOC (only for $\mathbf{RQ}_2$), CC, #CONTRIBUTORS and #ORGANIZATIONS. Table 5 shows the descriptive statistics for the all numerical variables involved in $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. Our online appendix [23] provides the scripts and the dataset used for our empirical study.

[12]`https://bit.ly/2YFltBU`

Table 5: Descriptive statistics for numerical variables used in the models of $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. $N = 21$.

Variable	Type	Min	Max	Mean
MI	Dependent	38.30	89.30	65.30
COMMITS_INTENSITY	Dependent	0.016	1.19	0.25
ISSUES_INTENSITY	Dependent	0.00	0.60	0.09
CC	Co-factor	1.55	70.87	14.55
LOC	Co-factor	3,122.00	644,716.00	119,690.00
#CONTRIBUTORS	Co-factor	23.00	2,599.00	532.00
#ORGANIZATIONS	Co-factor	0.00	36.00	7.48

5.2. Results Analysis

This section reports the results analysis for both $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. The raw data extracted for answering the two research questions are reported Table 6.

***RQ$_2$* *Results*.** Table 7 reports the results for our second research question. The table describes the three models we created, *i.e.,* (i) Patterns, (ii) Patterns + Product and (iii) Full, to see the differences in terms of predictors significance.

The first and more important result is the significance of the predictor WG—corresponding to the presence/absence of a community pattern featuring a Workgroup—in each of the three models. Looking at the estimated coefficient (*i.e.,* Estimate column), we can see that there is a positive correlation with the dependent variable MI, implying that a system developed by a community that shows the characteristics of a Workgroup (WG=1) tends to have higher MI values, and so a better degree of maintainability. This relation could be explained by the definition of the Workgroups community pattern itself. Indeed, this pattern is characterized by a strong cohesion among the members of the team that are all focused on a specific business area. This aspect could imply that developers collaborate a lot with each other. Such a degree of collaboration leads to keep more attention to software maintainability since different developers make often changes on the same source files.

In support of this statement, we made an additional investigation and counted the mean number of contributors per file in projects that implement/do not exhibit the Workgroup pattern. To conduct this investigation, we split the list of projects reported in Table 1 into two sets: (i) projects implementing the Workgroup community pattern and (ii) projects implementing other community patterns. As a result, we noticed that Workgroup communities have, in general, a higher number of working developers per file, as shown in the box plots in Figure 2, hinting their higher degree of collaboration. This, however, needs further investigations.

Table 6: Raw data extracted in the context of $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ and used to build the GLM models. $N = 21$.

Project	Patterns	#COMMITS	#CONTRIBUTORS	MI	#ORGS.
Arduino	IC; FN; FG	6596	262	67.21	6
Bootstrap	IN; NOP	16665	1064	40.17	9
Boto	IC; IN	7282	682	57.74	34
Bundler	FG; NOP	39236	761	60.17	0
Cloud9	IC; FN; FG	9160	82	53.24	0
Composer	IC; FN; FG	7409	774	65.17	0
Cucumber	IN; NOP	600	23	80.15	2
Ember.Js	FN; WG	17594	902	74.43	15
Gollum	IC; FN; FG	1970	186	69.21	4
Hammer.Js	IN; NOP	1282	101	77.36	4
Hawkthorne	IC; FG; IN	5568	82	73.03	1
Modernizr	IN; NOP; WG	2412	260	89.26	5
Mongoid	FN; FG; WG	6932	497	72.86	5
Netty	IC; FN; FG	13308	419	68.00	32
PDF.Js	IN; NOP	9735	307	52.53	0
Refinery	FG; WG	14003	542	78.51	0
Salt	FN; FG; WG	86637	2599	56.00	36
Scikit-Learn	FG; IN; NOP	23110	1146	38.34	0
Scrapy	FN; FG; WG	7063	298	68.72	2
SimpleCV	IC; FG; IN; NOP	2649	108	60.01	0
SocketRocket	FG; IN; NOP	537	81	69.27	2

Table 7: The results of the GLM built for answering $\mathbf{RQ}_2$ - the impact of community patterns on **maintainability** (measured with MI). 'Estimate.' column reports the estimated regression coefficient (whose standard error is reported in 'Std.Err.' column) of the model. 'Signif.' column reports the significance of each coefficient according to the significance codes described at the bottom of the table. $N = 21$.

	Patterns			Patterns + Product			Full		
Term	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**
(Intercept)	57.02	14.94	**	69.26	10.93	***	75.66	10.47	***
FG	−4.25	7.81		−7.50	5.56		−7.59	5.69	
FN	−8.18	10.34		−1.13	7.49		−2.05	6.62	
IC	18.51	9.00	.	11.07	6.61		6.20	6.16	
IN	−7.77	11.21		−4.26	7.86		−4.46	7.06	
NOP	7.89	10.36		5.63	7.25		3.38	6.96	
WG	25.25	9.50	*	15.80	7.13	*	15.91	6.36	*
CC				−0.43	0.14	**	−0.41	0.14	*
LOC				−0.00	0.00		0.00	0.00	
Contributors							−0.01	0.00	.
Organizations							−0.02	0.26	

Significance codes: '***'$p < 0.001$, '**' $p < 0.01$, '*' $p < 0.05$, '.' $p < 0.1$

This first finding led us to reject the null hypothesis $Hn_1(WG)$ in favor of the alternative hypothesis $Ha_1(WG)$: *"the community pattern WG has an impact on the maintainability index"*. Unfortunately, none of the other patterns even became significant over the three iterations, thus we could not reject (either confirm) the other null hypotheses, *i.e.,* $Hn_1(p)$, where $p \in \{IC, FN, FG, IN, NOP\}$. More work needs to be enacted in this research direction and possibly with more evidence from a more varied selection of open-source projects to warrant for a more definitive conclusion.

Another variable that well explains our dependent one is CC, made significant in the second and third model. The estimated coefficient shows a negative sign, thus meaning that higher cyclomatic complexity values tend to decrease the maintainability. This finding is quite easy to explain, since complex code is less understandable and maintainable, as already stated by other research on the topic [31].

Finally, these results show a weak significance of IC in the first model, and a weak significance of the number of contributors (#CONTRIBUTORS) in the third model. While the latter shows also an Estimate of -0.01 and a Standard Error of 0 in the third model, meaning that it has very little impact on the dependent variable, the former is a bit trickier to explain: we note that in the second and third model, IC loses its significance, meaning that other confounding factors have a greater impact in explaining the model.

Finding 2. *The presence of community patterns featuring Workgroups (WG) has a positive impact on maintainability, meaning that projects maintained by WG-like communities tend to have higher maintainability in their respective codebases. The impact of other community patterns requires further investigation.*

Table 8: The results of the GLM built for answering $\mathbf{RQ_3}$ - the impact of community patterns on **commits intensity** (measured with #COMMITS / LOC). 'Estimate' column reports the estimated regression coefficient (whose standard error is reported in 'Std.Err.' column) of the model. 'Signif.' column reports the significance of each coefficient according to the significance codes described at the bottom of the table. $N = 21$.

	Patterns			Patterns + Product			Full		
Term	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**
(Intercept)	1.18	0.24	***	1.08	0.24	***	1.14	0.26	***
FG	−0.08	0.12		−0.05	0.12		−0.09	0.14	
FN	−0.80	0.16	***	−0.85	0.16	***	−0.82	0.17	***
IC	−0.15	0.14		−0.09	0.14		−0.10	0.16	
IN	−0.80	0.18	***	−0.83	0.17	***	−0.80	0.18	**
NOP	−0.13	0.16		−0.11	0.16		−0.17	0.18	
WG	−0.09	0.15		−0.01	0.15		−0.02	0.16	
CC				0.00	0.00		0.00	0.00	
Contributors							0.00	0.00	
Organizations							−0.00	0.01	

Significance codes: '***'$p < 0.001$, '**' $p < 0.01$, '*' $p < 0.05$, '.' $p < 0.1$

Table 9: The results of the GLM built for answering $\mathbf{RQ_3}$ - the impact of community patterns on **issues intensity** (measured with #CREATED_ISSUES / LOC). 'Estimate.' column reports the estimated regression coefficient (whose standard error is reported in 'Std.Err.' column) of the model. 'Signif.' column reports the significance of each coefficient according to the significance codes described at the bottom of the table. $N = 21$.

	Patterns			Patterns + Product			Full		
Term	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**	**Estimate**	**Std. Err.**	**Signif.**
(Intercept)	0.31	0.17	.	0.21	0.16		0.23	0.18	
FG	−0.11	0.09		−0.08	0.08		−0.10	0.10	
FN	0.01	0.12		−0.03	0.11		−0.02	0.12	
IC	−0.14	0.10		−0.08	0.10		−0.08	0.11	
IN	−0.06	0.13		−0.08	0.11		−0.08	0.13	
NOP	−0.05	0.12		−0.03	0.10		−0.05	0.12	
WG	−0.15	0.11		−0.07	0.10		−0.07	0.11	
CC				0.00	0.00	.	0.00	0.00	
Contributors							0.00	0.00	
Organizations							−0.00	0.00	

Significance codes: '***'$p < 0.001$, '**' $p < 0.01$, '*' $p < 0.05$, '.' $p < 0.1$

RQ_3 Results. Table 8 shows the results of the first set of hypotheses of $\mathbf{RQ_3}$, regarding the impact of community patterns on the commits intensity, while Table 9 depicts the results for the second set of the hypotheses, the ones regarding the impact on the issues intensity. Similarly as what was done for $\mathbf{RQ_2}$, the tables report the three incremental models outcomes, namely (i) Patterns, (ii) Patterns + Product and (iii) Full.

Considering the first set of model, it is possible to appreciate that in all the three computed models, FN and IN have a high statistical significance (as well as the intercept), meaning that they have a strong influence over the intensity of commits. This is also quite understandable, taking into account the nature of the considered community patterns. On the one hand, Formal Networks (FN) are very formal communities, and by their nature, contributions (commits in this case) must follow a strict protocol to be accepted. On the other hand, Informal Networks (IN) do not have such rigid rules, so it is potentially easier to contribute to a project. In both cases, either positively or negatively, the presence of these patterns highly influence how much people contribute to a project. These results allow us to reject $Hn_2(FN)$ and $Hn_2(IN)$ null hypotheses, in favor of the alternative hypotheses $Ha_2(FN)$ (*"the community pattern FN has an impact on the number of commits"*) and $Ha_2(IN)$ (*"the community pattern IN has an impact on the number of commits"*). Unfortunately, none of the other patterns even became significant over the three iterations, thus we could not reject (either confirm) the other null hypotheses, *i.e.,* $Hn_2(p)$, where $p \in \{IC, FG, NOP, WG\}$. More work needs to be enacted in this research direction and possibly with more evidence from a more varied selection of open-source projects to warrant for a more definitive conclusion.

When considering, however, the issues intensity as dependent variable, we cannot find important conclusions regarding community patterns. Indeed, looking at Table 9, we can see that none of the models report any statistical significance for the community patterns.

It is worth noting that issues are generally created later after the addition of a feature or, in general, after a release. Thus, it is more likely that the issues regarding a product developed under a particular governance mechanism (*i.e.,* community pattern) are created *outside* the consid-

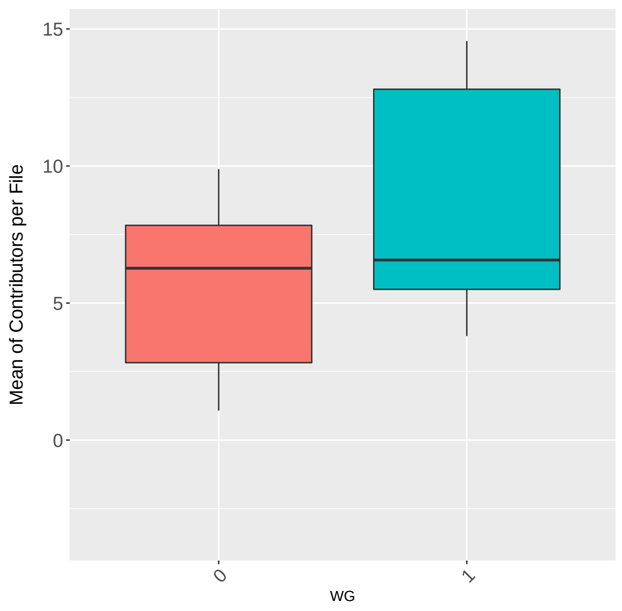


Figure 2: Mean of contributors per file in projects that exhibit($WG = 1$) vs. do not exhibit ($WG = 0$) the Workgroup community pattern. $N = 21$.

ered time window, rather than within it, meaning that the effects of a community pattern are not immediately visible on the short term.

Although these results do not allow us to reject any of the second set of null hypotheses $Hn_3(p)$, they suggest a good starting point for a further research, aimed to investigate the mid/long term effects of community patterns on the quality of the product and on the engagement of the community.

Finding 3. *IN and FN community patters have shown to have a significant impact over the intensity of commits made to a project. However, no community pattern showed a significant impact on the intensity of created issues. This result could be due to the fact that we considered a limited time window and issues (introduced using a certain organization structure) could have been created later. Our conclusion is that other investigations with additional data are required.*

6. Discussion and Implications

6.1. Practical Implications: Software Community Management and Beyond

The theoretical underpinnings of the present work assume that a community management protocol can be designed and operationalized to support the data-driven governance for specific software community targets, namely, that a specific community A aiming for stability might require specific protocol actions A_x and A_y. Our results indicate already some of the potential such actions. For example, suppose a commmunity has an overly high number of so-called lone-wolves; our results indicate that an unintended *polarisation* maybe exists in the community which drives community contributions towards an overly *formal* group or an overly *informal* community. In this instance, the consequent actions would require adopting specific software community management tactics to *influence* the re-design of the community in the desired direction (*e.g.,* by adding code contribution protocols to an overly informal community). On the one hand, the study and practical elaboration of the aforementioned tactics enabled by this and similar studies are still very much in their infancy. On the other hand, the growth and turmoil that often interests both open- and closed-source communities alike demands immediate action. From the perspective of this study however, we regard the association rules in Tab. 2 as opportunities to start the preparation of the aforementioned tactics, using them as the roots for factual actions that can be undertaken towards community re-design.

Beyond the aforementioned community management implications, our study also reflects on the negative consequences connected to acting short on community aspects and their management. For example, on the one hand our data indicates that work-groups have a peculiar relation with the patterns found and therefore deserve special attention in the future of software maintainability; for example, if maintainability is indeed a feature of WGs then perhaps every software community prone to long-lived software should have a community component designed to operate like such a WG community type. On the other hand, our study results also indicate that patterns themselves are, for example, somewhat linked with the amount of contributions reported for the emerging software artefacts catered by the communities reflecting those patterns in a non-trivial relation that our fairly limited exploratory study was not able to narrow down.

While we do not have enough data to conjecture on the implications of the non-trivial relations highlighted before, one observation is clear: the relation between community patterns, smells, and the underlying software product and process are very much non-trivial and will likely require considerable systematic empirical research in the future. For example, research questions such as *"what community conditions are reflected by software architectures that deliberately simplify software complexity?"* but even simply *"how can community governance benefit from architectural tactics to redesign communities into more efficient forms?"*. Questions such as the ones highlighted before have a considerable practical implication and are bound to play a key role in the future pages of social software engineering and software community management in practice.

6.2. Community Management in Practice: A focused look

To give a better account of the findings and implications of our study, let us take a deeper dive into two major communities—the largest in our sample in fact—namely, ARDUINO and SALTSTACK. Both community have a strong tendency towards structure and formality, with a lesser extent at that within the ARDUINO community. On the one hand, both communities basically sit at two opposites with respect to the number of organizations featured inside the respective communities (*i.e.,* 6 for ARDUINO and 36 for SALTSTACK). On the other hand, the presence of a lower number of organisations involved seems to justify the lower formality in ARDUINO when compared to SALTSTACK, which appears as a more structured and co-organized counterpart in this comparison.

Future investigations could look further into these such relations which appear to be non-trivial and yet have a clear yield over the maintainability and long-term sustainability of community structures as well as their maintained software. What is more, the framework we propose in the scope of this paper does not warrant any further elaboration on the process of *re-design* or "rewiring" of organisational structures to migrate from one form (say the ARDUINO *IC-FN-FG* form) to another and yet this migration exercise could warrant any number of beneficial community characteristics, starting from a higher software maintainability to community features [73] that are currently unknown. For example, in a hypothetical ordering of the patterns we discovered in the scope of this and precedent works (*e.g.,* see the proposal in Figure 3) with respect to the dimensions of *globality* (X-axis) and *formality* (Y-axis), the patterns corresponding to ARDUINO and SALTSTACK appear under an unprecedented lens of analysis, one which looks at the tradeoffs between organisational conditions across those communities and the technical characteristics warranted by such tradeoffs.

Focusing back on our ARDUINO vs. SALTSTACK case, the two patterns reflect perhaps two different stages of maturity of the organisation where, if more than one organisation plays a role, then a more formal organisational structure ensues, moving the pattern higher up in the schema. At this point, theoretical models such as the one represented in Figure 3 need further work and validation, perhaps starting from a replication of this study with larger, longitudinal datasets.

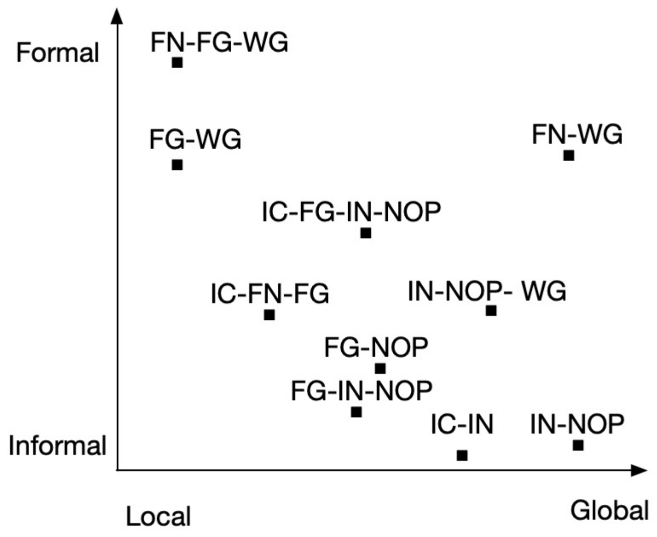


Figure 3: Arranging software community structure patterns, a 2-dimensional hypothesis space.

6.3. Observations and Lessons Learned

From an overall perspective, two observations become evident.

First, there is in fact a considerable influence played by community types over the processes and products of software life-cycles. While certain patterns featuring specific community types—*e.g.,* our reports over the Workgroups type—offer a more evident influence and occurrence, a general understanding on which organisational approach (featuring WGs or other types) is most appropriate to which software activity remains undetermined. What is more, from several statistical analyses stemming from our study it becomes evident that a larger-scale, perhaps longitudinal study is needed to find specific organisational patterns best fitting software endeavours.

Second, the relevant occurrence of specific community smells in combination with specific types leads to conjecturing that there exist *most-effective* community patterns wherefore specific community smells are mitigated *by-design.* For example, we found no evidence of silo effects within none of the most formal (*i.e.,* Formal Groups) and semi-formal community types (*i.e.,* NoPs) which may indicate that the stricter working dynamics typical of those types inhibit independent emergence of sub-groups. At the same time, specific community smell effects, such as the Lone Wolf effect, seem to manifest more or less indiscriminately in many types. In summary, the organisational health parameters and thresholds for the aforementioned effectiveness of patterns remain to be determined as well. The aforementioned study most likely needs to involve a rigorous data-driven investigation of known community smells and patterns together with a varied array of software quality metrics and parameters as well.

7. 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.

7.1. 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, not only both Yoshi and CodeFace4Smells were previously validated [76, 69, 75, 43], showing good detection capabilities, but also Multimetrics, Cloc, and GrimoireLab can be considered as reliable measurement tools. Furthermore, in order to avoid possible misuse or misinterpretation of the results, we carefully followed the tools' official documentation. This increases the reliability of our study and make us confident of the accuracy of the collected data. Despite these countermeasures, we could not avoid possible wrong interpretations caused by the way the tools compute the metrics. As an example, GrimoireLab computes an estimate of the number of organizations based on the database built by the SortingHat[13] module, which aggregates data from different sources. Namely, GrimoireLab maps the various organizations to developers based on their *email domain*—for instance contributors' emails having the `bitergia.com` domain are assigned to Bitergia organization. Because of this assumption, GrimoireLab could still detect no organizations in a project. This limitation may underestimate the actual number of organizations involved in a project.

Both Multimetrics and Cloc work at the granularity level of a single source file, requiring us to aggregate the metrics they computed to bring them to the project granularity level. Namely, we summed up the size values together, while we averaged the other ones. Moreover, due to computational constraints, we run them only on the source file written in the main project language, possibly losing important information from other source files written in a different language. Another potential threat to construct validity could be the operationalisation of *communication* and *collaboration*, as they come only from the mining of GitHub commit history and issues conversation. Although there might be many other channels for communication (*e.g.,* Slack) and issue tracking (*e.g.,* Jira), we find that the sole mining of GitHub is reliable enough to our extent. As a matter of fact, we manually checked whether the conversations among developers were taking place on GitHub, by inspecting the level of use of the integrated issue tracker within each of the considered repositories. We found that all of the 21 projects actively open and resolve GitHub issues, hinting that most of the communication happens there.

Another threat to the validity of the study might be represented by the usage of the maintainability index. We are aware that MI has some limitations when used as an indicator of the overall maintainability of a software system. As a matter of fact, its effectiveness has been the subject of several studies [34, 19, 45, 18, 28], highlighting its poor explainability and practicality—e.g., identifying the best possible actions to put in place to improve the MI value [34]. We still opted for this metric because of its capability to summarize the maintainability status of software projects as a whole with just a single comparable value, as well as its language and paradigm independence. Over the last two decades, there have been some efforts in replacing the current MI with a wide range of software metrics that could describe different sub-characteristics of maintainability [34]. The use of multiple dependent variables to describe the maintainability from different points of view was left out from this study, and it is part of our research agenda.

The period under consideration, *i.e.,* a three-months-wide time window, might also be another potential threat to the validity of the study, since we might have neglected some aspects which were more evident on a larger (or narrower) time interval. However, this was a design choice made by the developer of both of the tools that we have employed, which is supported by previous studies [78, 77]. In fact, this time window not only is the is required to observe a meaningful organisational activity which reflects an observable and reasonably stable organizational configuration, but also allows to correctly analyze the organisational aspects of a software community excluding outdated information.

Finally, another potential threat to construct validity could be related to the indirect relations between the selected metrics (*i.e.,* larger systems, in terms of LOC, are likely to have a higher number of commits, issues, contributors, etc.). To mitigate this threat, we exploited intensive metrics to describe our dependent variables in RQ_3, namely commits intensity and issues intensity. However, independent and confounding variables could suffer from this problem as well. For instance, it is reasonable to think that also other metrics, such as the cyclomatic complexity or the number of contributors could be related to LOC. Therefore, as future work, we plan to focus on a wider set of intensive metrics, given that in the context of this study we mostly focused on extensive ones.

7.2. 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.,* [60, 59, 83]), whose validation process involved not only statistical evaluation, but also a survey study actively involving the interested communities. 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.

Another threat is related to the actual suitability of the employed statistical method, *i.e.,* Generalized Linear Model. In this regard, before selecting it we verified the assumptions that the model makes on the underlying data.

[13]`https://github.com/chaoss/grimoirelab-sortinghat`

Moreover, to strengthen the causality links between dependent and independent variables, we involved a set of confounding factors in order to discover possible additional causes that may have an impact on the dependent variables. We are aware of the fact that we selected only a limited number of process-related factors, which could under-represent the actual process dynamics of the projects. Indeed, we could have left out some other relevant process metrics, such as the number of re-opened issues, thus causing possible biased interpretations of the results obtained from $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$. However, since some of them resulted to be too difficult to compute and that the main focus of the work was to investigate the effects caused by the different community patterns, we selected a set of metrics that were easier to compute and explain.

7.3. Threats to External Validity

The main issue concerned with the generalisation of the results is the number of software communities analysed in the study. While a set of 25 systems is not a statistically significant sample of the most active projects present in GitHub, it allowed us 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 first research question. In addition, to make our findings as generalisable 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 to a larger set of communities. However, as consequence of this trade-off, the analysis performed to answer $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$ suffered for the lack of a good amount of observations (*i.e.,* projects), requiring us a replication of the study with additional community data in the future. Currently, on the basis of these results, we can only partially answer $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$.

The choice of focusing on certain community patterns and smells might be a threat to the generalisation 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.

8. Conclusion and Future Work

In this paper, we studied the relation between community patterns and a set of community-, product- and process-related factors 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 the (i) community smells that affect the same set of projects using CodeFace4Smells and (ii) some key product and process metrics using Multimetrics, Cloc and GrimoireLab. Finally, we (i) exploited association rule mining, particularly the aPriori algorithm, to discover relations between them and (ii) built multiple Generalized Linear Models that related the patterns, along with product and process metrics, to the maintainability of the system and the quality of the development process. The key findings of the study show that: (1) specific community smells may arise depending on the peculiarities of the community organization; (2) specific community patterns may have a direct impact on short term process and product attributes.

From a practical perspective, our results aid practitioners to prevent the occurrence of targeted community smells, as well as raising the community managers' potential awareness of which community patterns may be active within the community and generating said smells as well. Furthermore, the statistical investigation reported in this paper brings about findings over the nature of community types, as well as their proneness to which specific influences over software processes and products.

In the future, we plan to investigate the discovered relations more rigorously as part of additional and follow-up longitudinal studies. We are planning, for example, to replicate these studies on a larger dataset, considering a wider time windows, in the attempt of answering the questions left pending. Furthermore, we plan to conduct a qualitative study, *i.e.,* a survey or semi-structured interviews, to further analyse the investigated communities and the correlation between community pattern and smells, as well as their impact on process and product quality attributes. Finally, in the future we plan to more systematically investigate the known metrics reflecting software process and product characteristics in a wider selection of settings (*e.g.,* industrial as well as hybrid open-/closed-source) with the community aspects in our study, with the goal to scope out the phenomenon we addressed in this exploratory study.

Acknowledgements

This work is supported by the European Commission grant no. 825480 (SODALITE H2020) and no. 825040 (RADON H2020). We thank all members of the SODALITE and RADON consortia for their inputs and feedback to the development of this paper. Furthermore, we thank the editor and anonymous reviewers for the invaluable feedback over this paper.

References

[1] Agrawal, R., Imieliński, T., Swami, A., 1993. Mining association rules between sets of items in large databases. SIGMOD Rec. 22, 207–216. doi:10.1145/170036.170072.

[2] Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules in large databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. p. 487–499.

[3] Alba, R.D., 1973. A graph-theoretic definition of a sociometric clique. Journal of Mathematical Sociology 3, 3–113. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.89.8697.

[4] Albino, V., Garavelli, A.C., 1998. A neural network application to subcontractor rating in construction firms. International Journal of Project Management 16, 9–14.

[5] Almarimi, N., Ouni, A., Chouchen, M., Saidani, I., Mkaouer, M.W., 2020. On the detection of community smells using genetic programming-based ensemble classifier chain, in: Proceedings of the 15th International Conference on Global Software Engineering, pp. 43–54.

[6] Baron, D.P., Besanko, D., 1992. Information, control, and organizational structure. Journal of Economics & Management Strategy 1, 237–275.

[7] Bird, C., Nagappan, N., Gall, H., Murphy, B., Devanbu, P., 2009. Putting it all together: Using socio-technical networks to predict failures, in: Proceedings of the 2009 20th International Symposium on Software Reliability Engineering, IEEE Computer Society, Washington, DC, USA. pp. 109–119. doi:10.1109/ISSRE.2009.17.

[8] Bloodgood, J.M., Morrow Jr, J., 2003. Strategic organizational change: exploring the roles of environmental structure, internal conscious awareness and knowledge. Journal of Management Studies 40, 1761–1782.

[9] Borges, H., Hora, A., Valente, M.T., 2016. Understanding the factors that impact the popularity of github repositories., in: IEEE International Conference on Software Maintenance and Evolution, IEEE, –. pp. 334–344.

[10] Cataldo, M., Herbsleb, J.D., Carley, K.M., 2008a. 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. pp. 2–11. URL: http://doi.acm.org/10.1145/1414004.1414008, doi:http://doi.acm.org/10.1145/1414004.1414008.

[11] Cataldo, M., Herbsleb, J.D., Carley, K.M., 2008b. 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, ACM, New York, NY, USA. pp. 2–11. URL: http://portal.acm.org/citation.cfm?id=1414008, doi:http://doi.acm.org/10.1145/1414004.1414008.

[12] Catolino, G., Palomba, F., Tamburri, D., Serebrenik, A., Ferrucci, F., 2019a. Gender diversity and community smells: Insights from the trenches. IEEE Software .

[13] Catolino, G., Palomba, F., Tamburri, D.A., . The secret life of software communities: What we know and what we don't know .

[14] Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F., 2019b. Gender diversity and community smells: insights from the trenches. IEEE Software 37, 10–16.

[15] Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F., 2019c. 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. pp. 11–20.

[16] Catolino, G., Palomba, F., Tamburri, D.A., Serebrenik, A., Ferrucci, F., 2020. Refactoring community smells in the wild: the practitioner's field manual, in: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Society, pp. 25–34.

[17] Chidamber, S.R., Kemerer, C.F., 1994. A metrics suite for object oriented design. IEEE Transactions on Software Engineering .

[18] Coleman, D., Ash, D., Lowther, B., Oman, P., 1994. Using metrics to evaluate software system maintainability. Computer 27, 44–49. doi:10.1109/2.303623.

[19] Counsell, S., Liu, X., Eldh, S., Tonelli, R., Marchesi, M., Concas, G., Murgia, A., 2015. Re-visiting the 'maintainability index' metric from an object-oriented perspective, in: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications, pp. 84–87. doi:10.1109/SEAA.2015.41.

[20] Cross, R., Liedtka, J., Weiss, L., 2005. A practical guide to social networks. Harvard Business Review , –.

[21] Crowston, K., Howison, J., 2005. The social structure of free and open source software development. First Monday 10.

[22] Crowston, K., Shamshurin, I., 2017. Core-periphery communication and the success of free/libre open source software projects. J. Internet Serv. Appl. 8, 10:1–10:11. URL: http://dblp.uni-trier.de/db/journals/jisa/jisa8.html#CrowstonS17.

[23] De Stefano, M., Pecorelli, F., Iannone, E., Tamburri, D.A., 2021. Impacts of software community patterns on process and product: An empirical study. URL: https://figshare.com/articles/dataset/Impacts_of_Software_Community_Patterns_on_Process_and_Product_An_Empirical_Study/13140182/3, doi:10.6084/m9.figshare.13140182.v2.

[24] De Stefano, M., Pecorelli, F., Tamburri, D.A., Palomba, F., De Lucia, A., 2020. Splicing community patterns and smells: A preliminary study, in: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, Association for Computing Machinery, New York, NY, USA. p. 703–710. URL: https://doi.org/10.1145/3387940.3392204, doi:10.1145/3387940.3392204.

[25] Falessi, D., Smith, W., Serebrenik, A., 2017. Stress: A semi-automated, fully replicable approach for project selection., in: ESEM, IEEE. pp. 151–156. URL: http://dblp.uni-trier.de/db/conf/esem/esem2017.html#FalessiSS17.

[26] Fisher, R.A., 1922. On the Interpretation of chi^2 from Contingency Tables, and the Calculation of P. Journal of the Royal Statistical Society 85, 87–94. URL: http://dx.doi.org/10.2307/2340521, doi:10.2307/2340521.

[27] Gallagher, S., 2006. Introduction: The arts and sciences of the situated body. Janus Head 9, 1–2.

[28] Ganpati, A., Kalia, A., Singh, H., 2012. A comparative study of maintainability index of open source software.

[29] Giatsidis, C., Thilikos, D.M., Vazirgiannis, M., 2011. Evaluating cooperation in communities with the k-core structure, in: 2011 International conference on advances in social networks analysis and mining, IEEE. pp. 87–93.

[30] Giatsidis, C., Thilikos, D.M., Vazirgiannis, M., 2013. D-cores: measuring collaboration of directed graphs based on degeneracy. Knowledge and information systems 35, 311–343.

[31] Gill, G., Kemerer, C., 1992. Cyclomatic complexity density and software maintenance productivity. Software Engineering, IEEE Transactions on 17, 1284 – 1288. doi:10.1109/32.106988.

[32] Grinter, R.E., Herbsleb, J.D., Perry, D.E., 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. pp. 306–315.

[33] Halstead, M., 1977. Elements of software science.

[34] Heitlager, I., Kuipers, T., Visser, J., 2007. A practical model for measuring maintainability, in: 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007), pp. 30–39. doi:10.1109/QUATIC.2007.8.

[35] Henderson-Sellers, B., Tegarden, D., 1994. A critical re-examination of cyclomatic complexity measures., in: Lee, M.K.O., Barta, B.Z., Juliff, P. (Eds.), Software Quality and Productivity, Chapman & Hall. pp. 328–335. URL: http://dblp.uni-trier.de/db/conf/icsqp/icsqp1994.html#Henderson-SellersT94.

[36] Herbsleb, J.D., Grinter, R.E., 1999. Architectures, coordination, and distance: Conway's law and beyond. IEEE software 16, 63–70.

[37] Hislop, D., 2004. Knowledge Management In Organizations: A Critical Introduction. Oxford University Press. URL: http://www.amazon.co.uk/exec/obidos/ASIN/0199262063/citeulike-21.

[38] Højsgaard, S., Halekoh, U., Yan, J., 2005. The r package geepack for generalized estimating equations. Journal of Statistical Software, Articles 15, 1–11. URL: https://www.jstatsoft.org/v015/i02, doi:10.18637/jss.v015.i02.

[39] Ito, K., Washizaki, H., Fukazawa, Y., 2016. Handover anti-patterns, in: Proceedings of the 5th Asian Conference on Pattern Language of Programs (Asian PLoP 2016), Taipei, Taiwan.

[40] Jansen, S., 2014. Measuring the health of open source software ecosystems: Beyond the scope of project health. Information and Software Technology 56, 1508–1519.

[41] Jarvenpaa, S.L., Leidner, D.E., 1999. Communication and trust in global virtual teams. ORGANIZATION SCIENCE 10, 791–815. URL: http://orgsci.journal.informs.org/cgi/content/abstract/10/6/791, doi:10.1287/orsc.10.6.791.

[42] Joblin, M., Mauerer, W., Apel, S., Siegmund, J., Riehle, D., 2015a. From developer networks to verified communities: A fine-grained approach, in: Proceedings of the 37th International Conference on Software Engineering - Volume 1, IEEE Press, Piscataway, NJ, USA. pp. 563–573. URL: http://dl.acm.org/citation.cfm?id=2818754.2818824.

[43] Joblin, M., Mauerer, W., Apel, S., Siegmund, J., Riehle, D., 2015b. 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. pp. 563–573. doi:10.1109/ICSE.2015.73.

[44] Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., Germán, D.M., Damian, D.E., 2016. An in-depth study of the promises and perils of mining github. Empirical Software Engineering 21, 2035–2071. URL: http://dblp.uni-trier.de/db/journals/ese/ese21.html#KalliamvakouGBS16.

[45] Kaur, K., Singh, H., 2011. Determination of maintainability index for object oriented systems. SIGSOFT Softw. Eng. Notes 36, 1–6. URL: https://doi.org/10.1145/1943371.1943383, doi:10.1145/1943371.1943383.

[46] Kwan, I., Schroter, A., Damian, D., 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, 307–324. doi:10.1109/TSE.2011.29.

[47] Martini, A., Bosch, J., 2017. Revealing social debt with the caffea framework: An antidote to architectural debt., in: ICSA Workshops, IEEE Computer Society. pp. 179–181. URL: http://dblp.uni-trier.de/db/conf/icsa/icsaw2017.html#MartiniB17.

[48] McCabe, T.J., 1976. A complexity measure. IEEE Transactions on Software Engineering SE-2, 308–320. doi:10.1109/TSE.1976.233837.

[49] Meneely, A., Williams, L., 2011. Socio-technical developer networks: Should we trust our measurements?, in: Proceedings of the 33rd International Conference on Software Engineering, pp. 281–290.

[50] Morrison, C., 2003. Interpret with caution: Multicollinearity in multiple regression of cognitive data. Perceptual and motor skills 97, 80–2. doi:10.2466/PMS.97.5.80-82.

[51] Nagappan, N., Murphy, B., Basili, V., 2008. The influence of organizational structure on software quality, in: 2008 ACM/IEEE 30th International Conference on Software Engineering, IEEE. pp. 521–530.

[52] Nakakoji, K., Yamamoto, Y., NISHINAKA, Y., Kishida, K., Ye, Y., 2003. Evolution patterns of open-source software systems and communities. International Workshop on Principles of Software Evolution (IWPSE) doi:10.1145/512035.512055.

[53] Nelder, J.A., Wedderburn, R.W., 1972. Generalized linear models. Journal of the Royal Statistical Society: Series A (General) 135, 370–384.

[54] Nordio, M., Estler, H.C., Meyer, B., Tschannen, J., Ghezzi, C., Di Nitto, E., 2011. How do distribution and time zones affect software development? a case study on communication, in: 2011 IEEE Sixth International Conference on Global Software Engineering, IEEE. pp. 176–184.

[55] O'Brien, R., 2007. A caution regarding rules of thumb for variance inflation factors. Quality & Quantity 41, 673–690. doi:10.1007/s11135-006-9018-6.

[56] Olsson, J., Sandell, J., 2009. Enterprise 2.0 as a way to facilitate, enhance, and coordinate intelligence work within large organizations: A case study at toyota material handling europe. 3rd European Competitive Intelligence Symposium (ECIS 2009) .

[57] Oman, P., Hagemeister, J., 1994. Construction and testing of polynomials predicting software maintainability. Journal of Systems and Software 24, 251 – 266. URL: http://www.sciencedirect.com/science/article/pii/0164121294900671, doi:https://doi.org/10.1016/0164-1212(94)90067-1. oregon Workshop on Software Metrics.

[58] Oomes, A., 2004. Organization awareness in crisis management, in: Proceedings of the international workshop on information systems on crisis response and management (ISCRAM).

[59] Palomba, F., Bavota, G., Di Penta, M., Fasano, F., Oliveto, R., De Lucia, A., 2018a. A large-scale empirical study on the lifecycle of code smell co-occurrences. Information and Software Technology 99, 1–10.

[60] Palomba, F., Bavota, G., Penta, M.D., Oliveto, R., Poshyvanyk, D., Lucia, A.D., 2015. Mining version histories for detecting code smells. IEEE Transactions on Software Engineering 41, 462–489. doi:10.1109/TSE.2014.2372760.

[61] Palomba, F., Tamburri, D.A.A., Fontana, F.A., Oliveto, R., Zaidman, A., Serebrenik, A., 2018b. Beyond technical aspects: How do community smells influence the intensity of code smells? IEEE Transactions on Software Engineering .

[62] Palomba, F., Zanoni, M., Fontana, F.A., De Lucia, A., Oliveto, R., 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. pp. 244–255.

[63] Persson, A., Stirna, J., 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. pp. 243–252.

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

[65] Shapiro, S.S., Wilk, M.B., 1965. An analysis of variance test for normality (complete samples)†. Biometrika 52, 591–611. URL: https://doi.org/10.1093/biomet/52.3-4.591, doi:10.1093/biomet/52.3-4.591.

[66] Shepperd, M., 1988. A critique of cyclomatic complexity as a software metric. Software Engineering Journal 3, 30–36.

[67] Spearman, C., 2010. The proof and measurement of association between two things. International Journal of Epidemiology 39, 1137–1150. URL: https://doi.org/10.1093/ije/dyq191, doi:10.1093/ije/dyq191.

[68] Stirna, J., Persson, A., 2009. Anti-patterns as a means of focusing on critical quality aspects in enterprise modeling, in: Enterprise, Business-Process and Information Systems Modeling. Springer, pp. 407–418.

[69] Tamburri, D.A., Kazman, R., Fahimi, H., 2016. The architect's role in community shepherding. IEEE Software 33, 70–79.

[70] Tamburri, D.A., Kruchten, P., Lago, P., Van Vliet, H., 2015a. Social debt in software engineering: insights from industry. Journal of Internet Services and Applications 6, 10.

[71] Tamburri, D.A., Kruchten, P., Lago, P., van Vliet, H., 2013a. What is social debt in software engineering?, in: Cooperative and Human Aspects of Software Engineering (CHASE), 2013 6th International Workshop on, pp. 93–96. doi:10.1109/CHASE.2013.6614739.

[72] Tamburri, D.A., Kruchten, P., Lago, P., van Vliet, H., 2015b. Social debt in software engineering: insights from industry. J. Internet Services and Applications 6, 10:1–10:17. URL: http://dblp.uni-trier.de/db/journals/jisa/jisa6.html#TamburriKLV15.

[73] Tamburri, D.A., Kruchten, P., Lago, P., van Vliet, H., 2015c. Social debt in software engineering: insights from industry. J. Internet Services and Applications 6, 10:1–10:17. URL: http://dx.doi.org/10.1186/s13174-015-0024-6, doi:10.1186/s13174-015-0024-6.

[74] Tamburri, D.A., Lago, P., van Vliet, H., 2013b. Organizational social structures for software engineering. ACM Comput. Surv. 46, 3.

[75] Tamburri, D.A., Nitto, E.D., 2015. When software architecture leads to social debt, in: 2015 12th Working IEEE/IFIP Conference on Software Architecture, ACM Press, Piscataway (NY), US. pp. 61–64. doi:10.1109/WICSA.2015.16.

[76] Tamburri, D.A., Palomba, F., Serebrenik, A., Zaidman, A., 2018. Discovering community patterns in open-source: a systematic approach and its evaluation. Empirical Software Engineering 24, 1369–1417.

[77] Tamburri, D.A., Palomba, F., Serebrenik, A., Zaidman, A., 2019a. Discovering community patterns in open-source: A systematic approach and its evaluation. Empirical Software Engineering 24, 1369–1417.

[78] Tamburri, D.A.A., Palomba, F., Kazman, R., 2019b. Exploring community smells in open-source: An automated approach. IEEE Transactions on Software Engineering .

[79] Tourani, P., Adams, B., Serebrenik, A., 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.. pp. 24–33. doi:10.1109/SANER.2017.7884606.

[80] Tseitlin, A., 2013. The antifragile organization. Commun. ACM 56, 40–44.

[81] Welker, K., 2001. Software maintainability index revisited. J. Def. Softw. Eng none.

[82] Zich, J., Kohayakawa, Y., Rödl, V., Sunderam, V., 2008. Jumpnet: Improving connectivity and robustness in unstructured p2p networks by randomness. Internet Mathematics 5, 227–250. URL: http://dblp.uni-trier.de/db/journals/im/im5.html#ZichKRS08.

[83] Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S., 2005. Mining version histories to guide software changes. IEEE Transactions on Software Engineering 31, 429–445.