Predicting the Emergence of Community Smells using Socio-Technical Metrics: a Machine-Learning Approach

Fabio Palomba,[1] Damian Andrew Tamburri[2]

[1]*SeSa Lab - University of Salerno, Italy*
[2]*JADE Lab - Eindhoven University of Technology /Jheronimus Academy of Data Science, The Netherlands*
fpalomba@unisa.it, d.a.tamburri@tue.nl

Abstract

Community smells represent sub-optimal conditions appearing within software development communities (*e.g.*, non-communicating sub-teams, deviant contributors, etc.) that may lead to the emergence of social debt and increase the overall project's cost. Previous work has studied these smells under different perspectives, investigating their nature, diffuseness, and impact on technical aspects of source code. Furthermore, it has been shown that some socio-technical metrics like, for instance, the well-known socio-technical congruence, can potentially be employed to foresee their appearance. Yet, there is still a lack of knowledge of the actual predictive power of such socio-technical metrics. In this paper, we aim at tackling this problem by empirically investigating (i) the potential value of socio-technical metrics as predictors of community smells and (ii) what is the performance of within- and cross-project community smell prediction models based on socio-technical metrics. To this aim, we exploit a dataset composed of 60 open-source projects and consider four community smells such as Organizational Silo, Black Cloud, Lone Wolf, and Bottleneck. The key results of our work report that a within-project solution can reach F-Measure and AUC-ROC of 77% and 78%, respectively, while cross-project models still require improvements, being however able to reach an F-Measure of 62% and overcome a random baseline. Among the metrics investigated, socio-technical congruence, communicability, and turnover-related metrics are the most powerful predictors of the emergence of community smells.

Keywords: Community Smells, Social Debt, Empirical Software Engineering;

1. Introduction

Software engineering is de-facto a social activity [1, 2] which involves often thousands if not more stakeholders arranged globally, and often separated by physical and cultural distance (*i.e.*, different cultures and culture clashes [3, 4, 5]), expertise or power distance [6, 7, 8, 9], and more). Literature calls the sum of the negative and unforeseen costs and consequences of such conditions *community smells*, namely sub-optimal organizational and socio-technical situations that hamper or altogether impede the straightforward production, operation, and evolution of software [10, 11, 12]. Previous work established the interaction between community smells and their technical counterparts (*i.e.*, code smells such as *Spaghetti Code*, *God Class* and more [13, 14, 15]) as well as, more generally, technical debt in its various forms [16, 17]. At the same time, community smells have been connected to all sorts of nasty phenomena, e.g., employee turnover [12], bad architecture decisions [11], and more, which often go beyond causing technical issues in software code but may well cause organizational turmoil or even decline [18] of software projects' organizational stability, if not project failure. Therefore, predicting community smells before they manifest along software projects' timelines may prove critical in anticipating sub-optimal organisational and socio-technical conditions before they become unmanageable. In summary, on the one hand, these conclusions from the state of the art indicate that community smells represent a first-class phenomenon to be considered when managing large-scale software systems from an organizational and socio-technical perspective.

On the other hand, previous experience reports and empirical evidence from the state of the art [10, 11, 12] also remarked that predicting community smells at large and anticipating their existence is challenging, because of their socio-technical and evolutionary nature tied to the types and characteristics of the organisational-social structure around [19, 20, 21]. Yet, the early prediction of their emergence is highly desirable for both developers and project managers in order to take informed decisions and possibly re-organize the community structure to avoid them [22, 21, 23, 24]. In our previous work [22], we investigated various aspects related to community smells, including their correlation with well-known socio-technical metrics (*e.g.*, socio-technical congruence or turnover). Our results showed the existence of a number of correlations that lead us to believe that socio-technical metrics could be potentially useful for the prediction of community smells.

Hence, in this paper we aim at empirically exploring the actual predictive power of socio-technical metrics as well as the extent to which their adoption within machine learning solutions can lead to accurate prediction of community smells.

To address our research goals, we conduct a large-scale empirical study involving 60 open-source software communities and measure how well a set of 40 socio-technical metrics can predict 4 community smells that have been shown to create harmful forms of both social and technical debt. First, we employed an information gain measure [25] to estimate the contribution given by each metric to the prediction and, in the second place, we built within- and cross-project community smell prediction models that exploit the considered socio-technical metrics. Our assumption when testing cross-project solutions is that similar working conditions in other *smelly* projects could be used to anticipate community smells. In so doing, we face a number of key aspects of machine learning modeling like, for instance, data normalization and balancing. Also, we verify the performance of five different machine learning algorithms for the task of community smell prediction.

The key findings of our study report that: (1) Socio-technical congruence, communicability, and turnover-related metrics are top-factors to consider when predicting the emergence of community problems; (2) Within-project models can effectively be employed for predicting community smells—in particular when Random Forest is used as classifier—since they reach a median F-Measure of 77%; and (3) A cross-project approach leads to a median F-Measure of 62%, being therefore a promising alternative despite some further improvements would be required.

To sum up, this paper offers three main contributions:

1. The first, large-scale empirical exploration of predictive models for community smells—this exploration is entirely novel in the state of the art in software engineering research;

2. The analysis of within- vs. cross-project community smells prediction, that provides insights into the different ways practitioners can exploit machine learning for early detection of community-related issues and whether working conditions ot other *smelly* projects could be exported and used to anticipate community smells on other projects;

3. A benchmark dataset for researchers to proceed along this line of research and compare novel mechanisms with the devised models;

Structure of the paper. The rest of this paper is structured as follows. Section 2 outlines the related work. Section 3 outlines our research design while Sections 4 to 5 showcase our results. In Section 6 we discuss possible threats to validity of our results and explain how we mitigated them. Finally, we conclude the paper and report our future research agenda in Section 7.

2. Related Work

In this section, we discuss the literature related to community smells as well as the research conducted on other socio-technical aspects.

2.1. Community smells and related research

The work we outlined in the previous pages addresses the application of machine-learning techniques to the prediction of community smells, that is, nasty organisational and socio-technical phenomena. Related work mostly resides in the domains of Machine-Learning and Big Data analytics for social-networks motif mining [26, 27, 28] as well as organization networks' analysis [29, 30]. For example, Trucco et al. [29, 31] use Bayesian models to elicit organization factors for risk engineering; however, the case-study presented by Trucco et al. is well beyond the state of the art in software engineering and provides reasonably large organization networks but reflecting mostly the domain of Supply-Chain Management. Conversely, previously in software engineering research, Russo [27] has used learning techniques and motif mining for the purpose of profiling system call changes as part of software evolution. This latter work is closer to our own in terms of focusing on software artifacts and predicting anti-patterns in their key characteristics; however, Russo focuses purely on a specific technical aspect of software evolution, rather than the organization structure around it and the anti-patterns that might emerge as its operations unfold. From another perspective, but still focusing on the use of motif analysis for anti-patterns mining in organization networks, Argentieri et al. [32] provide an overview of search-based techniques dedicated to cross-motif analysis and prediction; specifically, the authors look at combinations and mutual relations between motifs.

In comparison to the aforementioned works, we do not take the motif analysis angle at all; rather, we focus on state-of-the-art and practitioner-friendly metrics as predictors of community smells presence and severity. Our intent is to provide pre-trained models for practitioners to use in their own projects, e.g., as part of retrospectives, risk-analysis, decision-making, and more.

Beyond the above, from a rule-based perspective, community smells have seen intense research around the tool CodeFace, originally presented by Joblin et al. [33]. More specifically, the aforementioned tool was augment and experimented with for the detection [12] and evaluation of the impact of community smells

over code smells [10]. More specifically, Palomba et al. discover that community smells are the top factor when it comes to predicting specific code smells; these works provide fundamental motivations for our work as well as groundwork to establish the ground-truth behind our baseline dataset. Similar works on community smells have concentrated on establishing their impacts on other dimensions of software engineering (e.g., Architecture Debt [16], organization structure types [21], and more). With the contributions in this work we aim at providing a basic predictive mechanism for, on the one hand, practitioners to embed in their own DevOps pipelines, hopefully using the produced insights to avoid needless waste and, on the other hand, academics willing to improve beyond our results and current accuracy levels.

2.2. Research on other socio-technical aspects

The socio-technical nature of software engineering has been explored for years and from several perspectives. The seminal work by Nagappan et al. [34] showed in practice the influence of organizational structure and other "human" aspects over software quality. This and similar works (e.g., Repenning et al. [35] or Viana et al. [36]) bring evidence that motivates our study of social communities in organizations and the debt connected to them. On the one hand, while Nagappan et al. established the correlation between organizational structures and software quality, we aim to predict patterns of sub-optimality across said structures, e.g., to allow for preventive action by means of social networks analysis [37] or predictive organizational rewiring. Finally, seminal studies on on socio-technical congruence, first defined by Cataldo et al. [38] can support the predictive modeling and analytics around community smells. On one hand, socio-technical congruence is the degree to which technical and social dependencies match, when coordination is needed. On the other hand, STC may require further elaboration as well as different degrees of granularity for it to play a key role in anticipating and correcting sub-optimal organisational behavior. From a more recent perspective, the social aspects playing a key role in either code community hosting platforms (e.g., GitHub) or closed-source software engineering have been investigated from a social coding perspective (e.g., see Trockman et al. [39]) as well as from the developer attraction and retention perspective (e.g., see Qiu et al. [40]). These and other recent theories around social software engineering factors and relations are fundamental to refine predictive models such as the ones proposed in this paper.

3. Empirical Study Setup

The *goal* of the study is to investigate if and to what extent the emergence of community smells can be predicted by socio-technical metrics, with the *purpose* of providing practitioners with an automated mechanism able to promptly pinpoint the possible presence of issues within the organization community of software development teams. The *perspective* is of both researchers and practitioners: the former are interested in understanding how much community smells can be predicted by lightweight socio-technical indicators, while the latter aim at finding methods to automatically identify and possibly avoid the emergence of community-related problems.

3.1. Research Questions and Methodological Overview

Our study is driven by three main research questions that aim at exploring the problem of community smell prediction under three different perspectives. First, in our previous work [22] we studied the *correlation* between a number of socio-technical metrics and the amount of community smells in open-source projects, finding that these metrics can foreseen community-related problems. As such, our first research question aims at understanding more closely the *predictive power* of socio-technical metrics for community smell prediction. Thus, we ask:

> **RQ$_1$.** *What is the predictive power of socio-technical metrics when it comes to community smell prediction?*

In other words, **RQ$_1$** has a preliminary/descriptive nature: indeed, while in our previous study [22] we only observed correlations, the first research question has the goal of taking a closer look into the potential of socio-technical metrics for the prediction of the emergence of community smells. In this sense, the idea is to have a preliminary investigation aimed at characterizing the amount of information provided by each metric before employing them in machine learning models able to predict the emergence of community smells in general and individual smells in particular. Furthermore, **RQ$_1$** serves as a way to corroborate the correlation findings we have previously discovered.

Once assessed which metrics can be actually used as predictors of community smells, we then proceed with the definition of a supervised learning technique. As part of the second research question, we train the devised model using data coming from previous history of individual projects, *i.e.*, using a within-project strategy, with the aim of assessing how much information directly coming from the considered projects can be used for prediction of future community-related issues. Thus, we formulate the following **RQ**:

> **RQ$_2$.** *What is the performance of a community smell prediction model built using socio-technical metrics when trained using within-project data?*

As a final step, we evaluate whether and to what extent can we exploit socio-technical information of external

projects to train a community smell prediction model, *i.e.*, we assess the performance of a cross-project approach. In this case, our goal is to measure how effectively can new projects that lack of community smell and socio-technical data use external information to find potential problems within their development communities. Hence, we ask:

> **RQ$_3$.** *What is the performance of a community smell prediction model built using socio-technical metrics when trained using cross-project data?*

In short, we address **RQ$_1$** by quantifying the relevance of each socio-technical metric for the prediction of community smells; in this respect, we employ an *Information Gain* measure [25], which is a statistical technique that can *estimate* the gain provided by each independent variable to the prediction of the dependent one, giving a ranking of the features based on their importance. In the context of **RQ$_2$** and **RQ$_3$** we follow well-established guidelines [41, 42] to devise within- and cross-project models, respectively. As such, we consider and address common modeling problems such as (1) data normalization, (2) data balancing, and (3) testing of different machine learning algorithms, properly configured with respect to their hyper-parameters. Finally, we interpret the results achieved by considering various evaluation metrics (*e.g.*, F-Measure or AUC-ROC [43]) that can provide a broad vision of the strengths and weaknesses of using supervised learning for community smell prediction. Sections 4, 5 present more details on the methodology adopted and discuss the results.

3.2. Context of the Study and Dataset Preparation

The *context* of the study consists of a publicly available dataset that we built in our previous study [22] and publicly available on GITHUB.[1]. It contains **labeled data** related to four types of community smells identified over 60 open-source software communities, whose basic data are reported in Appendix A. More specifically, all systems are hosted on GITHUB[2] and their selection was originally driven by two main factors: first, we focused on communities having at least 30 contributors and 1,000 commits—this was needed to actually observe community smells, which are problems usually occurring in large projects [44, 45, 20] ; similarly, the rationale for the selection of the 30/1000 rule reflects previous work in organisations and social networks research [46] which points out that SNs which reflect a total of 100 nodes exhibit the organisational and community structure behavior of so-called non small-world topology networks and therefore are considered "large". Considering that in software organizational structures every single developer commits to and caters for about 2-3 software artefacts across the Developer Social Networks in our dataset, we fixed the total minimum of 30 contributors to characterise "large" software projects. Second, we only considered systems for which at least ten commits were performed during the last observed month, with the aim of mitigating threats due to outdated phenomena. As such, starting from all projects respecting these two conditions, we randomly picked 60 of them.

Once selected the objects, we identified and validated community smells. To this aim, we first restricted our focus to four community smells that have been shown by previous research [45, 47] to be (1) among the most problematic community-related issues to deal with and (2) a potential threat to the emergence of technical debt.[3] Specifically, these are:

1. ORGANISATIONAL SILO EFFECT: This refers to the presence of siloed areas of the developer community that do not communicate, except through one or two of their respective members;
2. BLACK CLOUD EFFECT: This reflects an information overload due to lack of structured communications or cooperation governance;
3. LONE WOLF EFFECT: This smell appears in cases where the development community presents unsanctioned or defiant contributors who carry out their work with little consideration of their peers, their decisions and communication;
4. BOTTLENECK OR "RADIO-SILENCE" EFFECT: This is an instance of the "unique boundary spanner" [48] problem from social-networks analysis: one member interposes herself into every formal interaction across two or more sub-communities with little or no flexibility to introduce other parallel channels.

To identify them, we relied on our tool CODEFACE4SMELLS [22], a fork of CODEFACE [49] which was originally designed to extract coordination and communication graphs mapping the developer's relations within a community. Our tool builds on top of CODEFACE and applies a rule-based approach to identify instances of the four community smells described above. For 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.

A basic example is given in Figure 1. In this example two developers, "1" and "2", are collaborating on some

[1] Dataset: https://github.com/maelstromdat/codeface4smells_TR
[2] Link: http://github.com/

[3] Please note that, even though other community smells have been observed in literature (*e.g.*, [45]), there is still no way to identify them. Therefore, we cannot consider a larger set of community smells in our study.

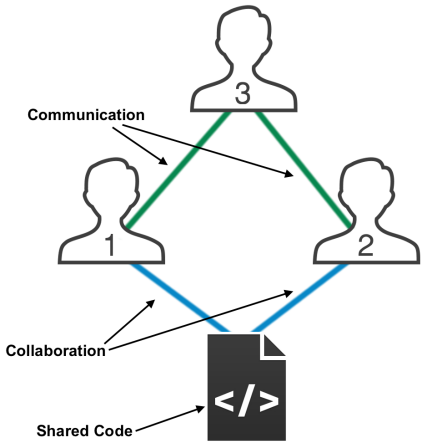

Figure 1: Lone Wolf Community Smell identification pattern - image taken from [22].

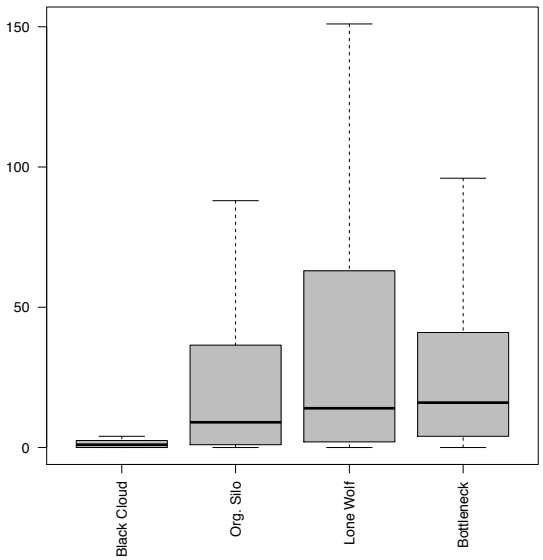

Figure 2: Distribution of community smells in our dataset - image taken from [22]

code, but they are not connected by any communication link other than developer "3", who is not co-committing on a shared file. In this case, either developer "1" or developer "2" (or both) can develop a Lone Wolf community smell. The full list of identification rules as well as more examples are available in our online appendix [50].

It is important to note that the dataset contains community smells *for each release* of the considered projects, meaning that they have been detected over their entire life span—this is a key detail for this study, as it enables the execution of a release-by-release strategy when training the devised within-project model (as described in Section 5). The distribution of the community smells per release is shown in Figure 2: as visible, for three of them, *i.e.*, Organizational Silo, Lone Wolf, and Bottleneck, the median number of occurrences for each release is around 10, while Black Cloud is much lower (2). This aspect clearly affects the way a machine learning algorithm aiming at predicting them should be setup. Last, but not least, it is important to comment on the reliability of this dataset when used as ground-truth by a machine learning solution. Unlike other types of software engineering information that can be more easily verified (*e.g.*, defects or design flaws [51, 52, 53, 54]), it is not possible to create a "tangible" oracle of community smells since (1) they involve teams and (2) are not tracked by organizations and *cannot be* without an automated solution [45]. As such, we are not able to compute common measures like precision and recall when evaluating the tool. Alternatively, to evaluate the identification rules adopted by CodeFace4Smells, in our previous works [22, 45], we conducted surveys and/or semi-structured interviews with both the original industrial and open-source practitioners belonging to the communities we considered, showing them the results of the tool and asking for confirmation. More specifically, we first mined the community smells appearing in the 60 software projects also used in the context of this paper as well as the email addresses of the most active developers of those communities (*i.e.*, the ones who committed at least 10 changes during the year before the release dates taken into account), ending up with 172 developers who were willing to participate in the study. Secondly, we inquired the developers about the existence and perception of the community smells considered. Finally, we performed a follow-up confirmatory study involving 35 developers (of those who participated already to the survey), in which we openly discussed about community smells and the performance of the tool. As an outcome, we discovered that the problem of community smells is highly recognized in practice, with them being considered as the main source of social debt. Furthermore, all the survey respondents and interviewees reported the validity and usefulness of CodeFace4Smells, 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*. This makes us confident of the high reliability of the tool.

Once identified the smelly community structures in our dataset, the final step required to conduct our study consists of the identification of the *non-smelly* ones, namely those community structures that are not affected by smells and that, therefore, can be used to complement the smelly ones and train a machine learner. For each release of the 60 projects, we consider as non-smelly all those parts of the developer's social network that are not detected as smelly by CodeFace4Smells.

4. RQ$_1$. On the Predictive Power of Socio-Technical Metrics

This section describes methodology and results that address our first research question.

4.1. Research Methodology

The first step of our empirical study consists of understanding the predictive power of socio-technical metrics when it comes to the prediction of community smells. Such an analysis derives from the results of our previous study [22], where we discovered a number of correlations between socio-technical metrics and community smells: for this reason, in the context of this paper we take into account exactly the same set of metrics previously analyzed, which are briefly reported in Table 1. This set consists of 40 different metrics belonging to five main categories that capture community-related aspects under different perspectives, *e.g.*, the turnover of developers rather than their overall closeness. The rationale behind the selection of those metrics is summarized in Section 2. Although the aforementioned rationales reflect a considerable amount of metrics, there exist as many as 350+ factors that may play a key role in predicting and managing sub-optimal organizational conditions, as well as software project success and failure as we ourselves highlighted in previous research [56]. With this first work, we aim at laying foundations (*i.e.*, by conducting a first analysis and providing a dataset) for future work in the area.

As anticipated in Section 3, we then address **RQ$_1$** by running an *information gain* measure [25] able to quantify the gain provided by each feature to the prediction. More formally, let M be a supervised learning model, let $S = \{s_1, \ldots, s_n\}$ be the set of socio-technical metrics composing M, an *information gain* measure [25] applies the following formula to compute a measure which defines the difference in entropy from before to after the set M is split on an attribute s_i:

$$InfoGain(M, s_i) = H(M) - H(M|s_i) \quad (1)$$

where the function $H(M)$ indicates the entropy of the model that includes the predictor s_i, while the function $H(M|s_i)$ measures the entropy of the model that does not include p_i. As for entropy, this is computed using the Shannon's entropy [67], which computes the probability that the predictor s_i affects the dependent variable's possible outcomes. It is computed as follow:

$$H(M) = -\sum_{i=1}^{n} prob(s_i) \log_2 prob(s_i) \quad (2)$$

In simpler terms, the algorithm quantifies how much uncertainty in M is reduced after splitting M on predictor s_i. In our work, we implement information gain through the *Gain Ratio Feature Evaluation* measure [25] available in the WEKA toolkit [68], in combination with the `weka.attributeSelection.Ranker` search method. This ranks $s_1, \ldots, s_n$ in descending order based on the contribution provided by s_i to the decisions made by M. More specifically, the output of the algorithm is represented by a ranked list in which the features having the higher expected reduction in entropy are placed at the top. With this procedure, we can evaluate the relevance of each socio-technical metric in the prediction model. Along with this analysis, we also assess whether a certain feature mainly contributes to the prediction of smelly or non-smelly social structures: this is done through the `evaluateAttribute` function of the WEKA implementation of the algorithm. We run our analyses considering each project independently, thus obtaining 60 different ranks. We prefer this option since it allows us to both assess if certain metrics are more powerful on specific projects and evaluate the overall relevance of each metric. To analyze the resulting ranks and have statistically significant conclusions, we finally exploit the Scott-Knott Effect Size Difference (ESD) test [69]. This represents an effect-size aware variant of the original Scott-Knott test [70] that has been recently recommended for software engineering research [71, 72, 73] because it (i) uses hierarchical cluster analysis to partition the set of treatment means into statistically distinct groups according to their influence, (ii) corrects the non-normal distribution of an input dataset, and (iii) merges any two statistically distinct groups that have a negligible effect size into one group to avoid the generation of trivial groups. To measure the effect size, the tests uses the Cliff's Delta (or d) [74]. We employ the publicly available `ScottKnottESD` implementation[4] originally developed by Tantithamthavorn *et al.* [69]. Finally, it should be noted that all metrics we are using as predictors in our machine learning exercise are independent from the metrics used for the detection of the golden-set we are using in our dataset and, as such, there is no risk of over-fitting or misleading results given by the inter-dependence between dependent and independent variables.

4.2. Analysis of the Results

An overview of the results for our first research question is presented in Table 3. As shown, all the considered socio-technical indicators have some predictive power and, indeed, the mean information gain is of at least 0.10. Interestingly, the standard deviation indicates that the results do not consistently vary from project to project, meaning that the metrics represent valid indicators independently from the specificities of a certain software development community.

Among them, `socio-technical congruence` and `communicability` are the ones providing the highest information gain for community smell prediction. This

[4]Link: https://github.com/klainfo/ScottKnottESD

Table 1: Socio-technical metrics considered in our study. All of them have been shown to be correlated to community smells in our previous work [22]. Note that the "Devs" or developers are defined as the total sum of members in the developer social network whereas the "core (either global, code, or ml) developers" are defined as the total sum of developers which are part of the core of the organizational structure in line with Wallerstein core-periphery network structure theory, which Joblin *et al.* [55] have shown to be true for Software Communities as well; conversely, non-core developers reflect the total sum of every other developer not belonging to the structural core.

Category	Metric	Description
Developer Social Network metrics	devs	Number of developers present in the global Developers Social Network
	ml.only.devs	Number of developers present only in the communication Developers Social Network
	code.only.devs	Number of developers present only in the collaboration Developers Social Network
	ml.code.devs	Number of developers present both in the collaboration and in the communication DSNs
	perc.ml.only.devs	Percentage of developers present only in the communication Developers Social Network
	perc.code.only.devs	Percentage of developers present only in the collaboration Developers Social Network
	perc.ml.code.devs	Percentage of developers present both in the collaboration and in the communication DSNs
	sponsored.devs	Number of sponsored developers (95% of their commits are done in working hours)
	ratio.sponsored	Ratio of sponsored developers with respect to developers present in the collaboration DSN
Socio-Technical Metrics	st.congruence	Estimation of socio-technical congruence
	communicability	Estimation of information communicability (decisions diffusion)
	num.tz	Number of timezones involved in the software development
	ratio.smelly.devs	Ratio of developers involved in at least one Community Smell
Core community members metrics	core.global.devs	Number of core developers of the global Developers Social Network
	core.mail.devs	Number of core developers of the communication Developers Social Network
	core.code.devs	Number of core developers of the collaboration Developers Social Network
	sponsored.core.devs	Number of core sponsored developers
	ratio.sponsored.core	Ratio of core sponsored developers with respect to core developers of the collaboration DSN
	global.truck	Ratio of non-core developers of the global Developers Social Network
	mail.truck	Ratio of non-core developers of the communication Developers Social Network
	code.truck	Ratio of non-core developers of the collaboration Developers Social Network
	mail.only.core.devs	Number of core developers present only in the communication DSN
	code.only.core.devs	Number of core developers present only in the collaboration DSN
	ml.code.core.devs	Number of core developers present both in the communication and in the collaboration DSNs
	ratio.mail.only.core	Ratio of core developers present only in the communication DSN
	ratio.code.only.core	Ratio of core developers present only in the collaboration DSN
	ratio.ml.code.core	Ratio of core developers present both in the communication and in the collaboration DSNs
Turnover	global.turnover	Global developers turnover with respect to the previous temporal window
	code.turnover	Collaboration developers turnover with respect to the previous temporal window
	core.global.turnover	Core global developers turnover with respect to the previous temporal window
	core.mail.turnover	Core communication developers turnover with respect to the previous temporal window
	core.code.turnover	Core collaboration developers turnover with respect to the previous temporal window
	ratio.smelly.quitters	Ratio of developers previously involved in any Community Smell that left the community
Social Network Analysis metrics	closeness.centr	SNA degree metric of the global DSN computed using closeness
	betweenness.centr	SNA degree metric of the global DSN computed using betweenness
	degree.centr	SNA degree metric of the global DSN computed using degree
	global.mod	SNA modularity metric of the global DSN
	mail.mod	SNA modularity metric of the communication Developers Social Network
	code.mod	SNA modularity metric of the collaboration Developers Social Network
	density	SNA density metric of the global Developers Social Network

was somehow expected since these metrics reflect the coordination and communication level existing in an organization, thus covering the two key aspects of community smells. From a more technical viewpoint, the results obtained for these two metrics can be interpreted as follow. Since the information gain measures the extent to which a certain feature will impact the purity of the dataset, namely how much the feature will help discriminating the dependent variable classes of the model, we can claim that `socio-technical congruence` and `communicability` are the metrics that maximize the differences between smelly and non-smelly community structures, meaning that these are the characteristics that most impact the fact that a community structure will be smelly—these will likely be the first two features that a machine learning model will consider when performing predictions. It is worth noting that both the metrics mainly support the prediction of smelly communities, thus confirming that they seem to be good predictors for community smells. The third relevant factor is represented by the core global developers turnover with respect to the previous temporal window (`core.global.turnover`): its high relevance may indicate that the emergence of community-related problems can be accelerated/reduced when there is a high/low turnover of core developers. This is in line with findings previously shown in organizational and social science research [75, 76, 77], and suggests that software communities can be, to some extent, subject of similar dynamics as other types of communities. The high relevance of turnover is also visible when considering the other metrics aiming at capturing this aspect: for example, `core.mail.turnover`, `core.code.turnover`, and `ration.smelly.quitters` have high information gain values (0.47, 0.37, and 0.37, respectively). Hence, on the one hand our results indicate that turnover metrics may potentially provide a notable amount of information

Table 2: Metrics selection rationale; metrics and rationales follow the same clustering schema from Tab. 1.

Metric	Rationale
Developer Social Network Metrics	These metrics are largely derived from the works of Meenly et al. [37, 57] as well as Joblin et al. [33] who, respectively, (1) introduced the fundamental roles of Developer Social Networks in predicting failures (both at technical and organizational level) and (2) prototyped technologies for the verifiable investigation of fully-formed developer communities. Both research groups find independently that DSN metrics are not only relevant to specify the stability and predictive characteristics of organisational structures but also that they reflect relevant technical characteristics in code which reflect socio-technical issues in the organizational structure.
Socio-Technical Metrics	These metrics are derived from the state-of-the-art in social software engineering as reflected by the topics discussed in the "Cooperative and Human Aspects of Software Engineering (CHASE)" and "Social Software Engineering (SSE)" workshop series investigating the social and organisational aspects emerging within software communities, and any relation thereto. Conversely, more mature and prominent sample research exists which highlights the relation between these metrics and sub-optimal conditions in the organisational structure, e.g., Kwan et al. [58] who investigate build failures connected to sub-optimal coordination patterns or Tamburri [11] who studies sub-optimal architectural decision patterns connected to incommunicability.
Core-Community Members Metrics	Metrics in this cluster range between the truck-factor (whose relation to sub-optimal organizational conditions is highlighted by Avelino et al. [59] as well as others [60]) to core-periphery numbering metrics, which are largely inspired by the work of Manteli et al. [61, 62] who investigate core-periphery organizational (anti-)patterns and their relation with community member descriptors as well as core-periphery mismatches.
Turnover Metrics	These metrics are derived directly from the original committers of the studies reported in [10, 11, 12] who desired to investigate their turnover rates across several distributed software development sites across the world. Such industrial requirement to focus on Turnover as a key dimension was mapped to all aforementioned core-community members metrics.
Social-Network Analysis Metrics	These metrics are derived from the state of the art in organisations' research and reflect the most relevant centrality measures recurrently used in connection to sub-optimal organisational conditions such as prima-donna effects (e.g., see Dekker et al. [63]) or lack of boundary-spanning [64, 65, 66].

bits to a machine learning model that would allow it to better discern community smelly patterns. On the other hand, we can also claim that turnover metrics have a potentially less predictive power than coordination and communication ones: in other words, our findings seem to suggest that keeping `socio-technical congruence` and `communicability` aspects under control is more important for the emergence of community smells than the fact of having developers entering/leaving the community—this is again an indication that the implementation of good governance mechanisms may increase the health of the community independently from the specific developers composing it [76, 21].

According to our findings, also some typical social-network analysis metrics, like `degree.centr` (info gain=0.53) and `density` (0.41), are relevant factors to consider, meaning that even the structure of the community can impact the emergence of community-related problems. On the other hand, metrics capturing the truck factor, i.e., the number of team members that have to suddenly leave a project before it stalls, provide a lower information gain. As an example, the `code.truck` metric have an information gain of 0.16 with a standard deviation of 0.10, which indicates that in some cases it is almost irrelevant for predicting community smells. The low amount of information provided by truck factor metrics seem to confirm, again, that the emergence of community smells can be ruled by putting in place effective governance mechanisms that stimulate the coordination/communication among developers rather than having an excessive control on factors involving individual developers that may, in this case, leave the project at a certain point in time. From a statistical point of view, the discussion above is confirmed. The ranking provided by the Scott-Knott ESD test is exactly the same as the one of the Gain Ratio Feature Evaluation, meaning that the contribution given by the considered socio-technical metrics is statistically significant.

To broaden the scope of the discussion, the results obtained from this research question tell us that community smells are a multifaceted phenomenon that is characterized by various aspects not only related to coordination and communication, but also to turnover and community structure. As such, we argue that more research on the topic should be conducted in order to provide an improved understanding of how these aspects influence the emergence of community smells.

> ***Finding 1.*** *All the considered socio-technical factors provide some information gain for the prediction of community smells. Socio-technical congruence, communicability, and turnover-related metrics are the most powerful ones, while it seems that factors aiming at capturing the truck factor are less relevant for community smell prediction.*

Table 3: Results outline for $\mathbf{RQ}_1$; column 1 identifies the analyzed metric, Column 2 and 3 provide Mean and Standard Deviation respectively while Column 4 elaborates on smelliness and finally Column 5 elaborates on results from the Scott-Knott test. The SK-ESD values reported refer to the percentage of datasets in which a certain feature appeared to be at the top rank.

Metric	Mean	St. Dev.	Class	SK-ESD
st.congruence	0.54	0.05	smelly	74
communicability	0.53	0.03	smelly	67
core.global.turnover	0.53	0.10	smelly	62
degree.centr	0.53	0.07	non-smelly	59
core.global.devs	0.48	0.03	non-smelly	39
perc.ml.only.devs	0.47	0.06	smelly	46
core.mail.turnover	0.47	0.03	non-smelly	46
devs	0.46	0.09	non-smelly	44
ml.code.devs	0.46	0.09	smelly	40
mail.truck	0.44	0.07	non-smelly	40
perc.ml.code.devs	0.43	0.06	non-smelly	38
density	0.41	0.04	smelly	35
code.only.core.devs	0.40	0.06	non-smelly	30
perc.code.only.devs	0.39	0.04	non-smelly	32
sponsored.core.devs	0.39	0.10	smelly	32
core.code.turnover	0.37	0.09	smelly	32
ratio.smelly.quitters	0.37	0.07	non-smelly	32
code.mod	0.37	0.07	non-smelly	31
ratio.mail.only.core	0.36	0.08	non-smelly	29
closeness.centr	0.36	0.06	smelly	27
ratio.sponsored.core	0.34	0.10	non-smelly	24
global.turnover	0.34	0.06	smelly	16
mail.mod	0.34	0.09	non-smelly	9
code.turnover	0.32	0.05	non-smelly	8
sponsored.devs	0.31	0.10	non-smelly	1
ratio.code.only.core	0.31	0.10	non-smelly	1
ratio.ml.code.core	0.30	0.07	non-smelly	1
global.mod	0.30	0.04	non-smelly	1
global.truck	0.26	0.03	smelly	1
betweenness.centr	0.24	0.03	non-smelly	1
core.code.devs	0.22	0.06	non-smelly	1
ratio.smelly.devs	0.22	0.04	smelly	1
ml.only.devs	0.20	0.06	non-smelly	1
code.only.devs	0.20	0.08	smelly	1
ratio.sponsored	0.17	0.06	non-smelly	1
mail.only.core.devs	0.17	0.07	non-smelly	1
code.truck	0.16	0.10	smelly	1
ml.code.core.devs	0.14	0.04	non-smelly	1
num.tz	0.13	0.09	non-smelly	1
core.mail.devs	0.10	0.09	smelly	1

5. $\mathbf{RQ}_2$ - $\mathbf{RQ}_3$. Assessing Community Smell Prediction Models

This section overviews the methodological details and the results for $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$.

5.1. Research Methodology

To address both research questions, we need to devise community smell prediction models. This requires the design and definition of a number of steps, which we describe in the following.

Dependent Variable. In our work, we have two types of response variables. In the first case, we consider a binary variable in the set {`true`, `false`}, which represents the presence/absence of a community smell. Thus, we first assess how well can a machine learning model predict the simple presence of a community smell, independently from its specific type. In the second case, we consider the problem of classifying the precise smell affecting a software community: as such, the response variable is a nominal value that can assume values in {`organizational-silo`, `black-cloud`, `lone-wolf`, `bottleneck`, `none`}, where `none` indicates non-smelly social structures.

Independent Variables. The features used to predict community smells are the socio-technical metrics previously presented in Table 1. However, in this context we take into account the problem of multi-collinearity [78], which appears when two or more variables of the model are highly correlated to each other, possibly leading the machine learner not to distinguish which of them should consider when predicting the dependent variable. We compute the Spearman's rank correlation [79] between all possible pairs of metrics to determine whether there are pairs strongly correlated (*i.e.*, with a Spearman's $\alpha > 0.8$). If two independent variables are highly correlated, we exclude the one having the least predictive power, as measured in $\mathbf{RQ}_1$ through the *Gain Ratio Feature Evaluation* measure [25].

Machine-learning Algorithm. To the best of our knowledge, this is the first work that aims at predicting the emergence of community smells. As such, still nothing is known on how different machine learning algorithms work in this context. For this reason, we experiment with five different classifiers, namely Random Forest, J48, Logistic Regression, Decision Table, and Naive Bayes. These algorithms make different assumptions on the underlying data as well as have different advantages and drawbacks in terms of execution speed and overfitting [80], thus giving us the possibility to explore deeper the problem of community smell prediction. It is important to point out that before running them, we configure their hyper-parameters as recommended in literature [81]; for this task, we exploit the Grid Search algorithm [82].

Training Strategy. In the context of $\mathbf{RQ}_2$ we aim at building and evaluating a within-project model. To this aim, we employ a release-by-release training approach, where the data of a release R_i is used to train a model that can predict the emergence of community smells on the release R_{i+1}. The process is repeated for each pair of releases of the considered projects until the end of the observed history. Of course, we have to exclude first and last release of each project from the validation: the former has no previous data to use as training, while the latter has no future data to use as testing. We opt for this time-sensitive strategy since our data follows a *temporal distribution*: this implies that

other widely-used validation approaches (*e.g.*, 10-fold cross [83]) do not represent valid alternatives, as they would potentially train the model with data coming from releases that are subsequent with respect to the data in the test set—thus leading to unrealistic and biased results [84]. Furthermore, a release-by-release strategy can actually simulate a real-case scenario where a prediction model is updated as soon as new information is available.

When training the experimented machine learning techniques, we take into account the data imbalance problem [85], which occurs when the number of data points available in the training set for a certain class (*e.g.*, the number of smelly instances) is far less than the amount of data points available for another class (*e.g.*, the number of non-smelly instances), possibly reducing the ability of machine learning algorithms to learn how to classify the minority class. This problem eventually occurs in our case, as, on average, only 12% of community structures of the considered releases are smelly. Hence, we apply the *Synthetic Minority Over-sampling Technique* (SMOTE) [85]. We use the standard implementation of the algorithm available in Weka: as such, the parameter K, which refers to the number of nearest neighbors that the algorithm should use while oversampling the training data, is equal to 5. When predicting the presence/absence of community smells, the algorithm creates synthetic instances of any kind of community smells. Instead, when predicting the smell type, it is repeated multiple times so that it can over-sample the instances of each community smell. More precisely, in the first case the algorithm has been run once per release—we used a release-by-release strategy, hence the training set should have been balanced at each release. In the second case we run it once per smell per release, *i.e.*, we ensured that the same proportion of a certain community smell was available in all the releases considered. In both cases, the algorithm gives us a balanced training set, where all classes have a similar amount of instances. For the sake of completeness, it is important to remark that we iteratively apply SMOTE on each release R_i used as training set, *leaving the test set intact*, so respecting a real case scenario where the number of smelly and non-smelly structures is not balanced.

Turning the focus on $\mathbf{RQ}_3$, in this case we are interested in assessing a cross-project model. We employ a *leave-one-out* validation strategy [84]: we use a project P_i as test set, while we train the experimented machine learning algorithms with data coming from all the remaining projects. This process is then repeated 60 times, so that we let each project be the test set once. In a cross-project scenario, there are three problems to consider. First, the distribution of the socio-technical metrics can substantially differs from project to project, meaning that a machine learner could potentially fail in learning because of the influence of outliers and/or other anomalies [86]. For this reason, we scale and normalize the computed metrics using the `normalize` function available in the Weka toolkit. Second, cross-project models can suffer from data unbalance as well: so, also in this case we iteratively apply SMOTE to balance the training data—we run it once for each training set built. Finally, cross-project models are sensitive to the *selection of the training data*. Indeed, heterogeneous information coming from very different projects with respect to the tested one could have a negative influence on the predictive ability of the machine learner. While there is some ongoing research on this topic [87, 88, 89], we could not find a mature enough and/or publicly available instrument that could allow us a proper project selection for training our learners. For this reason, it can be said that our work sets the *lower-bound for cross-project community smell prediction*. Should our results be already promising, this would imply that even better prediction performance could be obtained by practitioners if a more careful selection of the training data would be performed.

Evaluation Metrics. We assess the goodness of the experimented models by computing well-known metrics such as precision, recall, F-Measure, AUC-ROC, and Matthew's Correlation Coefficient (MCC) [90]. These metrics allow us to study the performance of both within- and cross-project community smell prediction models under different perspectives.

5.2. Analysis of the Results

As explained above, we experimented with five different machine learning algorithms such as Random Forest, J48, Logistic Regression, Decision Table, and Naive Bayes. Such an experiment revealed that Random Forest is the technique obtaining the best performance considering all the evaluation indicators and overcome J48, i.e., the second best performing technique, by up to 7% and 5% in terms of F-Measure and AUC-ROC, respectively. It is also worth remarking that the model built using Random Forest performed, overall, 11% better with respect to model built using the same classifier but using all features: this confirms that the feature selection process applied actually provided gains in terms of performance. Based on the observations above, we therefore decided to focus the discussion on the results achieved by Random Forest, while a comprehensive overview of the performance of the other classifiers is available in our online appendix [50].

Within-project community smell prediction. As a preliminary methodological step, we verified possible multi-collinearities among the features of the model. This led to the removal of five variables, namely `ratio.sponsored` (it had a correlation of 0.86 with `ratio.sponsored.core` and was removed because it

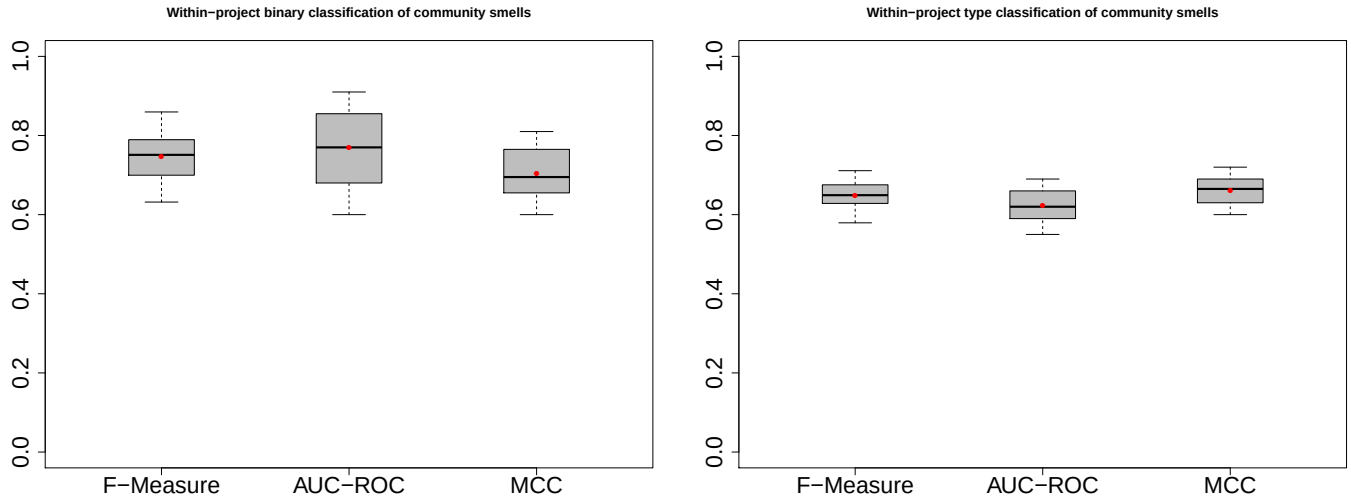

Figure 3: Box plots reporting the performance of the investigated within-project community smell prediction model in terms of F-Measure, AUC-ROC, and MCC.

provides a lower information gain), `num.tz` (because of its correlation of 0.82 with `code.only.devs`), `devs` (because of the 0.92 correlation value with `perc.ml.only.devs`), `ratio.mail.only.core` (because of the correlation of 0.87 with `ratio.smelly.quitters`), and `betweenness.centr` (correlation of 0.83 with`degree.centr`).

Figure 3 depicts box plots describing the distribution of F-Measure, AUC-ROC, and MCC when running the community smell prediction model, trained using within-project data, over the considered software projects. The figure reports both the performance of the model used to predict a binary value reporting presence/absence of community smells (on the left-hand side) and the model exploited to predict the types of community smells appearing in next project releases (on the right-hand side). Looking at the figure, we can immediately point out that, overall, the performance of the model is high when considering all the evaluation metrics. Indeed, it reaches a median F-Measure is 77% when predicting the existence of community smells, while the F-Measure is equal to 63% when trying to identify the exact type of smell that will occur. The results are consistent among the other metrics. On the one hand, these results indicate that a within-project approach allows a pretty accurate prediction of community smell emergence. On the other hand, the AUC-ROC values—which measure how well the classifiers can separate the binary classes [91]—tell us that within-project models are also robust. As such, we can answer $\mathbf{RQ}_2$ by saying that a within-project solution can properly support developers and project managers when assessing the health status of a community. An interesting observation can be done when considering the MCC values of the binary model when compared to the one aiming at predicting community smell types. As shown in Figure 3, we observe that, despite the higher average accuracy, the variability of the first is larger, meaning that there exist software projects where it is easier to predict the exact type of community smells rather than their simple presence. Differently from the other evaluation metrics, MCC is computed using the entire confusion matrix and, therefore, it takes into account the ability of a learner to classify false and true negatives. The results tell us that the binary model has sometimes more difficulties in classifying actual community smells (false negatives) and smell-free community structures (true negatives) with respect to the model that classifies community smell types. This can be explained by considering that the binary model does not aim at distinguishing the characteristics of the individual smells. As a consequence, it learns more general features that define a community smell structure: based on what we found, this sometimes biases the model, that cannot properly classify actual community smells and smell-free community structures. On the contrary, the finer-grained model is defined to learn the features characterizing single community smells, possibly helping the machine learner to use the right features when classifying true and false negatives.

Going deeper into the results, we noticed that predicting the existence of community smells in future releases of a system represents an easier task with respect to the prediction of the precise type of problem that will occur, as visible in the difference of 14% in terms of F-Measure. This was somehow expected (and reasonable) for various reasons. In the first place, a binary classification allows the machine learner to have more data points to learn and, perhaps more importantly, the task of learning the characteristics of just two classes is generally easier for machine learning algorithms [92]. Secondly, more community smell types can be determined by the same set of socio-technical metrics, thus creating

noise during the classification. This claim is supported by an additional analysis that we have done on the features used by RANDOM FOREST to predict the different types of community smells. In particular, we measured the Gini index [93]—the entropy-based metric used by the classifier to decide which features it should use for prediction—obtained by the socio-technical metrics when used to predict the different types of community smell types. As a result, we discovered that `socio-technical congruence` and `core.global.turnover` are the main metrics used for predicting both ORGANIZATIONAL SILO and BLACK CLOUD, while `communicability` has a high impact on the decisions made by the classifier when identifying LONE WOLF and BOTTLENECK. As such, the false positive rate increases, thus reducing the accuracy.

Finally, we observed that the performance of both model types does not vary over different projects; looking at the shapes of the box plots, which are all narrowed toward the median of the distributions, we can conclude that the prediction accuracy is similar independently from the underlying development community.

> ***Finding 2***. *It is possible to predict the emergence of community smells in next releases of a software project with an F-Measure of 77% (on median), while the precise type of community problem that will occur can be predicted with an F-Measure of 63%. The higher false positive rate of latter model is due to the presence of highly-relevant socio-technical features that are used for the identification of more community smell types.*

Cross-project community smell prediction. As previously done, we first controlled for multi-collinearity. In this case, with the step we removed seven variables, namely: `ratio.sponsored` (correlation of 0.93 with `ratio.sponsored.core`), `num.tz` (correlation of 0.81 with `code.only.devs`), `devs` (correlation of 0.85 with `perc.ml.only.devs`), `ratio.mail.only.core` (correlation of 0.97 with`ratio.smelly.quitters`), `betweenness.centr` (correlation of 0.86 with `degree.centr`), `perc.code.only.devs` (correlation of 0.94 with `code.only.core.devs`), and `core.mail.devs` (correlation of 0.9 with `ml.code.core.devs`).

Figure 4 reports the results of this analysis. Similarly to $\mathbf{RQ}_2$, we show the performance of cross-project models when used to predict a binary value on presence/absence of community smells (left-hand side of Figure 4) and the exact type of community problem (right-hand side of Figure 4).

The results are somehow similar to what reported for within-project models. Also in this case the binary model works better than the one classifying community smell types: the median F-Measure of the former reaches 62%, while the one of the latter is 58%. The difference is similar when considering AUC-ROC (60% vs 57%) and MCC (59% vs 58%). As clearly visible, however, the overall accuracy of cross-project models is notably lower than within-project ones. This is likely due to the heterogeneity of the data contained in the training set; this leads us to confirm previous findings in the field on the lower performance of cross-project prediction models [52, 94, 95, 96].

At the same time, the performance is still to be considered promising: even without selecting the data to use as training, the cross-project model obtains a prediction accuracy that is higher than the one of a random model. This statement is supported by an additional experiment we have performed, in which we compared the model with a simple classifier that randomly guesses the smelliness of a community—note that also in this case we applied SMOTE first to avoid having a too relaxed baseline. We found that the random model has an F-Measure close to 46%, being therefore worst than the cross-project model devised. Based on these findings, we can argue that more research on the topic could lead to potential additional benefits when it turns to the prediction of community smells using cross-project data.

When considering the prediction of community smell types, the performance drops substantially. In this case, the heterogeneity issue combined with the ability of certain socio-technical metrics to influence more community smells are at the basis of the reduced performance. Also in this case, we tried to understand how bad this performance is when compared with the one of a random model. We found that the devised cross-project model has an F-Measure 27% higher than the baseline (which reached a median F-Measure of 31%), confirming that it still represents a more viable approach for being used in practice. In conclusion, our model poses the ground for more research on effective mechanisms to make cross-project community smell prediction practical.

When comparing the results of within- and cross-project models, we can notice that the variability of the performance is larger in the first case—this is true for all considered evaluation metrics. This means that, while the performance of within-project models is generally higher, the latter seem to be more stable. From a practical standpoint, our findings tell us that a cross-project setting guarantees similar performance when employed in different contexts: this result is pretty promising in our view, since it implies that further improvement in the way the model is trained (*e.g.*, proper selection of the projects to be used as training data) could not only lead to better performance, but also to the definition of models that can be pragmatically used by developers in unseen contexts.

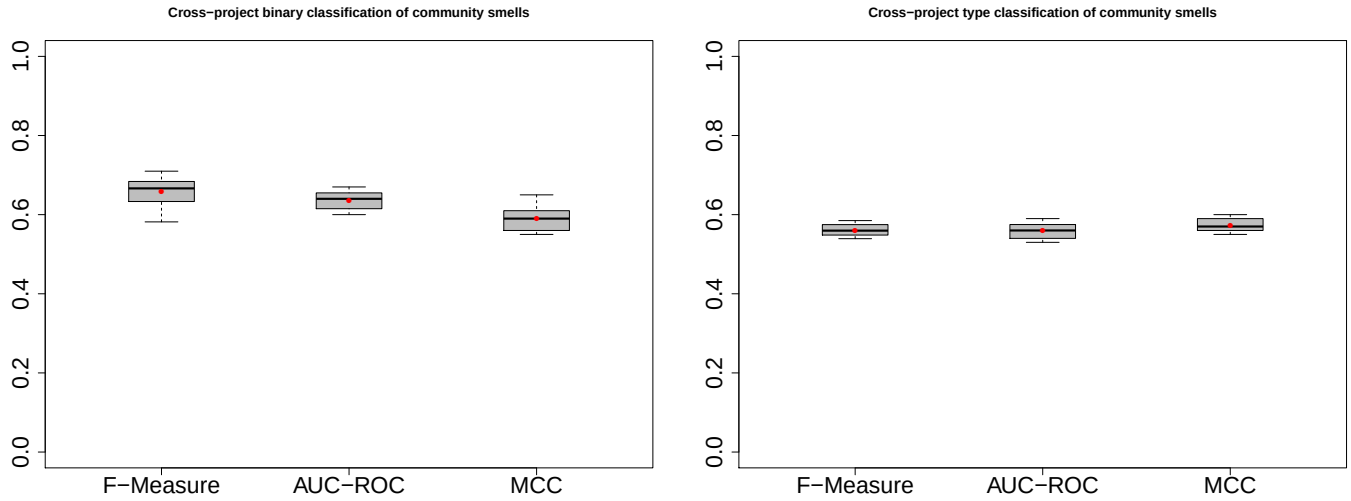

Figure 4: Box plots reporting the performance of the investigated cross-project community smell prediction models in terms of F-Measure, AUC-ROC, and MCC.

Finding 3. *The devised cross-project model for predicting community smells has a median F-Measure of 62%, AUC-ROC of 60%, and MCC of 59%. When trying to predict the type of community smells, the performance drops to 58%, 57%, and 58% for F-Measure, AUC-ROC, and MCC, respectively. A comparison with random models reveal that a cross-project solution still represent a more performing approach. Yet, the performance could be improved by reducing data heterogeneity and finding further features able to characterize community smells.*

5.3. Cross-study Findings

The results outlined in the previous section are in line with observations made in previous work [12]. More specifically, in Tamburri et al. [12], we observed that a restricted set of 18 socio-technical metrics from the same overview we adopt here, directly correlated—either positively or negatively—with community smells; beyond this contribution, the manuscript proceeded in highlighting all observable *stability thresholds* for at least some of the factors we also adopt in the scope of this work (e.g., a minimum value of 0.8 for the truck-number of the collaboration DSN was reported as a prerequisite for stability).

Among the aforementioned 18 factors, three key forces emerge, namely, *socio-technical congruence*, *communicability*, and *truck-factor* (as well as connected turnover metrics) and are reported—both in this work and in Tamburri et al. [12]— as strongly correlated with the emergence and prediction of no less than 2 community smells. Said metrics are therefore set to become first-class citizens in terms of software metrics needed to track and govern software projects appropriately (e.g., as augmentations of platforms such as *Bitergia* or *OpenHub*). On the one hand, low numbers across the board on such metrics indicates that corrective action needs to be undertaken in the communication and/or collaboration structure across the project. On the other hand, high numbers on the aforementioned metrics may themselves represent conditions whose impact on software processes remains to be fathomed.

Concerning all remaining metrics, nothing definitive can be said. While on previous study we reported all metrics being connected to some form of socio-technical condition, the observations made in this study—although confirmative of previous observations—does not exclude any effect being enacted by the dimensions represented in those other metrics with respect to the forces at play, i.e., community smells. Further work into figuring out all necessary software people metrics to measure, predict, and manage social debt—and connected forces—is needed.

6. Threats to Validity

There are a number of threats to validity that could have biased our results.

Construct Validity. The first threat in this category concerns with the correctness of the dataset exploited in the study. In this respect, we relied on a publicly available source that we built in the context of our research [22], which contained labeled data of both socio-technical metrics and community smells.

Another potential threat is related to the selection of the independent variables used to build the experimented models. We exploited socio-technical metrics that have

been shown to correlate with the occurrence of community smells [22] and, therefore, our selection was based on previous findings. Nevertheless, we cannot rule out that other metrics, not considered in the study, could provide additional contribution to the performance of community smell prediction models. We plan to investigate this aspect further as part of our future research agenda.

Conclusion Validity. In the context of $\mathbf{RQ}_1$, we measured the predictive power of the considered socio-technical metrics by means of the Gain Ratio evaluation measure [25]. The usage of this algorithm is recommended since it can quantify the amount on predictive uncertainly that is reduced using a specific feature [86], providing us with a mechanism able to rank features based on their importance that we could use to interpret the predictive power of the exploited socio-technical metrics. Finally, we backup our observations through the use of a statistical test, *i.e.*, the Scott-Knott ESD [73], that confirmed the results given by the Gain Ratio evaluation measure.

As for $\mathbf{RQ}_2$ and $\mathbf{RQ}_3$, a first threat is related to the methodology applied to discard redundant features from the models built. We first relied on the Sperman's correlation coefficient [79] to identify pairs of features strongly correlated to each other and, secondly, we excluded one of them by considering the gain they provided to the model (computed using the information gain measure). In so doing, we could only take into account the intra-metric correlation between features, but not their complementarity: for instance, it would be possible that two features taken together offer higher gain compared to the gain obtained by a single feature. While some recent work has proposed heuristics to identify such complementarity relations [97], we could not find any open implementation that would have allowed us to apply them in our context. Hence, we leave to future research work the understanding of how this methodological choice impacts our findings. In the second place, it is worth discussing the implications of the data balancing method employed, *i.e.*, SMOTE [85]. The process it uses to create synthetic instances is stochastic and, therefore, the particular execution on our dataset could have led to inconclusive results. To address this potential threat, we conducted an additional analysis in which we ran SMOTE multiple times and verified the resulting performance. We observed that the performance are in line with those reported in Section 5, with a standard deviation of 0.03, overall. We could therefore conclude that the performance discussed in the paper were not obtained by chance.

As for the the interpretation of the performance achieved by the models built. We mitigated this problem by considering more than one evaluation metric (*e.g.*, F-measure, AUC-ROC, and MCC). Also, previous work has shown the importance of considering data pre-processing actions to properly set machine learning models [81, 42, 98, 53]. For this reason, in our research we considered the application of techniques to deal with data normalization, feature selection, data balancing, and hyper-parameters configuration.

In addition, the validation strategy may be object of discussion: when building the within-project model, we adopted a release-by-release strategy because our data follows a temporal order. Furthermore, it simulates a real-case scenario where a prediction model is updated as soon as new data from a previous release are available [47]. Hence, the selected validation strategy was the only one actually suitable in our context. When building a cross-project solution, the main threat is related to the way we train the model. In particular, we could not deal with the sensitivity of cross-project models to heterogeneous data and, as such, we use all the projects (but the tested one) within the training set. We are aware of this limitation but, unfortunately, there is still no instrument that allows an accurate selection of training data for cross-project models and/or that is publicly available, despite the ongoing research on the topic [87, 88, 89]. Nevertheless, we (i) argue that the reported results represent a lower-bound, yet promising ground for further studies on cross-project community smell prediction and (ii) encourage future research on the matter.

External Validity. With respect to the generalizability of the results, our study considers 60 open-source communities that are all active but different in terms of size, scope, number of contributors, and domain. Yet, we recognize that the performance of the experimented models may differ on other datasets. Hence, we encourage replications of our study targeting different communities as well as closed-source environments.

It is also important to comment the fact that recent work [99] has shown that the top-ranked metrics from the Scott-Knott ESD test may vary among instances of predictions. As a consequence, it is possible that the results of $\mathbf{RQ}_1$ represent a summary of the models, but not provide specific explanation to the predictions. Hence, the results may not be generalized to all projects. We recognize this limitation and, as part of our future research agenda, we plan to address it by means of explainable AI methods, which is a rising research field in the context of software engineering and that may be a useful instrument to provide more insights into the generalizability of the results of our study.

7. Conclusion

In this paper, we aimed at studying how well can community smells be predicted using a set of socio-technical metrics that previous work [22] has shown to be correlated with the phenomenon. To this aim, we carried out an empirical investigation—involving 60 open-source projects—in which we (1) measured the actual predictive power of socio-technical metrics and (2) assessed the performance of within- and cross-project models for

community smell prediction. As a side effect of our study, we also verified the capabilities of various machine learning algorithms. The main results of our study showed that socio-technical congruence, communicability, and turnover-related metrics are top-factors to monitor when predicting the emergence of community-related problems. Furthermore, within-project models can be effectively exploited by practitioners to predict community smells, especially when trained using RANDOM FOREST as classifier. Indeed, our experiment showed that it is possible to achieve an F-Measure of 77%. Finally, in case of lack of historical data, cross-project models represent a promising solution (F-Measure of 62%), yet they still need improvement, *e.g.*, by means of the adoption of proper techniques for selecting training data.

These findings represent the main input for our future research agenda, which includes a replication of our study in an industrial context as well as the definition of *ad-hoc* mechanisms able to improve the performance of community smell prediction models (*e.g.*, though the use of explainable AI methods to improve the model explainability [99]). Furthermore, we will consider the design of empirical studies aiming at understanding the impact of different feature selection methods, able to capture the complementarity relation among features (*e.g.*, the adaptive heuristic proposed by Singha and Shenoy [97]), on the reported results. We also plan to investigate other features that can be used to boost the performance obtained: for instance, we plan to investigate whether and how pure technical-related metrics (*e.g.*, the number of packages) can be used to inform our models and better predict the emergence of community-related issues. At the same time, the major limitation we perceive in terms of actionability of our results—to be addressed in the future—is that, on the one hand, we obtained a useful and usable ML model which is fully reproducible but, on the other hand, we do not provision the model on a DataOps pipeline that practitioners could use. We plan this DataOps-based increment of our ML modelling exercise as a future work.

Acknowledgment

The authors would like to thank the Associate Editor and the anonymous Reviewers for their insightful comments provided during the peer-review process of the article, which were instrumental to improve the quality of the manuscript. Fabio Palomba gratefully acknowledges the support of the Swiss National Science Foundation through the SNF Project No. PZ00P2_186090 (TED). Furthermore, Damians' work is partially supported by the European Commission grant no. 0421 (Interreg ICT), Werkinzicht and the European Commission grants no. 787061 (H2020), ANITA, no. 825040 (H2020), RADON, and no. 825480 (H2020), SODALITE.

References

[1] Paul Ralph, Mike Chiasson, and Helen Kelley. Social theory for software engineering research. In Sarah Beecham, Barbara A. Kitchenham, and Stephen G. MacDonell, editors, *EASE*, pages 44:1–44:11. ACM, 2016.

[2] Anna Hannemann, Hans-Jorg Happel, Matthias Jarke, Ralf Klamma, Steffen Lohmann, Walid Maalej, and Volker Wulf. Social aspects in software engineering. In *Software Engineering (Workshops)*, volume 150 of *LNI*, pages 239–242. GI, 2009.

[3] Helen Sharp, Hugh Robinson, and Mark Woodman. Software engineering: Community and culture. *IEEE Software*, 17(1):40–47, 2000.

[4] Hannu Jaakkola. Culture sensitive aspects in software engineering. In Antje Dusterhoft, Meike Klettke, and Klaus-Dieter Schewe, editors, *Conceptual Modelling and Its Theoretical Foundations*, volume 7260 of *Lecture Notes in Computer Science*, pages 291–315. Springer, 2012.

[5] Lucas Gren. On gender, ethnicity, and culture in empirical software engineering research. *CoRR*, abs/1904.09820, 2019.

[6] Gert Jan Hofstede, Catholijn M. Jonker, and Tim Verwaart. Modeling power distance in trade. In Nuno David and Jaime Simao Sichman, editors, *MABS*, volume 5269 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2008.

[7] Daniel J. Greenhoe. Properties of distance spaces with power triangle inequalities. *PeerJ PrePrints*, 4:e2055, 2016.

[8] Premalatha Packirisamy. Managing power distance to retain talent: Evidence from india. *IJHCITP*, 8(3):49–67, 2017.

[9] Victor Sanchez-Anguix, Tinglong Dai, Zhaleh Semnani-Azad, Katia P. Sycara, and Vicente J. Botti. Modeling power distance and individualism/collectivism in negotiation team dynamics. In *HICSS*, pages 628–637. IEEE Computer Society, 2012.

[10] F. Palomba, D. A. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[11] D. A. Tamburri. Software architecture social debt: Managing the incommunicability factor. *IEEE Transactions on Computational Social Systems*, 6(1):20–37, Feb 2019.

[12] D. A. Tamburri, F. Palomba, and R. Kazman. Exploring community smells in open-source: An automated approach. *IEEE Transactions on Software Engineering*, pages 1–1, 2019.

[13] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrea De Lucia. Do they really smell bad? a study on developers' perception of bad code smells. In *Proceedings of the International Conference on Software Maintenance and Evolution (ICSME)*, pages 101–110. IEEE Computer Society, 2014.

[14] Fabio Palomba, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. Toward a smell-aware bug prediction model. *IEEE Trans. Software Eng.*, 45(2):194–218, 2019.

[15] Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. Mining version histories for detecting code smells. *IEEE Trans. Software Eng.*, 41(5):462–489, 2015.

[16] Antonio Martini and Jan Bosch. Revealing social debt with the caffea framework: An antidote to architectural debt. In *ICSA Workshops*, pages 179–181. IEEE Computer Society, 2017.

[17] Antonio Martini, Terese Besker, and Jan Bosch. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming*, 163:42–61, 2018.

[18] Damian Andrew Tamburri, Kelly Blincoe, Fabio Palomba, and Rick Kazman. "the canary in the coal mine…" a cautionary tale from the decline of sourceforge. *Software: Practice and Experience*, n/a(n/a).

[19] Damian A. Tamburri, Patricia Lago, and Hans van Vliet. Organizational social structures for software engineering. *ACM Comput. Surv.*, 46(1):3:1–3:35, July 2013.

[20] Damian Andrew Tamburri, Patricia Lago, and Hans van Vliet. Uncovering latent social communities in software development. *IEEE Software*, 30(1):29–36, 2013.

[21] Damian A Tamburri, Fabio Palomba, Alexander Serebrenik, and Andy Zaidman. Discovering community patterns in open-source: A systematic approach and its evaluation. *Empirical Software Engineering*, 24(3):1369–1417, 2019.

[22] Damian Andrew Andrew Tamburri, Fabio Palomba, and Rick Kazman. Exploring community smells in open-source: An automated approach. *IEEE Transactions on Software Engineering*, 2019.

[23] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. Social debt in software engineering: insights from industry. *J. Internet Services and Applications*, 6(1):10:1–10:17, 2015.

[24] Jacopo Soldani, Damian Andrew Tamburri, and Willem-Jan Van Den Heuvel. The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232, 2018.

[25] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.

[26] Bogdan Eduard-Madalin Mursa, Anca Andreica, and Laura Diosan. Mining network motif discovery by learning techniques. In Hilde Perez Garcia, Lidia Sanchez-Gonzalez, Manuel Castejon Limas, Hector Quintian-Pardo, and Emilio S. Corchado Rodriguez, editors, *HAIS*, volume 11734 of *Lecture Notes in Computer Science*, pages 73–84. Springer, 2019.

[27] Barbara Russo. Profiling call changes via motif mining. In Andy Zaidman, Yasutaka Kamei, and Emily Hill, editors, *MSR*, pages 203–214. ACM, 2018.

[28] Sarvenaz Choobdar, Pedro Manuel Pinto Ribeiro, and Fernando M. A. Silva. Motif mining in weighted networks. In Jilles Vreeken, Charles Ling, Mohammed Javeed Zaki, Arno Siebes, Jeffrey Xu Yu, Bart Goethals, Geoffrey I. Webb, and Xindong Wu, editors, *ICDM Workshops*, pages 210–217. IEEE Computer Society, 2012.

[29] Paolo Trucco, Enrico Cagno, Fabrizio Ruggeri, and Ottavio Grande. A bayesian belief network modelling of organisational factors in risk analysis: A case study in maritime transportation. *Rel. Eng. & Sys. Safety*, 93(6):845–856, 2008.

[30] Eoin Whelan and Brian Donnellan. Inter-organisational collaboration; a social network analysis approach. In Dan Remenyi, editor, *ECKM*, pages 615–620. Academic Conferences Limited, Reading, UK, 2005.

[31] Michelangelo Ceci, Corrado Loglisci, Eliana Salvemini, Domenica D'Elia, and Donato Malerba. Mining spatial association rules for composite motif discovery. In Renato Bruni, editor, *Mathematical Approaches to Polymer Sequence Analysis and Related Problems*, pages 87–109. Springer, 2011.

[32] Teo Argentieri, Virginio Cantoni, and Mirto Musci. Extending cross motif search with heuristic data mining. In *DEXA Workshops*, pages 57–61. IEEE Computer Society, 2017.

[33] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: A fine-grained approach. In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 563–573, Piscataway, NJ, USA, 2015. IEEE Press.

[34] Nachiappan Nagappan, Brendan Murphy, and Victor Basili. The influence of organizational structure on software quality: an empirical case study. In *International conference on Software engineering*, pages 521–530, Leipzig, Germany, May 2008. IEEE.

[35] Alexander Repenning, Navid Ahmadi, Nadia Repenning, Andri Ioannidou, David Webb, and Krista Marshall. Collective programming: making end-user programming (more) social. pages 325–330.

[36] Davi Viana, Tayana Conte, Dalton Vilela, Cleidson R. B. de Souza, Gleison Santos, and Rafael Prikladnicki. The influence of human aspects on software process improvement: Qualitative research findings and comparison to previous studies. In *EASE*, pages 121–125, 2012.

[37] Andrew Meneely, Laurie Williams, Will Snipes, and Jason A. Osborne. Predicting failures with developer networks and social network analysis. In Mary Jean Harrold and Gail C. Murphy, editors, *SIGSOFT FSE*, pages 13–23. ACM, 2008.

[38] Marcelo Cataldo, James D. Herbsleb, and Kathleen M. Carley. 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*, pages 2–11, New York, NY, USA, 2008. ACM.

[39] Asher Trockman. Adding sparkle to social coding: an empirical study of repository badges in the npm ecosystem. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *ICSE (Companion Volume)*, pages 524–526. ACM, 2018.

[40] Huilian Sophie Qiu, Yucen Lily Li, Hema Susmita Padala, Anita Sarma, and Bogdan Vasilescu. The signals that potential contributors look for when choosing open-source projects. *PACMHCI*, 3(CSCW):122:1–122:29, 2019.

[41] Erik Arisholm, Lionel C Briand, and Eivind B Johannessen. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[42] Chakkrit Tantithamthavorn and Ahmed E Hassan. An experience report on defect modelling in practice: Pitfalls and challenges. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, pages 286–295. ACM, 2018.

[43] Ricardo Baeza-Yates, Berthier Ribeiro-Neto, et al. *Modern information retrieval*, volume 463. ACM press New York, 1999.

[44] Christian Bird, David Pattison, Raissa D'Souza, Vladimir Filkov, and Premkumar Devanbu. Latent social structure in open source projects. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, SIGSOFT '08/FSE-16, pages 24–35, New York, NY, USA, 2008. ACM.

[45] Damian Andrew Tamburri, Philippe Kruchten, Patricia Lago, and Hans van Vliet. Social debt in software engineering: insights from industry. *J. Internet Services and Applications*, 6(1):10:1–10:17, 2015.

[46] Anirban Dasgupta, Jure Leskovec, Michael W. Mahoney, and Kevin J. Lang. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1), 1 2009.

[47] Fabio Palomba, Damian Andrew Andrew Tamburri, Francesca Arcelli Fontana, Rocco Oliveto, Andy Zaidman, and Alexander Serebrenik. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Transactions on Software Engineering*, 2018.

[48] Stanley Wasserman and Katherine Faust. *Social Network Analysis. Methods and Applications.* Cambridge University Press, 1994.

[49] Mitchell Joblin, Wolfgang Mauerer, Sven Apel, Janet Siegmund, and Dirk Riehle. From developer networks to verified communities: A fine-grained approach. In Antonia Bertolino, Gerardo Canfora, and Sebastian G. Elbaum, editors, *Proc. Int'l Conf. on Software Engineering (ICSE)*, pages 563–573. IEEE Computer Society, 2015.

[50] Fabio Palomba and Damian A. Tamburri. Predicting the emergence of community smells using socio-technical metrics - online appendix - `https://figshare.com/articles/Predicting_the_Emergence_of_Community_Smells_using_Socio-Technical_Metrics/9805394`, 2019.

[51] Dario Di Nucci, Fabio Palomba, Damian A Tamburri, Alexander Serebrenik, and Andrea De Lucia. Detecting code smells using machine learning techniques: are we there yet? In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 612–621. IEEE, 2018.

[52] Dario Di Nucci, Fabio Palomba, Rocco Oliveto, and Andrea De Lucia. Dynamic selection of classifiers in bug prediction: An adaptive method. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 1(3):202–212, 2017.

[53] Fabiano Pecorelli, Fabio Palomba, Dario Di Nucci, and Andrea De Lucia. Comparing heuristic and machine learning approaches for metric-based code smell detection. In *Proceedings of the 27th International Conference on Program Comprehension*, pages 93–104. IEEE Press, 2019.

[54] Fabiano Pecorelli, Dario Di Nucci, Coen De Roover, and Andrea De Lucia. On the role of data balancing for machine learning-based code smell detection. 2019.

[55] Mitchell Joblin, Sven Apel, Claus Hunsen, and Wolfgang Mauerer. Classifying developers into core and peripheral: An empirical study on count and network metrics. *arXiv preprint arXiv:1604.00830*, 2016.

[56] D. Tamburri, F. Palomba, and R. Kazman. Success and failure in software engineering: a followup systematic literature review. *ArXiv*, abs/2006.12086, 2020.

[57] Andrew Meneely and L. Williams. Socio-technical developer networks: should we trust our measurements? *2011 33rd International Conference on Software Engineering (ICSE)*, pages 281–290, 2011.

[58] I. Kwan, A. Schroter, and D. Damian. Does socio-technical congruence have an effect on software build success? a study of coordination in a software project. *IEEE Transactions on Software Engineering*, 37(3):307–324, 2011.

[59] Guilherme Avelino, Marco Tulio Valente, and Andre C. Hora. What is the truck factor of popular github applications? a first assessment. *PeerJ PrePrints*, 5:e1233, 2017.

[60] Filippo Ricca, Alessandro Marchetto, and Marco Torchiano. On the difficulty of computing the truck factor. In Danilo Caivano, Markku Oivo, Maria Teresa Baldassarre, and Giuseppe Visaggio, editors, *PROFES*, volume 6759 of *Lecture Notes in Business Information Processing*, pages 337–351. Springer, 2011.

[61] Christina Manteli, Bart van den Hooff, Hans van Vliet, and Wilco van Duinkerken. Overcoming challenges in global software development: The role of brokers. In Marko Bajec, Martine Collard, and Rebecca Deneckere, editors, *RCIS*, pages 1–9. IEEE, 2014.

[62] Christina Manteli, Bart van den Hooff, and Hans van Vliet. The effect of governance on global software development: An empirical research in transactive memory systems. *Information & Software Technology*, 56(10):1309–1321, 2014.

[63] Sidney W. A. Dekker. Deferring to expertise versus the prima donna syndrome: a manager's dilemma. *Cogn. Technol. Work.*, 16(4):541–548, 2014.

[64] Richard Leifer and André Delbecq. Organizational/environmental interchange: A model of boundary spanning activity. *The Academy of Mgmt. Rev.*, 3(1):40–50, January 1978.

[65] Natalia Levina and Emmanuelle Vaast. The emergence of boundary spanning competence in practice: Implications for implementation and use of information systems. *MIS Quarterly*, 29(2):335–363, 2005.

[66] Wenyu Du and Shan Ling Pan. Boundary spanning by design: Toward aligning boundary-spanning capacity and strategy in it outsourcing. *IEEE Trans. Engineering Management*, 60(1):59–76, 2013.

[67] Claude E Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[68] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[69] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E. Hassan, and Kenichi Matsumoto. An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Softw. Eng.*, 43(1):1–18, January 2017.

[70] A. J. Scott and M. Knott. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 30:507–512, 1974.

[71] Suhas Kabinna, Weiyi Shang, Cor-Paul Bezemer, and Ahmed E Hassan. Examining the stability of logging statements. In *Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd Int'l Conf. on*, volume 1, pages 326–337. IEEE, 2016.

[72] Heng Li, Weiyi Shang, Ying Zou, and Ahmed E Hassan. Towards just-in-time suggestions for log changes. *Empirical Software Engineering*, pages 1–35, 2016.

[73] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, Akinori Ihara, and Kenichi Matsumoto. The impact of mislabelling on the performance and interpretation of defect prediction models. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE Int'l Conf. on*, volume 1, pages 812–823. IEEE, 2015.

[74] Robert J. Grissom and John J. Kim. *Effect sizes for research: A broad practical approach*. Lawrence Earlbaum Associates, 2nd edition edition, 2005.

[75] Gregory G Dess and Jason D Shaw. Voluntary turnover, social capital, and organizational performance. *Academy of management review*, 26(3):446–456, 2001.

[76] Dan R Dalton, David M Krackhardt, and Lyman W Porter. Functional turnover: An empirical assessment. *Journal of applied psychology*, 66(6):716, 1981.

[77] Arie C Glebbeek and Erik H Bax. Is high employee turnover really harmful? an empirical test using company records. *Academy of Management Journal*, 47(2):277–286, 2004.

[78] Robert M. O'brien. A caution regarding rules of thumb for variance inflation factors. *Quality & Quantity*, 41(5):673–690, Oct 2007.

[79] Yadolah Dodge. *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008.

[80] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[81] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 2018.

[82] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[83] Ping Zhang. Model selection via multifold cross validation. *The Annals of Statistics*, pages 299–313, 1993.

[84] Damjan Krstajic, Ljubomir J Buturovic, David E Leahy, and Simon Thomas. Cross-validation pitfalls when selecting and assessing regression and classification models. *Journal of cheminformatics*, 6(1):10, 2014.

[85] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[86] Peter Flach. *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press, 2012.

[87] Fayola Peters, Tim Menzies, and Andrian Marcus. Better cross company defect prediction. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 409–418. IEEE Press, 2013.

[88] Rahul Krishna, Tim Menzies, and Wei Fu. Too much automation? the bellwether effect and its implications for transfer learning. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 122–131. ACM, 2016.

[89] Rahul Krishna and Tim Menzies. Bellwethers: A baseline method for transfer learning. *IEEE Transactions on Software Engineering*, 2018.

[90] Yoram Reich and SV Barai. Evaluating machine learning models for engineering problems. *Artificial Intelligence in Engineering*, 13(3):257–272, 1999.

[91] James A Hanley and Barbara J McNeil. The meaning and use of the area under a receiver operating characteristic (roc) curve. *Radiology*, 143(1):29–36, 1982.

[92] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[93] Joseph L Gastwirth. The estimation of the lorenz curve and gini index. *The review of economics and statistics*, pages 306–316, 1972.

[94] Dario Di Nucci, Fabio Palomba, and Andrea De Lucia. Evaluating the adaptive selection of classifiers for cross-project bug prediction. In *2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*, pages 48–54. IEEE, 2018.

[95] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 91–100. ACM, 2009.

[96] Foyzur Rahman, Daryl Posnett, and Premkumar Devanbu. Recalling the imprecision of cross-project defect prediction. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 61. ACM, 2012.

[97] Sumanta Singha and Prakash P Shenoy. An adaptive heuristic for feature selection based on complementarity. *Machine Learning*, 107(12):2027–2071, 2018.

[98] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Ahmed E Hassan. The impact of correlated metrics on the interpretation of defect models. *IEEE Transactions on Software Engineering*, 2019.

[99] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, Hoa Khanh Dam, and John Grundy. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Transactions on Software Engineering*, 2020.

Appendix A. Characteristics of the considered projects

In this appendix, we report the characteristics of the open-source communities analyzed in our work.

Table A.4: List of analyzed projects.

#	Project	Source code repository	Development mailing list
1	LibreOffice	http://anongit.freedesktop.org/git/libreoffice/core.git	gmane.comp.documentfoundation.libreoffice.devel
2	Firefox	https://github.com/mozilla/gecko-dev.git	gmane.comp.mozilla.firefox.devel
3	FFmpeg	git://source.ffmpeg.org/ffmpeg.git	gmane.comp.video.ffmpeg.devel
4	Cassandra	http://git-wip-us.apache.org/repos/asf/cassandra.git	gmane.comp.db.cassandra.devel
5	Git	https://github.com/git/git.git	gmane.comp.version-control.git
6	OpenSSL	git://git.openssl.org/openssl.git	gmane.comp.encryption.openssl.devel
7	GRUB	git://git.savannah.gnu.org/grub.git	gmane.comp.boot-loaders.grub.devel
8	nginx	https://github.com/nginx/nginx.git	gmane.comp.web.nginx.devel
9	Audacity	https://github.com/audacity/audacity.git	gmane.comp.audio.audacity.devel
10	VLC	git://git.videolan.org/vlc.git	gmane.comp.video.videolan.vlc.devel
11	Tomcat	git://git.apache.org/tomcat.git	gmane.comp.jakarta.tomcat.devel
12	GIMP	git://git.gnome.org/gimp	gmane.comp.video.gimp.devel
13	Guix	git://git.savannah.gnu.org/guix/dhcp.git	gmane.comp.gnu.guix.devel
14	Mahout	git://git.apache.org/mahout.git	gmane.comp.apache.mahout.devel
15	CXF	git://git.apache.org/cxf.git	gmane.comp.apache.cxf.devel
16	Rails	https://github.com/rails/rails.git	gmane.comp.lang.ruby.rails.core
17	AngularJS	https://github.com/angular/angular.js.git	gmane.comp.lang.javascript.angularjs
18	libuv	https://github.com/libuv/libuv.git	gmane.comp.lang.javascript.nodejs.libuv
19	Bitcoin	https://github.com/bitcoin/bitcoin	gmane.comp.bitcoin.devel
20	Scala	https://github.com/scala/scala	gmane.comp.lang.scala
21	matplotlib	https://github.com/matplotlib/matplotlib	gmane.comp.python.matplotlib.devel
22	Qt	git://code.qt.io/qt/qtbase.git	gmane.comp.lib.qt.devel
23	NodeJS	https://github.com/nodejs/node.git	gmane.comp.lang.javascript.nodejs
24	GitLab	https://github.com/gitlabhq/gitlabhq.git	gmane.comp.version-control.gitlab
25	Tornado	https://github.com/tornadoweb/tornado.git	gmane.comp.python.tornado
26	Arduino	https://github.com/arduino/Arduino.git	gmane.comp.hardware.arduino.devel
27	ipython	https://github.com/ipython/ipython	gmane.comp.python.ipython.devel
28	Lucene	git://git.apache.org/lucene-solr.git	gmane.comp.jakarta.lucene.devel
29	Capistrano	https://github.com/capistrano/capistrano	gmane.comp.lang.ruby.capistrano.general
30	Django	https://github.com/django/django.git	gmane.comp.python.django.devel
31	Salt	https://github.com/saltstack/salt.git	gmane.comp.sysutils.salt.user
32	mongoose	https://github.com/Automattic/mongoose.git	gmane.comp.lang.javascript.mongoose
33	APR	git://git.apache.org/apr.git	gmane.comp.apache.apr.devel
34	Jackrabbit	git://git.apache.org/jackrabbit.git	gmane.comp.apache.jackrabbit.devel
35	Gnome-shell	git://git.gnome.org/gnome-shell	gmane.comp.gnome.shell
36	Krita	git://anongit.kde.org/krita.git	gmane.comp.kde.devel.krita
37	Blender	https://git.blender.org/blender.git	gmane.comp.video.blender.devel
38	Vagrant	https://github.com/mitchellh/vagrant.git	gmane.comp.tools.vagrant
39	NetworkManager	git://anongit.freedesktop.org/NetworkManager/NetworkManager.git	gmane.linux.network.networkmanager.devel
40	Eclipse CDT	https://git.eclipse.org/r/cdt/org.eclipse.cdt	gmane.comp.ide.eclipse.cdt.devel
41	Enlightenment	https://git.enlightenment.org/core/enlightenment.git	gmane.comp.window-managers.enlightenment.devel
42	libva	git://anongit.freedesktop.org/git/libva	gmane.comp.freedesktop.libva
43	JDO	http://svn.apache.org/repos/asf/db/jdo	gmane.comp.apache.db.jdo.devel
44	Jena	git://git.apache.org/jena.git	gmane.comp.apache.jena.devel
45	OpenNLP	git://git.apache.org/opennlp.git	gmane.comp.apache.opennlp.devel
46	Cayenne	git://git.apache.org/cayenne.git	gmane.comp.java.cayenne.devel
47	Pig	git://git.apache.org/pig.git	gmane.comp.java.hadoop.pig.devel
48	Calligra	git://anongit.kde.org/calligra.git	gmane.comp.kde.devel.calligra
49	Wine	git://source.winehq.org/git/wine.git	gmane.comp.emulators.wine.devel
50	Mallet	https://github.com/mimno/Mallet.git	gmane.comp.ai.mallet.devel
51	Gstreamer	git://anongit.freedesktop.org/gstreamer/gstreamer	gmane.comp.video.gstreamer.devel
52	U-boot	http://git.denx.de/u-boot.git	gmane.comp.boot-loaders.u-boot
53	LLVM	http://llvm.org/git/llvm	gmane.comp.compilers.llvm.devel
54	gPhoto	svn://svn.code.sf.net/p/gphoto/code/trunk	gmane.comp.multimedia.gphoto.devel
55	Emacs	git://git.savannah.gnu.org/emacs.git	gmane.emacs.devel
56	QEMU	git://git.qemu.org/qemu.git	gmane.comp.emulators.qemu
57	Python	https://github.com/python/cpython.git	gmane.comp.python.devel
58	Mesa	git://anongit.freedesktop.org/mesa/mesa	gmane.comp.video.mesa3d.devel
59	Sympy	git://github.com/sympy/sympy.git	gmane.comp.python.sympy
60	Okular	git://anongit.kde.org/okular	gmane.comp.kde.devel.okular