Huang ZJ, Shao ZQ, Fan GS *et al.* Community smell occurrence prediction on multi-granularity by developer-oriented features and process metrics. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 37(1): 182–206 Jan. 2022. DOI 10.1007/s11390-021-1596-1

Community Smell Occurrence Prediction on Multi-Granularity by Developer-Oriented Features and Process Metrics

Zi-Jie Huang[1] (黄子杰), *Student Member*, *CCF*, *IEEE*, Zhi-Qing Shao[1,*] (邵志清)
Gui-Sheng Fan[1,2,*] (范贵生), *Member*, *CCF*
Hui-Qun Yu[1,3] (虞慧群), *Senior Member*, *CCF*, *IEEE*, *Member*, *ACM*, Xing-Guang Yang[1] (杨星光), and Kang Yang[1] (杨　康)

[1] *Department of Computer Science and Engineering, East China University of Science and Technology Shanghai 200237, China*

[2] *Shanghai Key Laboratory of Computer Software Testing and Evaluating, Shanghai 200237, China*

[3] *Shanghai Engineering Research Center of Smart Energy, Shanghai 200237, China*

E-mail: hzj@mail.ecust.edu.cn; {zshao, gsfan, yhq}@ecust.edu.cn; {y12180301, y12190041}@mail.ecust.edu.cn

Received May 18, 2021; accepted January 10, 2022.

Abstract Community smells are sub-optimal developer community structures that hinder productivity. Prior studies performed smell prediction and provided refactoring guidelines from a top-down aspect to help community shepherds. Simultaneously, refactoring smells also requires bottom-up effort from every developer. However, supportive measures and guidelines for them are not available at a fine-grained level. Since recent work revealed developers' personalities and working states could influence community smells' emergence and variation, we build prediction models with experience, sentiment, and development process features of developers considering three smells including Organizational Silo, Lone Wolf, and Bottleneck, as well as two related classes including smelly developer and smelly quitter. We predict the five classes in the individual granularity, and we also generate forecasts for the number of smelly developers in the community granularity. The proposed models achieve F-measures ranging from 0.73 to 0.92 in individual-wide within-project, time-wise, and cross-project prediction, and mean R^2 performance of 0.68 in community-wide Smelly Developer prediction. We also exploit SHAP (SHapley Additive exPlanations) to assess feature importance to explain our predictors. In conclusion, we suggest developers with heavy workload should foster more frequent communication in a straightforward and polite way to build healthier communities, and we recommend community shepherds to use the forecasting model for refactoring planning.

Keywords community smell, developer sentiment, socio-technical analysis, empirical software engineering

1 Introduction

Software quality is a major concern of stakeholders. In response, researchers and practitioners adopt various Software Quality Assurance (SQA) approaches in early and late phases of software release life cycle such as requirement engineering, static and dynamic analysis of programs, and code review to address the concern. However, the potential and the impact of SQA within the development process are less studied since the process contains complex socio-technical interactions that incorporate both the communication and the collaboration of developers and other stakeholders [1]. Consequently, the outputs of SQA tools in purely technical aspects are perceived as unhelpful by practitioners [2] due to the lack of adaptation to the context of developers' tasks and working states [3, 4].

Regular Paper
Special Section on Software Systems 2021—Theme: Internetware and Beyond
A preliminary version of the paper was published in the Proceedings of ICPC 2021.
This work was partially supported by the National Natural Science Foundation of China under Grant No. 61772200, and the Natural Science Foundation of Shanghai under Grant No. 21ZR1416300.

*Corresponding Author (Zhi-Qing Shao helped with the original idea and theoretic design. Gui-Sheng Fan significantly helped with the implementation and writing. They both guided the work of the paper and contributed equally.)

Inspired by the conception of code smell [5] (i.e., suboptimal code implementation choices causing technical debt [6]), Tamburri *et al.* [1] coined the term "community smell" to describe the unhealthy organizational structure of the Open-Source Software (OSS) developer communities causing social debt [7]. Social debt refers to unforeseen project cost connected to the presence of non-cohesive developer communities having communication or collaboration issues. Community smells were proved to be preventers of refactoring [8]. Lacking appropriate refactoring may cause continuous quality degradation that accelerates software aging, which is closely related to software failure.

To evaluate community smells, researchers performed smell detection over collaboration and communication activities at the granularity of community subgroups [1, 8, 9]. Furthermore, top-down strategies and empirical guidelines [10, 11] to refactor community smells such as team mentoring, monitoring, and restructuring were proposed for community shepherds (e.g., core members [12] and architects [13]).

Although OSS development originates from an iconic culture [14], it is now being shaped to a more diverse and egalitarian process [14, 15] by recent evolvement based on major social coding platforms such as GitHub. Unlike the conventions in hierarchical and centralized organizations, the orders and arrangements of community shepherds may not be strictly executed and followed [15] by individual developers (also known as OSS grassroots [14]). Moreover, shepherds may be leaving [12] from communities. Consequently, the top-down guidelines may be ineffective in practice. Therefore, it is necessary to reflect the interests of the majority in the developer community and focus on the individual developers whose opinions and thoughts are often neglected [14].

Community smells are caused by unhealthy development activities. Such activities are driven by developers' motifs [8]. Since restructuring smelly communities relies on the efforts of every member [10], we intend to build a bottom-up adaptive measure for individual developers to prevent community smells from occurring. Empirical research showed the introduction and the refactoring of community smells are influenced by developers' personalities [11]. Since developers' personalities are hard to capture in software artifacts [11], we focus on their expression, i.e., developer sentiment. Sentiment can significantly affect the quality of work [16]. Developer sentiment was proved to have impact on various aspects in software engineering including issue reopening [17] and commit changes [18]. Furthermore, we also involve developers' experience features and development process metrics to capture their performance and workload, which has been proved effective in related tasks such as defect severity prediction [19].

To the best of our knowledge, there lacks a community smell prediction model designed individually for developers. This paper fills the gap by creating machine learners upon a developer sentiment dataset [20] to predict if a developer is affected by three common community smells, such as Lone Wolf, Organization Silo, and Bottleneck [1]. Furthermore, the models also predict if a developer quitted the community after being affected by any community smell. Afterwards, discussions are made on the difference of features related to smelly and non-smelly predictions. We analyze the significance of the differences in features' distributions as well as the effect sizes of the differences. Finally, we also forecast the number of Smelly Developer's occurrence on a higher granularity, i.e., community-wide, to extend our scope for community shepherds to make decisions about community refactoring.

The main contributions of this paper are as follows.

- We build machine learning models integrating developer sentiment, experience, and process metrics. To support both the developers and the community shepherds on community refactoring, we perform predictions on 1) the individual granularity, i.e., community smells' occurrence on each developer, and 2) the community granularity, i.e., the number of developers affected by any of the smell in a given analysis window respectively.
- For the individual granularity, we reveal that several experience features and process metrics (e.g., commented sentences, code churn) are stronger predictors compared with the sentiment ones, while sentiment features also contribute moderate predictive power. Experimental results show our model has ideal performance in most cases in all three validation scenarios. We also draw a conclusion that developers with heavy workload should foster more frequent communication in a straightforward and polite way to ensure community healthiness after evaluating the relationship between the prediction results and the features.
- For the community granularity, we build three kinds of models, including classifiers (i.e., the summation of individual prediction results), regressors with the reconstructed dataset using mean and summation values of the features, and time series based predictors. We reveal that time series based prediction is less ideal

than regression and classification. However, regressors and classifiers have different behaviors, e.g., sentimental features are more important than other features for regressors. Experimental results show our model has good performance for community-wide community smells' forecasting in 12 months.

The major improvement based on the preliminary conference paper [21] of this work includes the followings.

• We extend our research questions and goals by introducing new community-wide community smell occurrence forecasting to validate whether our approach would be useful for community shepherds to make plans about refactoring smells.

• We revise our data and validation strategy with more explainable time-sensitive approaches. We regenerate our dataset to ensure no feature is calculated based on future data, and we introduce time-wise validation for prediction and model explanation. Meanwhile, we update our online appendix①, which includes additional experimental results and an extended dataset.

• We update our features for prediction. We discard the activeness features (i.e., three metrics that can only be generated in the current analysis window) to avoid the violation of time series. Meanwhile, we introduce 10 experience features that comply with time-wise prediction, as well as 10 process metrics to capture developers' activities in VCS (Version Control System).

• We renew the model explanation approach and provide more visualization, i.e., replacing information gain with SHAP since it provides more stable and interpretable results for local and global predictions.

• We enhance the former conclusion from the aspect of developers' workload and experience, i.e., more frequent communications should be fostered for developers with heavy workload. Meanwhile, we also discuss the inconsistency of feature importance between this paper and the preliminary conference paper [21].

The rest of this paper is organized as follows. In Section 2 we summarize related literature. Section 3 presents how we construct our dataset, while Section 4 outlines the settings and research questions, as well as the concerned evaluation metrics. In Section 5 we discuss the results, while Section 6 overviews the threats to the validity of the study and our effort to cope with them. Finally, Section 7 concludes the paper and describes future research.

2 Related Work

This section describes researches related to two aspects of this paper, i.e., community smell, and socio-technical analysis of software artifacts.

2.1 Community Smells

Researchers contributed a series of studies concerning the definition [1, 22], detection [1], diffuseness [1, 23], and variability [11] of community smells, as well as their impact on software maintainability [8]. Tamburri *et al.* [1] defined community smells as patterns of motifs over collaboration and communication graphs, and they implemented a detection tool called CODEFACE4SMELLS [1]. Furthermore, they validated qualitatively the acceptance of the detection results, and they discovered the results were all true positives. Palomba *et al.* [8] observed that community smells could be the preventers of refactoring. Meanwhile, community smells also intensify code smells continuously [8]. In terms of analyzing community smell in the granularity of developers individually, Catolino *et al.* [10] provided practitioners with refactoring suggestions and frameworks. Catolino *et al.* [11] also pointed out that communicability is important to prevent community smells, and developers' personalities play an important role in producing smells. The prediction of community smells has also been actively studied by the research community. Palomba and Tamburri [9] built a state-of-the-art model to predict community smells' emergence on within- and cross-project scenarios. They also revealed that socio-technical congruence, communicability, and turnover-related metrics are the most powerful predictors to the occurrence of community smells on community sub-groups.

The major differences of this work to the above-mentioned smell prediction papers are: 1) we predict the occurrence of smells from a bottom-up aspect, i.e., in the granularity of developers individually rather than the top-down aspects from sub-groups or communities; 2) we involve developers' experience and sentiment features as well as process metrics to build predictors, and we assess their predictive power as well as statistical characteristics in software projects, which were not covered in prior studies.

① https://github.com/SORD-src/JCST_Replication, Jan. 2022.

2.2 Socio-Technical Features of Software Artifacts

Ortu *et al.* [20] constructed a multi-aspect developer sentiment dataset based on comments and sentences in JIRA Issue Tracking Systems (ITS). This dataset is regarded as the golden dataset [24] for sentiment analysis in software engineering. Ortu *et al.* [20] also found sentiment may impact software productivity. For example, impolite comments [25] and bullies [26] are related to longer issue fixing time. Meanwhile, certain combinations of VAD [16] (Valence-Arousal-Dominance) scores may indicate longer issue resolution time, as well as productivity problems such as burnout.

Experience features are widely applied to calculate process metrics of software defect and code smell prediction models. Valvida-Garcia *et al.* [19] integrated experience features of reporters to build blocking bug prediction models based on various classical machine learning classifiers. Notably, another contribution [27] reported a trivial impact of an individual developer experience metric in purely technical aspects (i.e., the count of co-committing developers in a file) to similar tasks. However, empirical evidence showed there exist complex interactions among developer experience, community types, and community smells' emergence [28].

Process metrics are capable of capturing developer workload characteristics in software development. They have been proved effective in predicting change-sensitive code smells and software defects at commit granularity. For example, Yang *et al.* [29] applied 14 process metrics (called change metrics in their paper) related to the number of subsystems, change size, change interval, and developer experience to train supervised and unsupervised models for defect prediction. Based on these classical features, McIntosh and Kamei [30] involved code review process metrics to help predict fix-inducing changes in the OpenStack community. In terms of code smell-related tasks, Palomba *et al.* [3] incorporated developer-oriented metrics and process metrics to predict the severity of code smells, as they were perceived important by the original developers of code affected by code smells.

To sum up, the above-mentioned features have connections with major SQA tasks. However, a generally accepted theorem explaining the pattern and impact of the connection of socio-technical features in detail is not available at present.

3 Feature Extraction and Community Smell Detection

This section describes how we generate our dataset including the features and the community smells' occurrences.

3.1 Extracting Developer-Oriented Features and Process Metrics

First, we recover and adjust the raw sentiment data [20], i.e., sentiments of every developer of specific projects in JIRA ITS located in two data tables called jira_issue_comment and jira_user.

Then, we export the experience data using the community smell detection tool called Codeface4Smells. Moreover, it also calculates socio-technical metrics in given analysis windows based on developers' activities in mailing lists and VCS. The detailed settings of the tool will be described in Subsection 3.3.

Afterwards, we use a commonly applied tool in software repository mining tasks [3], i.e., PyDriller [31], to extract process metrics from the software VCS.

Finally, we group the features by time series, developers, and projects. Details of the features will be described in Subsection 4.2.2.

3.2 Selecting Projects and Fetching Mailing Lists

The JIRA sentiment dataset consists of 23 projects whose comments were populated with sentiment data. Among the 23 projects, we first exclude three projects whose mailing list service providers do not support archive extraction. Next, we exclude four more projects as their mailing lists do not cover the time range of their sentiment data. Afterwards, we remove three software projects as they share the same JIRA ITS or VCS repository with other projects, and such projects are incompatible with Codeface4Smells. Finally, we also filter out one project whose VAD data are missing in the dataset. To sum up, the actual 12 projects we use to perform the analysis are listed in Table 1. We fetch their mailing lists from the archives provided by their open-source foundations.

3.3 Detecting Community Smells

We apply the state-of-the-art tool Codeface4Smells to detect community smells. We follow strictly the instructions of the Codeface4Smells repository, e.g., we execute the application in the suggested Vagrant

Table 1. Analyzed Projects

Project Name	Repository	Number of Developers	Date Range	Number of Issues	Mailing List	Description
HBase	ASF	919	2007-04–2014-04	68 806	hbase-dev②	Distributed database
Hadoop Common	ASF	1 221	2009-05–2014-02	61 905	commons-dev②	Utilities library
Hadoop HDFS	ASF	745	2009-05–2011-04	42 188	hadoop-hdfs-dev②	Distributed file system
Cassandra	ASF	1 161	2009-03–2014-03	41 937	cassandra-dev②	Distributed database
Hadoop Map/Reduce	ASF	857	2009-05–2011-03	34 747	hadoop-mapreduce-dev②	Programming model
Hive	ASF	839	2008-09–2014-03	34 449	hive-dev②	Data warehouse
Harmony	ASF	306	2005-09–2011-07	28 325	harmony-dev②	Modular Java runtime
OFBiz	ASF	538	2006-07–2014-02	25 667	ofbiz-dev②	Business planning
Hibernate ORM	JBoss	3 958	2007-06–2014-01	23 549	hibernate-dev③	Object relational mapping
Camel	ASF	882	2007-04–2014-01	21 758	camel-dev②	Application integration
Wicket	ASF	1 210	2006-10–2014-01	17 030	wicket-dev②	Application framework
Zookeeper	ASF	484	2005-09–2011-07	13 634	zookeeper-dev②	Application management

instance, and we fix the broken dependencies in order to avoid platform-specific problems. We do not make any modifications except adding exportation features to the socio-technical analysis script in order to derive names and e-mails of developers affected by community smells.

As for configurations, community smell analysis must be performed in a given window. In our case, the window is three months as prior studies suggested [1, 9, 22]. According to the settings in the replication package, we also specify every commit to analyze in configuration files [22], and the commits are exported from Git repositories using PyDriller [31]. The configuration files are also provided in our online appendix④.

Codeface4Smells is able to detect five community smells, including Organization Silo, Lone Wolf, Bottleneck, Black Cloud, and Prima Donnas. However, Prima Donnas detection is not empirically proved effective [1] since its first appearance [22], and thus we do not take this community smell into consideration. Black Cloud is sparsely distributed in software systems [1]. In our dataset, there are several Black Cloud appearances. However, the developers affected are not captured in the sentiment dataset. Consequently, we could not perform Black Cloud prediction. Thus, our research scope includes three community smells, namely Organizational Silo, Lone Wolf, and Bottleneck.

Figs.1–3 illustrate examples of the three concerning community smells. Fig.1 and Fig.2 include examples of smells detectable based on both communication and collaboration graphs. Fig.1 shows a siloed area of developers (Dev.), while Fig.2 illustrates two developers affected by Lone Wolf collaborating on shared code with indirect communication. To clarify, Organizational (Org.) Silo is a subset of Lone Wolf [22]. Technically, the major difference between the two smells is the definition of lacking connectivity in the communication graph. Organizational Silo is detected if collaborating developers are disconnected in the communication graph. Lone Wolf is detected if collaborating developers are not neighbours in the communication graph, i.e., the developers lack one-degree direct connections among them. Fig.3 shows a case of Bottleneck, which is detected based on the communication graph only. The definition of Bottleneck is similar to the unique boundary spanner in social-network analysis which blocks communication among community sub-groups [1]. Apart from the three smells, we also involve Smelly Developer and Smelly Quitter as prediction classes, and the details will be described in Subsection 4.2.1.

4 Empirical Study Setup

The goal of our study is to evaluate to what extent the occurrence of community smells on developers can be predicted by their experience and sentiment as well as process metrics, with the purpose of understanding the impact of the concerning features on community smells from the two granularities of individual-wide and community-wide. This section outlines our methodology, which is depicted in Fig.4.

②http://mail-archives.apache.org/mod_mbox/{Mailing List Name}, Nov. 2021.

③https://lists.jboss.org/archives/list/{Mailing List Name}@lists.jboss.org/, Nov. 2021.

④https://github.com/SORD-src/JCST_Replication, Jan. 2022.

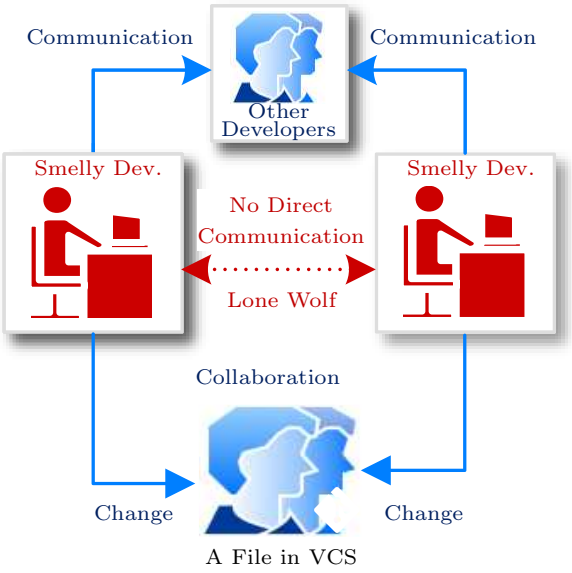


Fig.1. Lone Wolf community smell.

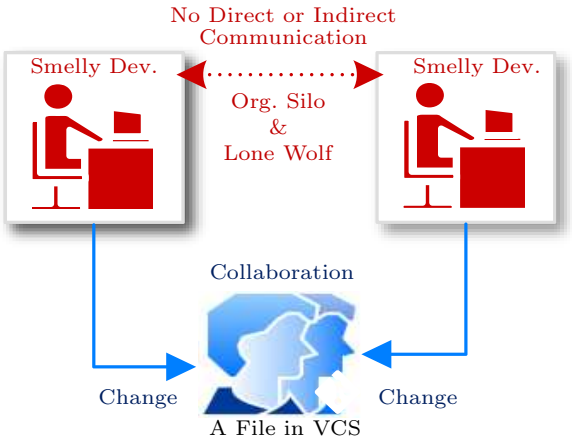


Fig.2. Lone Wolf and Organizational Silo community smell.

4.1 Research Questions and Motivation

We define our research questions according to the two goals of SQA related prediction studies [32], i.e., 1) to predict the likelihood of the issue's occurrence, and 2) to understand and to explain the characteristics associated with the outcome.

*RQ*1. To what extent can we predict the occurrence of community smells on individual developers using the proposed features?

Motivation. RQ1 refers to prediction settings and performance. Based on the dataset built in Section 3, we intend to explore if the community smells' occurrence could be predicted using only the data from the former analysis windows.

Approach. We define dependent and independent variables, and we build prediction models using machine learning classifiers. To pick the most appropriate classifier, we exploit several classical evaluation metrics to assess their performance.

*RQ*2. Can we explain the behavior of the best-performed models of RQ1 in individual-wide prediction?

Motivation. Recent work [32] reveals that most SQA machine learning papers lack interpretation of the generated models, while inexplainable models are not acceptable for practitioners. Moreover, using such models in practice may violate data privacy regulations [33]. Since predictive power could speak for the impact of features on the prediction results [3, 9], the local interpretations of model behaviors should reflect the interactions between feature values and community smell

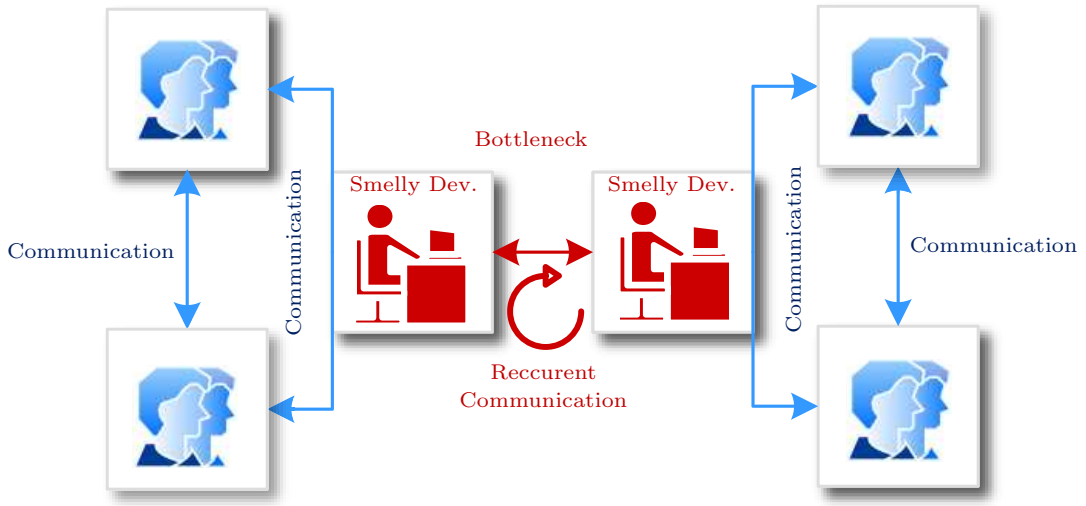


Fig.3. Bottleneck community smell.

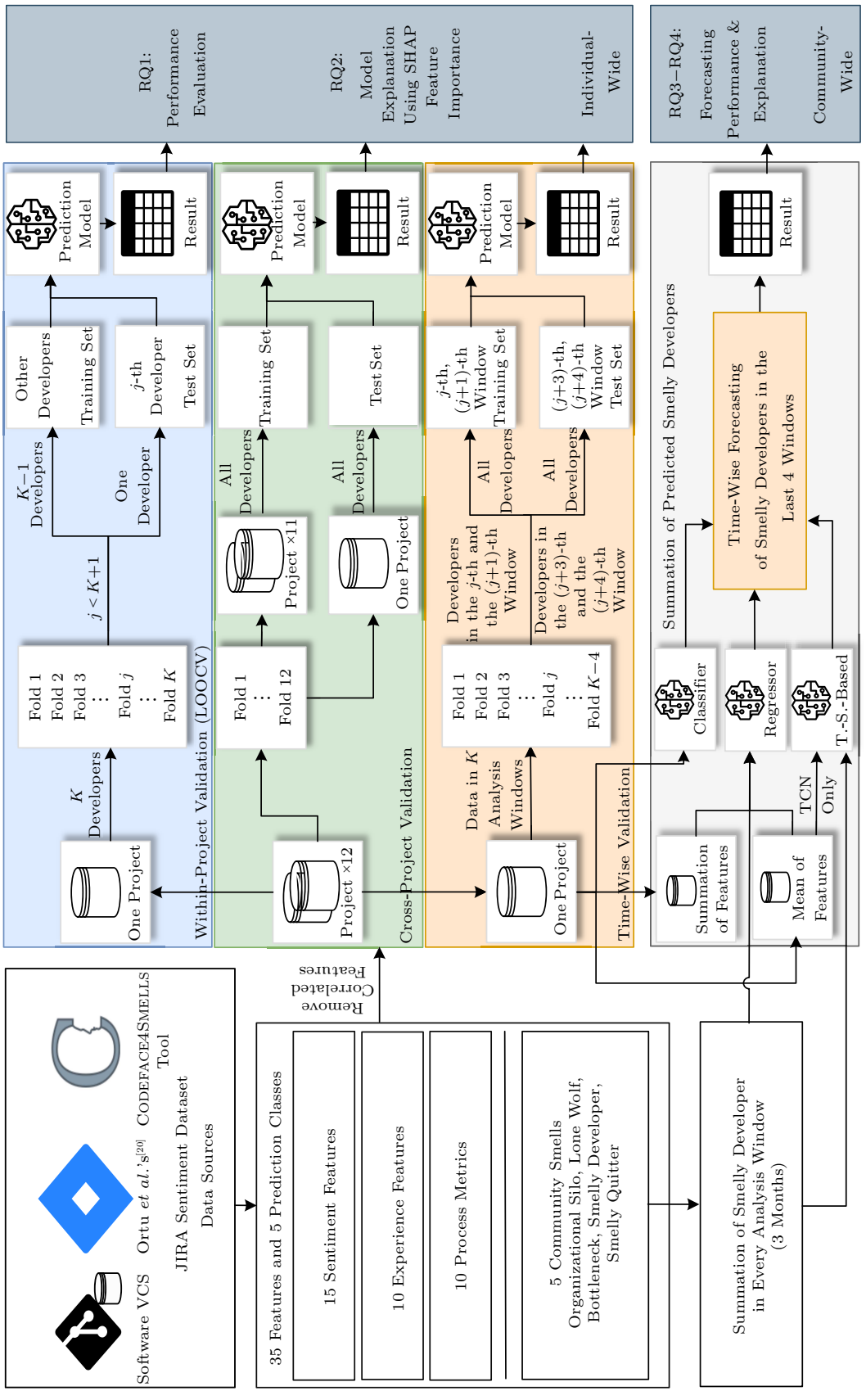


Fig.4. Overview of the prediction process.

occurrences [34], because 1) we analyze correctly predicted instances in the classes that the model is well-performed, and 2) SHAP produces feature importance close to human intuition [35]. Thus, we propose this RQ to understand the rankings and characteristics of features' importance generally by explaining the features' interactions with the model behaviour. We try to figure out if higher (lower) values of certain features lead to smelly (non-smelly) prediction results. Based on these observations, we intend to explain how sentiment, experience, and workload impact the healthiness of developer communities by differentiating the feature values related to smelly and non-smelly predictions. Furthermore, we attempt to make suggestions to practitioners based on the discovered relationships.

Approach. We investigate the predictive power (i.e., the absolute value of SHAP feature importance) of each feature to reveal the contribution of every feature to the five prediction classes of our prediction model. Furthermore, we assess the statistical relationships between features and their prediction results for the correct cases in Smelly Developer prediction. Such relationships include: 1) the correlation between the features' values and the Shapley values of the features calculated by SHAP, 2) the significance of differences in distribution as well as the effect sizes of feature values that produce positive and negative Shapley values, and 3) mean and variance of the features that produce positive and negative Shapley values.

*RQ*3. What is the accuracy that we can forecast the number of smelly developers in a community?

Motivation. In RQ3, we intend to investigate whether we can provide decision support for community shepherds to monitor community healthiness and to refactor community smells based on the number of developers affected.

Approach. We transform our dataset to reflect the number of smelly developers in every analysis window. Then, we exploit three types of time-sensitive prediction approaches, i.e., the time-wise classifier based approach which unifies the prediction results in RQ1, the time-wise regressor based approach built upon the mean and summation values of the features, and the time series based forecasting approaches.

*RQ*4. Can we explain the behavior of the best-performed models of RQ3 in community-wide prediction?

Motivation. Since the community-wide dataset may lose information compared with the individual-wide original one (e.g., only one entity for each analysis window instead of multiple entities concerning all developers), interpreting the results may be challenging. Nevertheless, we can still inspect the best-performed model and analyze the features' importance. Furthermore, we check the consistency with our results in RQ2. We also discuss the reason why our model works or not.

Approach. We evaluate the models if any XAI (eXplainable Artificial Intelligence) technique is applicable. For classifiers and regressors other than Linear Regression, we exploit SHAP and Scott-Knott Effect Size Difference (SK-ESD) to extract and to rank feature importance. For light-weight models such as Linear Regression, we use directly the coefficients of features.

4.2 RQ1: Individual-Wide Prediction Model Definition and Validation

4.2.1 Dependent Variables

In this paragraph, we list the definitions of the five smell-related prediction classes of our model. The three concerning community smells are defined as follows.

Organizational Silo. It refers to the presence of siloed areas of the developer community that do not communicate, except through one or two of their respective members [1], i.e., co-committing developers do not directly communicate at all [22].

Lone Wolf. It reflects co-committing software developers who exhibit uncooperative behavior and mistrust by not appropriately communicating [1], i.e., the collaboration edges that do not have a communication counterpart [22].

Bottleneck. Unique boundary spanners interpose themselves into every interaction across sub-communities [1].

An introduction of the three community smells is available in Subsection 2.1. Despite the three smells, we also consider two related classes.

Smelly Developer [9]. Smelly developers are developers affected by any of the three above-mentioned community smells. Since community smells are all related to the quality of collaboration and communication among community members, we intend to provide community shepherds with a higher-level observation of the community healthiness.

Smelly Quitter. Smelly quitters are developers who were affected by community smell in the previous analysis window and left the community (i.e., not present in the current analysis window) [9]. Our motivation to involve Smelly Quitter prediction is 2-fold. First, developer turnover is a major issue to consider when monitoring the activeness and healthiness of OSS developer

communities since they are established based on volunteered cooperations [12, 14]. Meanwhile, turnover is also proved related to software quality [29, 30]. Second, literature reported that negative sentiment may cause the developers to "destroy codebase" [36], and to "quit over mistreatment" [37]. Similarly, we assume it may also lead to the discharge of the community, and we intend to explore if they could help predict smelly quitters.

4.2.2 Independent Variables

We use features in Table 2 as independent variables. The valence, arousal, and dominance features are measured by extending lexicons [16] in SentiStrength [38]. The positive and negative sentiments are also measured using the original lexicons in SentiStrength. The values of the features are calculated by the mean occurrences of the sentiment-related lexicons of developers'

Table 2. Features Extracted from the Developer Sentiment Dataset, Codeface4Smells-defined Metrics, and Software VCS

Name	Type	Abbreviation	Description
Mean valence	Sentiments	VAL	Mean intensity of valence, i.e., how a developer enjoys a situation
Mean arousal	Sentiments	ARO	Mean intensity of arousal, i.e., increased alertness
Mean dominance	Sentiments	DOM	Mean intensity of the extent that a developer was feeling in control
Mean positive sentiment	Sentiments	POS	Mean intensity of all sentiment values lower than 0
Mean negative sentiment	Sentiments	NEG	Mean intensity of all sentiment values greater than 0
Mean sadness	Sentiments	SAD	Mean intensity of all sadness expressions
Mean anger	Sentiments	ANG	Mean intensity of all angry expressions
Mean love	Sentiments	LOV	Mean intensity of all love expressions
Mean joy	Sentiments	JOY	Mean intensity of all joyful expressions
Politeness proportion	Sentiments	POL	Proportion of polite expressions in all commentary sentences of a developer
Indicative GM proportion	Sentiments	IND	Proportion of sentences that express the fact or belief
Imperative GM proportion	Sentiments	IMP	Proportion of sentences that express the command or warning
Conditional GM proportion	Sentiments	CON	Proportion of sentences in the form like would, may, or will
Subjunctive GM proportion	Sentiments	SUB	Proportion of sentences in the form like wish or were
Mean degree of modality	Sentiments	MOD	Degree of uncertainty of a sentence
Total sentences commented	Experience	EXP_SEN	Number of total sentences commented in the JIRA ITS
Core developer experience	Experience	EXP_COR	Number of total windows that a developer acted as a core developer
Mailing list experience	Experience	EXP_ML	Number of total windows that a developer commented in the mailing list
Development experience	Experience	EXP_DEV	Number of total windows that a developer made code commits
Sponsored experience	Experience	EXP_SPO	Number of total windows that a developer committed only in working hours [9]
Organizational silo exp.	Experience	EXP_OS	Number of total windows that a developer was influenced by Organizational Silo
Lone Wolf experience	Experience	EXP_LW	Number of total windows that a developer was influenced by Lone Wolf
Bottleneck experience	Experience	EXP_BN	Number of total windows that a developer was influenced by Bottleneck
Smelly experience	Experience	EXP_S	Number of total windows that a developer was influenced by any community smells
Smelly Quitter experience	Experience	EXP_QUIT	Number of total windows that a smelly developer quitted the community [9]
Average commit size	Process	AVGCS	Mean number of modified files of all code commits made by a developer
Code churn	Process	CHURN	Number of changed lines of all code commits made by a developer
Number of changed files	Process	NF	Number of files changed by a developer
Number of commits	Process	NC	Number of code commits made by a developer
Number of bug fixes	Process	NBF	Number of bug-fixing commits made by a developer
Number of refactoring	Process	NR	Number of refactoring commits made by a developer
Mean co-committers	Process	MC	Mean number of committers work on the files modified by a developer
Dev. scattering changes	Process	DSC	Mean number of distinct sub-systems (packages) modified by a developer
Change entropy	Process	CE	Average Shannon's Entropy of the number of changes of the modified files
Code ownership	Process	OWN	Number of commits of a developer over total commits for all changed files

comments, and the range of the values is between [−1, 1]. The sadness, anger, love, and joy features are manually labeled binary values. The politeness features are presented by binary values measured using Danescu-Niculescu-Mizil *et al.*'s tool [39]. The four grammatical moods (GM) and the modality feature range in [−1, 1], and they are measured by auxiliary verbs and adverbs [40]. The examples of the sentimental expressions extracted from the developer sentiment dataset are available in our online appendix⑤. The experience features reflect the developers' former working states. The smelly features demonstrate the count of analysis windows in which developers are smelly. The process features measure developers' code commit activities in the previous analysis windows [3].

To comply with the features' definition in Table 2, we make necessary modifications when extracting raw sentiment features. Similar to [41], we divide the sentiment feature by zero and construct two positive and negative sentiment features, i.e., POS and NEG. We do not include an impoliteness proportional feature because the politeness labels available in the dataset are either polite or impolite, which will result in a high correlation if we involve both. We ignore the confidence coefficients of the politeness scores since Ortu *et al.* [20] already discarded the scores with the coefficients less than a conventional threshold (0.5) [25]. The dataset also contains a mood column whose proper meaning was not clearly described in the original paper [26]. We look into the documentation of the detection tool [40] and confirm this feature measures the GM of a sentence, and the results are mapped into four classes, i.e., indicative, imperative, conditional, and subjunctive. In the dataset, the four classes are presented in four values, namely 0, 1, 2, and 3. We map the mood attribute into four proportional features to measure the developers' characteristics of expression.

Furthermore, we introduce 10 experience features derived from the JIRA sentiment dataset and detection results of CODEFACE4SMELLS. Our motivation to introduce experience features comes from multiple community smell empirical studies [10, 11, 23], as they found that practitioners believed that developers' experience may make a community more prone to be affected by smells. Meanwhile, related work in bug report re-opening [17] found that a problematic component is more likely to be affected again, which makes us assume developers affected by community smells in the past may be more prone to smells in the future.

Additionally, we also involve and transform 10 process metrics that have been proved effective for code smell detection [3] and were commonly applied in software defect prediction [29, 30]. Different from socio-technical features measuring developer centrality such as EXP_COR, process metrics features measure the workload of developers and their collaborators. Since community smell is related to collaboration, we believe process metrics reflecting the states of collaboration would be a useful source of information for prediction.

4.2.3 *Data Balancing and Feature Selection*

Community smell datasets are highly imbalanced [9]. In our dataset, smelly developers account for 4.80% of the overall developer population, which may hinder model performance. Therefore, we preprocess our data with SMOTE, Random Oversampling, and Random Undersampling strategies if they lead to better performance. We also address the potential multicollinearity problem by removing the correlated features as prior researches suggested [9].

4.2.4 *Performance Assessment and Validation*

We build models separately in cross-project, within-project, and time-wise validation scenarios (see Fig.4). Afterwards, we compute performance metrics including precision, recall, *F*-measure, and AUC-ROC to pick the best classifier in the three scenarios. Apart from the traditional metrics such as *F*-measure, we also involve AUC-ROC because it is insensitive to imbalanced data [9]. The definitions of the metrics are listed in (1)–(4):

$$Precision = \frac{TP}{TP + FP}, \tag{1}$$

$$TPR = Recall = \frac{TP}{TP + FN}, \tag{2}$$

$$F\text{-}measure = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{3}$$

$$FPR = \frac{TN}{FP + TN}, \tag{4}$$

where TP is for true positive (positive samples predicted as positive), FN is for false negative (positive samples falsely predicted as negative), TN is for true negative, and FP is for false positive. AUC-ROC is calculated as the area under the TPR-FPR curve.

In order to rank the performance with effect size awareness, we also involve the SK-ESD [42] test. The

⑤https://github.com/SORD-src/JCST_Replication, Jan. 2022.

Scott-Knott test [43] is a statistical measure to compare and differentiate model performance using a hierarchical clustering approach to group the means of assessment metrics, e.g., F-measures of multiple models. The Scott-Knott test assumes input data to be normally distributed, and thus it may be ineffective for non-normally distributed data. The SK-ESD test is an enhanced version of the Scott-Knott test that corrects the non-normal distribution of the input to make it comply with the requirements to perform the Scott-Knott test. Meanwhile, it uses Cliff's Delta as an effect size measure to merge groups having negligible effect sizes. We use the original R implementation of Tantithamthavorn *et al.* [42].

For cross-project validation, we apply a strategy similar to [9], which is a project-wide Leave-One-Out Cross-Validation (LOOCV), i.e., we use one out of 12 projects as the test set, and the others as the training set to build our model. We merge the developers' features in the training set regardless of the projects they were working on. Notably, the same developers appearing in multiple projects are treated as different ones. Such a process is performed 12 times.

For within-project validation, we apply LOOCV in each project, i.e., we build models separately for each project, and we use a developer's data in an analysis window for testing and the others for training. Since LOOCV is proved reliable [42] for software engineering, we still acknowledge the potential flaw of such a method, as it may use the future data to predict the earlier targets [29]. To this end, we also involve time-wise validation to eliminate the drawbacks of LOOCV.

For time-wise validation, we group the data in two analysis windows, and each group contains developers' data in six months. Due to the characteristics of version iteration [29], we train our models using the data in the j-th and the $(j+1)$-th analysis windows, and validate the model using the data in the $(j+3)$-th and the $(j+4)$-th windows. For each iteration, we move forward by one analysis window, i.e., the next iteration for the above-mentioned example will be training our model based on the $(j+1)$-th and the $(j+2)$-th windows, and validating in the $(j+4)$-th and the $(j+5)$-th windows. Assuming that we have K windows of data, such a process is performed for $K-4$ times in each project.

4.2.5 *Training Machine Learners*

We apply the SCIKIT-LEARN package [44] to train machine learners using multiple classifiers that have been used in prior studies [3, 9, 45], including RF, Decision Tree (DT), Support Vector Machine (SVM), Multilayer Perceptron (MLP), Adaboost (ADA), Naive-Bayes (NB), and Logistic Regression (LogR). As related work suggested [3], instead of using default settings, we configure the hyper-parameters of the classifiers by exploiting Exhaustive Grid Search with a 10-fold cross-validation strategy to calculate the performance of every combination of parameters.

4.3 RQ2: Predictive Power of Features for RQ1

To answer this RQ, we expect to find statistical significance to explain the extent of predictive power that each independent variable contributes to the best-performed classifier in RQ1.

Since complex machine learning models are difficult to understand, their built-in feature importance results (i.e., classifier-specific importance [34]) are also hard to interpret. However, the explanation of models is essential for practitioners and developers to make decisions about building models and perform SQA activities [46]. In response, Jiarpakdee *et al.* [32, 33] explained the behavior of complex models using an interpretable approximation of the original model.

We apply the SHAP algorithm, which has been studied empirically in a recent software engineering paper [34] validating the stability of feature importance methods and predicting software defects. SHAP measures the contribution of a feature value to the difference between the actual local prediction and the global mean prediction [35] to distribute the credit for a classifier's output among its features [34] using the game-theory based Shapley values [47]. For each instance in the training set, SHAP transforms the features of the instance into a space of simplified binary features as input. Afterwards, SHAP builds the model g for explanation defined as a linear function of binary values, more specifically in (5):

$$g(\boldsymbol{z}) = \phi_0 + \sum_{i=1}^{M} \phi_i z_i, \tag{5}$$

where $\boldsymbol{z} \in \{0,1\}^M$ is the coalition vector (also known as simplified features [47]), and M is the maximum size of the coalition vector (i.e., the number of simplified features). Specifically, z_i is the i-th binary value in $\boldsymbol{z}$, where $z_i = 1$ means the corresponding feature is included in the coalition, and $z_i = 0$ indicates the feature is absent from the coalition. ϕ_0 is the average prediction value of the model, and ϕ_i is the Shapley value

of the i-th feature. A larger positive ϕ_i indicates a greater impact of the i-th feature to the positive prediction result of the model. Note that $|\phi_i|$ are SHAP feature importance scores that are guaranteed in theory to be locally, consistently, and additively accurate for each data point [34]. We use the PYTHON implementation of SHAP [35] in our study. Meanwhile, we also exploit SK-ESD to rank feature importance. Both SHAP and SK-ESD algorithms are executed on each prediction class independently. We report the results from all three validation scenarios.

Then, we take a closer look at the relationships of features' distribution and the prediction results for Smelly Developer, which is positive ($\phi_i > 0$) for smelly and negative ($\phi_i < 0$) for non-smelly. Since our data are not normally distributed, we apply the non-parametric Spearman's Rank Correlation Test [48] to measure the correlation between the features' values and their ϕ_i values calculated by SHAP. Given two sets of values equal in length, the test produces a correlation coefficient ρ with a p-value to measure the significance level. The p-value is the probability of obtaining a test statistic result at least as extreme as the result that was observed, and the correlation is statically significant if p-value < 0.05. We consider the rank of correlation is trivial if $|\ \rho\ | < 0.10$, low if $0.10 \leqslant |\ \rho\ | < 0.30$, moderate if $0.30 \leqslant |\ \rho\ | < 0.50$, high if $0.50 \leqslant |\ \rho\ | < 0.70$, very high if $0.70 \leqslant |\ \rho\ | < 0.90$, and perfect if $|\ \rho\ | \geqslant 0.90$ [49]. We also apply the Wilcoxon Ranksum Test measuring statistical significance [48] to analyze the significance of the difference in the distributions of the features. Wilcoxon Ranksum Test also produces a p-value, and we use p-value < 0.05 as an indicator of statistical significance. Meanwhile, we calculate Cliff's Delta (δ) to measure the effect size (i.e., the extent of the difference) for each pair of feature values that produce positive and negative ϕ_i values. The effect size is negligible if $|\ \delta\ | < 0.147$, small if $0.147 \leqslant |\ \delta\ | < 0.33$, medium if $0.33 \leqslant |\ \delta\ | < 0.474$, and large if $|\ \delta\ | \geqslant 0.474$. Additionally, we also report the mean and variance of the features leading to positive and negative prediction results.

4.4 RQ3–RQ4: Community-Wide Smelly Developer Forecasting

4.4.1 *Performance Assessment and Validation*

In this subsection, we intend to demonstrate the performance of the model to predict the number of smelly developers in the last four analysis windows (i.e., 12 months). We forecast 12 months because an empirical study [30] reported the commit-level defect prediction model will lose its effectiveness without being retrained after one year. For classifiers and regressors, since forecasting is time-sensitive, we use settings similar to the time-wise prediction in RQ1. We generate a new model for each predicted data, i.e., our prediction is 4-fold. Assuming that we have K windows of data, for each fold i ($i > 0$), we use the data of the first to the $(K-i-4)$-th analysis window as the training set, and predict smelly developers from the $(K-i-2)$-th to the $(K-i+1)$-th analysis windows. Note that the $(K-i-3)$-th window is skipped as [29] suggested. Finally, we calculate the summation of the predicted smelly developers as the result. For time series based approaches, we use the data from the first window to the $(K-i-3)$-th window as the training set to predict the last four analysis windows because such approaches support forecasting of multiple time windows by nature.

To demonstrate the performance of the models, we use three metrics for regression assessment including coefficient of determination (i.e., R^2), root mean squared error (RMSE), and mean absolute error (MAE), whose definitions are listed below in (6)–(8).

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(A_i - F_i)^2}{\sum_{i=1}^{n}(A_i - \bar{A}_i)^2}, \tag{6}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(A_i - F_i)^2}{N}}, \tag{7}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|A_i - F_i|, \tag{8}$$

where $A_i, \bar{A}_i, F_i$ are the actual, the mean, and the forecasted number of smelly developers respectively. Lower RMSE and MAE values indicate better performance. On the contrary, the model with a larger R^2 value is considered to be the best model. The RMSE and MAE values range from 0 to ∞. R^2 measures the degree of relationship between the forecasted and the real data, and it ranges from 0 to 1. If the result of any model is extremely irrelevant with the test set, the R^2 value may be negative. Since we are not interested in how badly the models perform [50], we treat negative values as zeros. We calculate the three assessment metrics for each project, and we use the mean value as the performance of each model.

4.4.2 *Training Fine-Grained Classifier*

The prediction in this subsection is performed in the individual-wide scenario using the best fine-grained approach in RQ1. After we generate the prediction results

for smelly developers, we perform a summation of the classifiers' outcomes to generate the result.

4.4.3 Training Coarse-Grained Regressors

We use regressors to predict the number of smelly developers based on coarse-grained features of every analysis window. We create two datasets using different approaches, i.e., mean and summation for every feature. Then, we use 10 regressors which were usually applied in SQA tasks [51], including tree-based regressors, linear regressors, and other state-of-the-arts, i.e., Gradient Boosting Regressor (GBR), Random Forest Regressor (RFR), Decision Tree Regressor (DTR), K-Neighbors Regressor (KNR), LogR, Linear Regressor (LR), Bayesian Ridge Regressor (BRR), Gaussian Process Regressor (GPR), Neural Network Based MLP Regressor (NNR), and Stochastic Gradient Descent Regressor (SDGR). We also tune hyper-parameters according to [51] and perform feature selections. However, data balancing techniques are not used since they are only applicable to classification tasks.

4.4.4 Training Time Series Based Models

ARIMA (AutoRegressive Integrated Moving Average) is a commonly used time series based prediction model in software evolution prediction [52] since it can cope with non-stationary series. We tune the three most-sensitive hyper-parameters [52] of ARIMA (i.e., p, d, and q) in a range of [0, 15]. In addition, we involve other approaches applied in SQA-related forecasting tasks [53] including Prophet, Exponential Smoothing (ExpS), and Theta which are capable of capturing trends and seasonal patterns of time series. We use the implementation of Python Darts package⑥. Such methods perform predictions directly based on the previous smelly developers, and they do not require the input of any additional features. Apart from the univariate approaches mentioned above, we also try Temporal Convolutional Network (TCN) using multiple inputs of other time series as covariates, i.e., we specifically transform other features to covariates for TCN.

4.4.5 Explaining the Models

We exploit a process similar to RQ2 to explain regressors and classifiers except for lightweight models such as LR. The feature importance of LR could be determined by the coefficients of features. However, since features are not available for univariate time series based models, we may not be able to explain them.

5 Results and Discussion

In this section, we answer the proposed research questions by demonstrating and discussing the results of our experiment. We also draw conclusions and demonstrate our findings.

5.1 RQ1: Individual-Wide Model Performance

First, we assess the correlations of our features, and the correlation heat-map is available in our online appendix⑦. Results show the VAD features correlate with each other ($\rho > 0.7$). Although we find removing any two of them does not cause a significant change in the classifier's performance ($< 2\%$), we still exclude VAL and DOM to avoid multicollinearity. We also exclude CON, EXP_DEV, EXP_ML, EXP_LW, EXP_OS, EXP_S, NBF, NF, and CE for the same reason. Consequently, we remove 11 features out of 35 original ones.

Then, we train multiple classifiers on the dataset, and RF is the best-performed one. The performance of the trained classifiers is available in our online appendix⑦. We present the weighted average performance of RF models in within-project, time-wise, and cross-project validation scenarios in Table 3. Moreover, the boxplots of the RF model performance as well as the settings including parameters and data balancing strategies applied are available in our online appendix⑦.

Our model shows good overall performance in all three scenarios, e.g., F-measure > 0.73 in all cases with acceptable AUC-ROC > 0.69.

To compare the effectiveness of the approach in this paper and its preliminary conference version [21], we demonstrate the performance using features in the conference paper in the context of time-sensitive data generation and validation approaches. The comparisons are available in the D.-O. column and the Full column in Table 3. We can conclude that since using only experience and sentiment features already produces acceptable results in four classes except for Smelly Quitter, involving process metrics significantly improves the performance of the model by 3%–17% in terms of AUC-ROC. In Smelly Quitter prediction, our model significantly improves AUC-ROC performance from 19% to

⑥https://github.com/unit8co/darts, Nov. 2021.

⑦https://github.com/SORD-src/JCST_Replication, Jan. 2022.

Table 3. Performance of Individual-Wide RF Models

Prediction Class	Validation/Feature	Precision		Recall		*F*-Measure		AUC-ROC	
		D.-O.	Full	D.-O.	Full	D.-O.	Full	D.-O.	Full
Bottleneck	Within-Project	0.83	**0.85**	0.83	**0.85**	0.83	**0.85**	0.79	**0.82**
	Time-Wise	0.72	**0.79**	0.73	**0.78**	0.72	**0.78**	0.66	**0.74**
	Cross-Project	0.75	**0.79**	**0.77**	0.71	0.73	0.73	0.64	**0.74**
Lone Wolf	Within-Project	0.88	**0.92**	0.88	**0.92**	0.88	**0.92**	0.81	**0.87**
	Time-Wise	0.87	0.87	**0.87**	0.85	0.86	0.86	0.72	**0.81**
	Cross-Project	0.80	**0.88**	0.81	**0.86**	0.80	**0.86**	0.68	**0.82**
Organizational Silo	Within-Project	0.88	**0.92**	0.88	**0.92**	0.88	**0.92**	0.83	**0.88**
	Time-Wise	0.87	0.87	**0.87**	0.85	0.86	0.86	0.71	**0.81**
	Cross-Project	0.79	**0.89**	0.79	**0.87**	0.77	**0.87**	0.66	**0.83**
Smelly Developer	Within-Project	0.81	**0.85**	0.81	**0.85**	0.81	**0.85**	0.79	**0.83**
	Time-Wise	0.80	**0.82**	**0.80**	0.79	**0.80**	0.79	0.75	**0.77**
	Cross-Project	0.79	**0.82**	0.75	**0.83**	0.76	**0.82**	0.74	**0.78**
Smelly Quitter	Within-Project	0.95	**0.98**	**0.96**	0.80	**0.96**	0.87	0.49	**0.82**
	Time-Wise	**0.94**	0.89	0.95	**0.96**	**0.94**	0.92	0.50	**0.69**
	Cross-Project	0.96	**0.98**	**0.98**	0.78	**0.97**	0.85	0.50	**0.87**

Note: D.-O. refers to developer-oriented features (i.e., experience and sentiment features) applied in our preliminary conference version [21], while Full refers to using all features available in this paper. Better performance is in bold.

37%, showing that process metrics are vital for Smelly Quitter prediction, and sentiment features are not necessarily related to work withdrawal. The features in the preliminary version achieve better *F*-measure performance because *F*-measure is biased for extremely imbalanced data, i.e., the classifier predicts almost all developers as non-smelly quitters, which accounts for the vast majority of the population. Consequently, the AUC-ROCs of Smelly Quitter prediction are around 0.5, indicating that it is no better than random guessing. Thus, we can conclude that the approach in this paper is better than the ones transformed from the conference paper [21] in terms of AUC-ROC and *F*-measure.

However, we suggest not to directly compare the performance of our paper and the conference paper [21] because the latter uses a time-insensitive validation and data generation approach which results in overestimated performance. Comparing the performance of the model in the conference paper [21] with the performance in the D.-O. column of Table 3, we can see the overestimation is 11% and 4% on average in terms of AUC-ROC and *F*-measure respectively. The conference paper [21] uses data in the analysis windows later than the predicted analysis window, which causes a potential flaw. In practical scenarios, the future data would not be acknowledged in the present model. Otherwise, community smells can be detected precisely by CodeFace4Smells, and thus we actually do not need any prediction. Nevertheless, compared with the overestimated results, this paper still improves the performance up to 11% in all cases in terms of AUC-ROC, while the difference in *F*-measure is −5%–1%. In consideration of the overestimation, our model is significantly superior in AUC-ROC with a similar performance in *F*-measure.

Finding 1. Our model could predict the occurrence of the concerning community smells on developers in most cases. It achieves mean *F*-measures ranging from 73% to 92% in five prediction classes. Meanwhile, the involvement of the process metrics improves the performance based on the features used in the conference paper [21]. Specifically, it improves the AUC-ROC of Smelly Quitter prediction by 19%–37%.

5.2 RQ2: Features' Importance and Distribution

Table 4 lists the features' importance of the correctly predicted cases of all five prediction classes in descendant order. The importance values measure the contributions of features to the performance. The boxplots demonstrating the distribution of feature importance scores are available online[⑧].

[⑧] https://github.com/SORD-src/JCST_Replication, Jan. 2022.

Table 4. Ranking and Mean SHAP Importance of Features

Feature	Mean	Within-P	Time-W	Cross-P
EXP_SEN	0.028	1	1	1
EXP_BN	0.020	2	1	1
AVGCS	0.018	1	2	2
CHURN	0.017	1	2	2
NC	0.016	1	2	2
OWN	0.012	1	3	7
MC	0.012	2	3	3
EXP_COR	0.012	2	3	3
JOY	0.011	2	3	4
ANG	0.010	2	4	4
DSC	0.010	2	3	5
NEG	0.009	2	3	5
IMP	0.009	2	4	4
SUB	0.008	2	5	6
ARO	0.008	3	4	7
POS	0.007	3	5	7
SAD	0.007	3	5	7
LOV	0.007	3	5	7
POL	0.006	3	5	7
IND	0.006	3	5	8
MOD	0.006	3	5	8
EXP_QUIT	0.002	4	6	9
NR	0.001	5	7	10
EXP_SPO	0.000	5	7	10

Note: Within-P refers to the ranking in within-project prediction. Similarly, Time-W refers to time-wise, and Cross-P refers to cross-project. The Mean column demonstrates the mean of the absolute value of SHAP feature importance.

Practitioners mainly focus on the top-1 and the top-3 features when using feature importance algorithm for interpretation [33]. The only top-ranked metric in all three validations is EXP_SEN. The metrics ranked in top-3 in all three validation scenarios include experience features (EXP_SEN, EXP_BN, and EXP_COR) and process metrics (AVGCS, CHURN, NC, and MC). Such a trend reflects that developers' experience and activities of communication and collaboration are a determining aspect of community smell occurrence, and the centrality and the discussion activeness of developers are stronger predictors than several sentiments. In particular, the number of commented sentences and the experience of acting as core developers are the features having the most predictive power. A related study [9] observed similar circumstances that communicability and the number of core developers were among the top predictors of community smells. Moreover, recent work [54] also reported that the number of commits is related to community smells' occurrence.

The rankings of the sentiment features' contributions are lower than the ones of the experience and process metrics. However, their contributions are average and moderate. For example, all 12 sentiment features appear at least once in the top-3 rankings.

Furthermore, we assess the feature importance in each validation scenario of every smell. The detailed feature importance data are available in our online appendix[9]. In general, the results follow a trend similar to Table 4, i.e., experience features (e.g., EXP_SEN and EXP_BN) and process metrics (e.g., AVGCS, CHURN, OWN, NC, MC) are the most determining features. Specifically, experience features are the top contributors of Bottleneck and Smelly Developer predictions, while both experience and process features are the most important ones for Organizational Silo and Lone Wolf predictions. For Smelly Quitter, all top-ranked features are process metrics.

To assess the features' interactions with the model behavior, we further investigate the impact of feature values on prediction results. Fig. 5 depicts a SHAP beeswarm plot displaying the feature values' impact on the correct prediction cases of Smelly Developer. Darker (lighter) points represent higher (lower) feature values. Meanwhile, data points in the right (left) represent higher (lower) ϕ_i values that lead to the prediction results of smelly (non-smelly) [46]. The SK-ESD ranking of each feature's importance calculated based on their $|\phi_i|$ values is also presented in the Y-axis. To explain Fig.5 in detail, we demonstrate the statistical relationships between the values of features and their ϕ_i values in Table 5. We present Spearman's ρ and Cliff's δ if the results are statistically significant, and we mark "–" for insignificant results.

For most experience and process metrics (e.g., EXP_SEN, EXP_BN, EXP_COR, CHURN, AVGCS, NC, DSC, and OWN), we can see that they follow an easily recognizable trend. In terms of process metrics, developers with heavier workload (e.g., having larger commit sizes, more modifications and changes, more sub-systems to work on, and more code ownership) tend to be predicted as smelly. In terms of experience, lower EXP_SEN leads to smelly prediction, indicating that the lack of communication may decrease community healthiness, which was also found in a prior research [11] suggesting more communication is an effective solution to community smells. In contrast, higher core experience and more smelly experience such as Bottleneck lead to smelly prediction. Such a trend is also quantifiably presented in Table 5. For example,

[9] https://github.com/SORD-src/JCST_Replication, Jan. 2022.

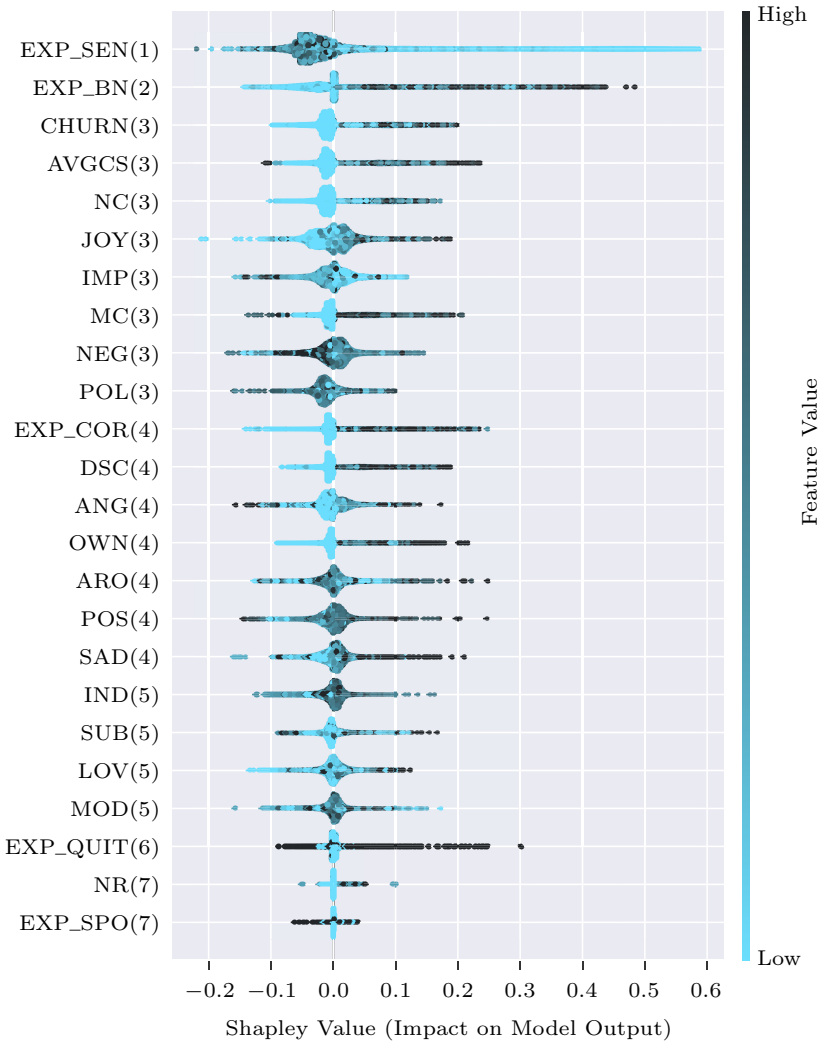


Fig.5. Feature values' impact on the correct prediction cases of model.

mean EXP_SEN values that lead to non-smelly prediction ($\phi_i < 0$) are larger than the ones that lead to smelly prediction, and they are significantly different in distribution. Moreover, the overall EXP_SEN values are moderately correlated with the SHAP ϕ_i values of the EXP_SEN feature.

The relationship between sentiment features and prediction results is not so clear as the other features, which cannot be easily concluded from Fig.5. To present the characteristics of sentiment features more clearly, we also depict the distribution of sentiment features that lead to smelly and non-smelly predictions in Fig.6 using an enhanced version of box-plot called boxen-plot [55]. Boxen-plot is more capable of displaying tails of large-sampled data, as it cuts the data into more quantiles.

In terms of the predictive power of GM, the IMP feature (i.e., imperative expressions of commands and warnings) is one of the top-ranked sentiment features. From experience, we assume instructive expressions are more likely to be serious and impolite, which would become an obstacle for cooperation. However, results show the distribution of the proportion of IMP is significantly different for related smelly and non-smelly predictions with a large effect size. In terms of feature values, the results also reveal that the IMP feature values deriving non-smelly predictions are 47% greater than the ones that derive smelly predictions. Moreover, another feature about certainty, i.e., indicative GM (IND), is also showing that using less indicative expressions yields the smelly prediction results. Hence, we conclude that ensuring certainty is vital to developers' communication and collaboration quality.

Since there exist several studies suggesting the relationship between software defect-proneness and politeness of developers [25, 26], we expect to find a significant

Table 5. Results of Statistical Analysis

Feature	Spearman's Rho		Cliff's Delta		Mean		Variance	
	ρ	Rank	δ	Effect Size	Smelly	Non-Smelly	Smelly	Non-Smelly
EXP_SEN	−0.47	++	−0.65	L	23.00	52.18	979.89	1 231.87
EXP_BN	0.60	+++	0.53	L	6.09	1.95	33.56	1.83
CHURN	0.48	++	0.69	L	26 267.71	2 903.22	1.80E+10	2.90E+08
AVGCS	0.28	+	0.41	M	9.15	4.77	1 789.84	88.28
NC	0.52	+++	0.78	L	48.95	4.92	6 368.45	198.35
JOY	0.13	+	0.05	–	0.11	0.10	0.02	0.01
IMP	0.31	++	−0.37	M	0.07	0.10	0.02	0.01
MC	−0.11	+	0.03	–	6.54	7.29	17.04	37.43
NEG	−0.16	+	−0.22	S	−0.17	−0.15	0.01	0.01
POL	−0.18	+	−0.16	S	0.48	0.53	0.06	0.04
EXP_COR	0.45	++	0.42	M	5.96	3.06	28.70	12.24
DSC	0.26	+	0.62	L	7.23	4.21	8.48	7.30
ANG	0.01	–	−0.07	–	0.04	0.04	0.00	0.00
OWN	0.13	+	0.41	M	0.17	0.09	0.02	0.02
ARO	−0.29	+	−0.31	S	0.96	1.05	0.05	0.05
POS	0.44	++	0.44	M	0.23	0.19	0.00	0.01
SAD	0.18	+	0.10	–	0.32	0.30	0.08	0.08
IND	−0.12	+	−0.16	S	0.64	0.70	0.02	0.04
SUB	–	–	−0.04	–	0.03	0.02	0.01	0.01
LOV	−0.33	++	0.20	S	0.21	0.18	0.04	0.04
MOD	0.13	+	−0.14	–	0.56	0.60	0.03	0.03
EXP_QUIT	0.40	++	–	–	1.29	1.27	0.42	0.35
NR	−0.52	+++	–	–	2.48	2.35	5.57	4.02
EXP_SPO	−0.18	+	–	–	1.74	1.87	3.40	3.61

Note: Effect sizes in {Large, Medium, Small, Negligible} are mapped to {L, M, S, −} respectively. Correlation ranks in {Perfect, High, Moderate, Low, Trivial} are mapped to {++++, +++, ++, +, −} respectively. Features with non-trivial correlation and non-negligible effect sizes are bolded.

predictive power of the politeness feature. Although it is one of the top-ranked sentiment features, the effect size of the difference of the features that produce smelly and non-smelly predictions is relatively small. Nevertheless, it still indicates that smelly and non-smelly developers communicate differently. Moreover, politeness has a low negative correlation with ϕ_i values, revealing that lower politeness is likely to produce a smelly prediction.

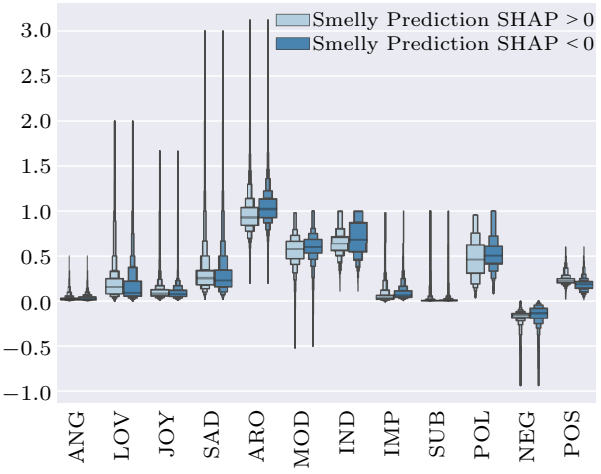


Fig.6. Distribution of sentiment features.

In terms of specific positive emotions, prior research [56] in developer sentiment regarded happiness as a positive factor to development tasks. However, we also notice that [18] reporting a great number of positive comments is related to bug occurrence. Our model shows that more positive emotions (POS) may also indicate community smells' occurrence. In our case, expressions containing more joy and love lead to non-smelly prediction. However, this relationship is relatively weak (e.g., having negligible or low effect size).

Negative emotions have always been an indicator of potential problems in software engineering [57]. Unexpectedly, except for the general negative sentiments (NEG), no notable difference of feature values is found for smelly and non-smelly predictions. However, according to their mean values and ϕ_i values calculated by SHAP, developers with negative sentiments are more likely to become smelly. Still, the effect sizes of such a trend are small or trivial. In case studies, we also recognize that neutral comments with test case results attached were very likely to be recognized as negative comments since the description of the results may be in a strict tone, which introduce noises into our dataset. However, simply discarding these comments is inappropriate, and recognizing the appearance of such con-

tent is also a challenge. Although sentiment analysis on developers has made significant progress, improving sentiment recognition in domain-specific comments still remains a problem for both natural language processing and software analytics communities to solve [58].

Due to the multi-facet and complex nature of sentiment and development activities, the difficulties in interpretations occurred frequently in developer sentiment analysis [12, 16, 17]. Similar to the three above-mentioned studies, our attempt to empirically explain why the model leads to conflicting or small effect-size observations in terms of the impact of positive and negative sentiments. The difficulties in interpreting the sentiment part of the results shed light on the necessity of a deeper understanding of developer sentiment and their impact on the software community through a qualitative and quantitative study. We believe the lack of comprehension of the task context [4] of developers' emotional expressions in different types of communities [28] and tasks is the reason why we cannot interpret the conflicting observations. It is necessary to improve and reshape the framework of comprehending developers' perception [3] as well as their task context [4] in both social and technical aspects.

Practically, we suggest developers should make more communications in a straightforward and polite way, especially for the busiest developers who have already been affected by community smells.

The above-mentioned observation has some conflict with the preliminary version of our paper [21] in terms of the overall rankings among sentiment and developer experience features due to the followings. 1) In this paper, we change the original feature importance algorithm (i.e., Information Gain) to SHAP which is less sensitive to feature interaction [34], more adaptive to human intuition [35], and provide the explanation for every local prediction instance, allowing us to focus only on the correctly predicted cases. 2) The two algorithms focus on different aspects of feature importance. The information Gain algorithm calculates the reduction in Shannon's Entropy for prediction results globally after involving each feature [3]. SHAP measures the impact of each feature on prediction results locally by comparing the output of every local prediction with the mean prediction to explain feature importance, i.e., its global explanation is constructed by local explanations. 3) The preliminary version of our paper [21] is not time-sensitive. To figure out the extent of agreement between SHAP and Information Gain, we also evaluate the gain ratio of the transformed time-wise features (i.e., using the model based on D.-O. features in Table 3), and we discover that both sentiment (e.g., ARO, MOD, POS, NEG, IND, IMP) and EXP_COR appear at the first rank. Apparently, the two algorithms only agree on EXP_COR in terms of the top-ranked feature. The inconsistency of the feature importance algorithm has been discussed in recent studies [32–34], and related work concludes model agnostic methodologies such as SHAP are more stable and reliable [34].

Moreover, this paper outlines the significant importance of process metrics compared with sentiment and experience features, especially for Smelly Quitter prediction. Since sentiment and experience features applied in our preliminary conference paper [21] reflect the working states of developers, the new process metrics provide the quantified workload data, which are able to measure the two major aspects that lead to community smells [10, 13, 23], i.e., productivity issues such as burnout [16] as well as the importance and diversity of developers [13]. As a result, a great improvement in prediction performance is achieved after involving such features. Nevertheless, the features presented in our conference version [21] still manage to improve the models' performance, and they also contribute to predictive power. Based on our findings, we believe direct and indirect data to measure developers' workload and working states are both essential for prediction.

Finding 2. In terms of community smells' occurrence on developers, experience features (e.g., sentences commented) and process metrics (e.g., average commit size) are stronger predictors than the sentimental ones. Moreover, the model behavior shows 1) the less communicated developers and the core developers with higher workload are more likely to be affected by community smells, 2) the less polite developers and developers using less certain expressions such as statements, instructions, and warnings tend to be affected by community smells, and 3) the developers frequently affected by community smells are more likely to become smelly again. To ensure community healthiness, we suggest developers should foster more frequent communication in a straightforward and polite way.

5.3 RQ3: Community-Wide Model Performance

In the data processing phase, we notice a serious extent of feature correlation for community-wide prediction, i.e., most features have a correlation greater than 0.7 with at least one feature, which is hard for

manual processing. Thus, we mitigate correlation by Autospearman [59], a state-of-the-art feature selection method to resolve multicollinearity while preserving performance to a great extent. However, removing correlated features still causes a significant decline in performance since there would be only eight features left in the coarse-grained dataset. Since performance and explainability are both our primary concern, we evaluate the classifiers using the data before and after feature selections. The boxplots demonstrating R^2 performance and the SK-ESD ranks of all evaluated models are available in our online appendix[10].

Table 6 presents the models with the best performance. The models using selected features are marked with _FS in their suffixes. Since $R^2 > 0.5$ could be regarded as good performance and $R^2 > 0.7$ is very good performance [50], we can conclude that the models using selected features can derive good performance, and the performance of models using all features can be close to very good.

To our great astonishment, the simpler LR model is also among the top predictors when using the dataset without any feature selection. Except for this case, tree-based models have the best performance. Among these models, although they are in the same rank in terms of R^2, regressors using the coarse-grained dataset perform significantly better than the classifiers in RQ1 in terms of MAE and RMSE.

Finding 3. The tree-based regression model and the simple LR model using the coarse-grained dataset of summation of feature values can produce good performance (e.g., with R^2 metrics close to 0.7) when forecasting community-wide community smell. However, feature selection would significantly impact the performance. Moreover, classifiers and time series based approaches perform worse than the regressors. In conclusion, we suggest community shepherds use our model built with full features to make community-wide refactoring decisions.

5.4 RQ4: Explaining Community-Wide Regression Models

In this subsection, we explain separately the models before and after feature selection. Note that the results of the model built with the correlated features may be biased because regressors may not be able to determine which feature is more important for prediction when dealing with the correlated ones. For the dataset without feature selection, we explain the LR model since it is a simple model with the top-ranked performance and high interpretability. We choose GBR_FS as our model for the dataset with feature selection since all the candidates are tree-based models explainable by SHAP, and GBR_FS has the best performance. Fig.7 demonstrates the feature importance and their SK-ESD ranks of the two classifiers. Fig.7(a) is a boxplot of LR feature coefficients, and Fig.7(b) is a SHAP violin plot showing features' values, distributions, and their impact on the model behavior. Note that excessive large outliers are not displayed in Fig.7(a) since showing them will cause a sparse distribution of data in the scope and lead to an unreadable graph. We use the violin plot for Fig.7(b) because trends can hardly be interpreted in the beeswarm plot if there exist fewer data.

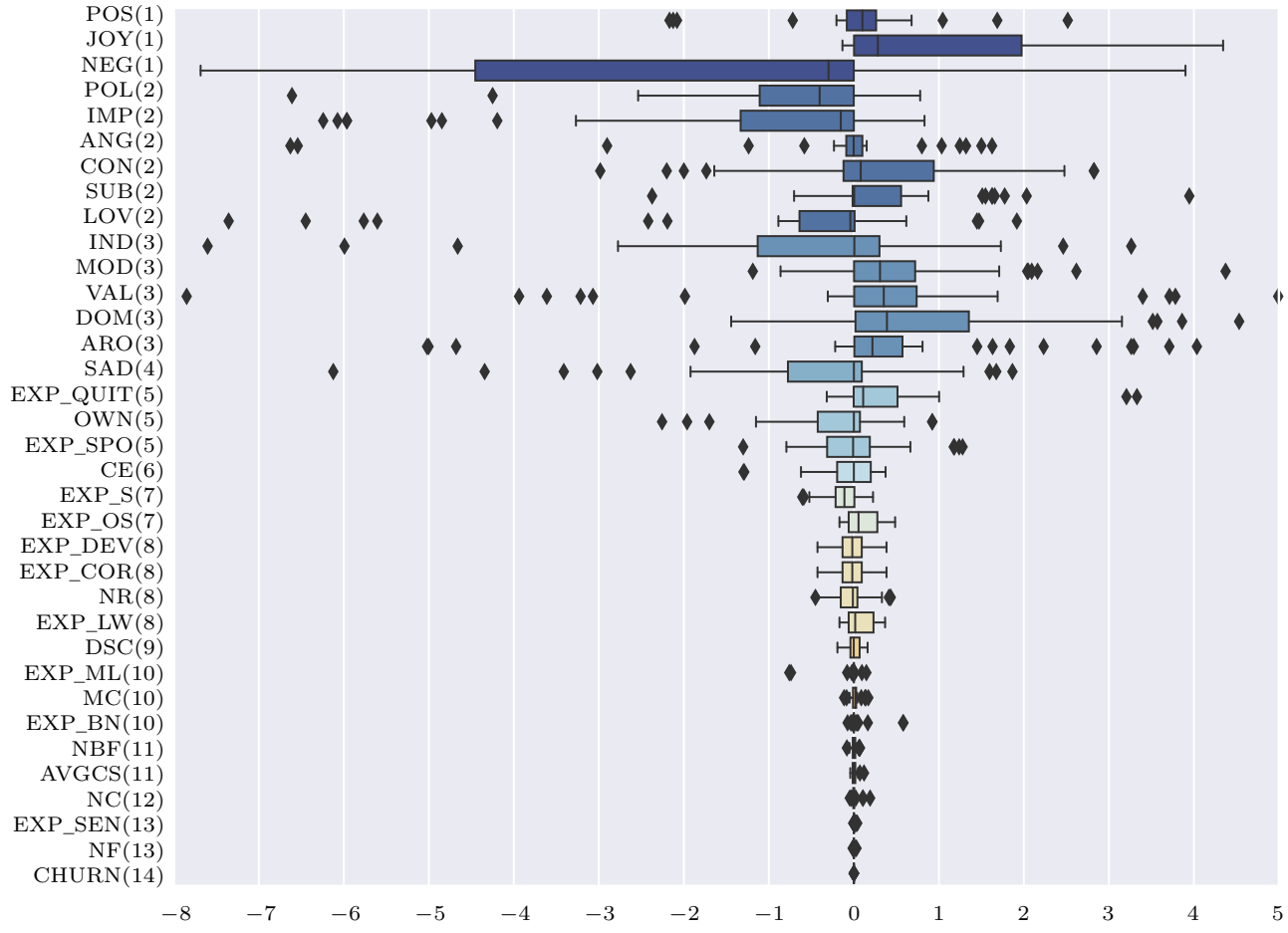
In Fig.7, we can observe a trend that the ranks of sentiment features are higher than those of the other features, which is different from conclusions in RQ2. However, it is hard to comprehend the overall sentiment of a community. Except for this trend, we can barely draw clear conclusions. For example, in Fig.7(b) we can hardly inspect the variation of feature values. Meanwhile, in Fig.7(a), most mean values of coefficients

Table 6. Performance of Community-Wide Models

Classifier	Type	Dataset	MAE	RMSE	R^2	Parameter
GBR	Regressor	Sum	1.94	**2.76**	**0.68**	(190, 1)
RFR	Regressor	Sum	2.19	2.78	**0.68**	(110, 13)
LR	Regressor	Sum	**1.90**	2.84	**0.68**	True
RF	Classifier	Original	15.84	17.67	0.61	(110, 9)
GBR_FS	Regressor	Sum	**3.20**	**4.78**	**0.51**	(100, 13)
RFR_FS	Regressor	Sum	5.68	6.71	0.48	(10, 13)
RF_FS	Classifier	Original	16.89	18.84	**0.51**	(200, 9)

Note: The parameter of the LR model refers to whether the features should be normalized. GBR, RFR and RF share the same important hyper-parameters, i.e., (max_depth, n_estimators). The sum dataset refers to the coarse-grained dataset using the summation of features. The fine-grained original dataset is only applicable for the classifiers. Better performance values are bolded.

[10] https://github.com/SORD-src/JCST_Replication, Jan. 2022.



(a)

(b)

Fig.7. Regression model feature importance. (a) LR feature coefficients. (b) GBR_FS SHAP-feature value violin plot.

are around 0, and these coefficients range from the negative to the positive values. A better explanation may be that the trend (or concept) of every window varies greatly, which may also explain why univariate forecasting cannot achieve good performance.

Compared with the regressors, the bad performance of the time series based models is unexpected. As the univariate methods do not derive good results, transforming features to covariates for TCN forecasting is also unhelpful. Since covariates in multivariate approaches refer to the variables out of the system (e.g., holidays and seasons), we think involving more features out-of-the-box may be useful. However, since the basic performance of the time series based approaches is significantly inferior, we do not continue to explore.

We also perform the Augmented Dickey-Fuller (ADF) test [52] for the numbers of smelly developers to check if the time series are stationary. We find the time series in half of the projects are not stationary (with p-value < 0.05 and test result values greater than the critical level of 10%), while the other half of the time series are stationary. The performance of time series based approaches is below 0.4 for every project, revealing that such models may not be ideal for community smell forecasting. From the bad performance, we may also infer that the occurrence of community smells is hard to be fit with any patterns (e.g., seasonal, trending, or cyclical) of time series data supported by the applied forecasting models. Thus, we suggest investigating further the variation of community type and nature [11, 28] to understand the pattern of community healthiness and to design computational models for specific types of communities, since they have already outperformed time series based approaches in other tasks [52].

Finding 4. Community-wide regression models are harder to interpret. However, we can still observe the trend that sentiment features are more important than the others. Meanwhile, the regressors have different behaviors compared with the individual-wide classifiers. Since there still exists some room for the model performance to improve, more higher-level community-wide patterns should be captured. Moreover, we suggest developing computational models for different types of communities to support the forecasting of Smelly Developer.

6 Threats to Validity

Some threats may have influenced our study. Construct validity refers to the relationship between theory and observation. Conclusion validity is related to treatment and outcome. Internal validity concerns the factors other than the considered independent variable that could affect the occurrence of community smells. External validity is about the generalizability of results.

6.1 Construct Validity

The major threat to construct validity is the reliability of our datasets. We combine two sources of information, i.e., community smell detection results and developer sentiment.

In terms of community smell detection, we employ an open-source tool called CODEFACE4SMELLS. Tamburri *et al.* [1] provided detailed replication data to prove the dependability of the tool, i.e., its outputs were all true positives. Hence, we believe the tool is reliable. In addition, we follow strictly the installation, configuration, and execution guides [22] of the detection tool. As for software repositories and mailing lists, we fetch them from original sources of ASF and JBOSS. To a great extent, we can confirm the reliability of this source.

The developer sentiment dataset was proposed and improved progressively by Mantyla *et al.* [16, 20, 25]. Most data are automatically evaluated by lexicon-based tools, and all tools employed are state-of-the-arts. For example, the sentiment evaluation tool called SENTISTRENGTH [38] has also been proved reliable [24].

The process of combining the community smell detection result and the developer sentiment dataset may cause some loss in information. We make our best effort to match developers from both sides according to their e-mails and names. However, 8.8% of the smelly developers are not found in the sentiment dataset. Thus, their data are dropped. Nevertheless, we still manage to preserve most of the data.

The coherence of mailing lists and developer comments in JIRA may be a threat as well. Therefore, we investigate the contents of mailing lists. In eight out of 12 projects, they are automatically generated JIRA discussions. Otherwise, they contain JIRA-centered discussions, i.e., comments attaching links of JIRA issues.

6.2 Conclusion Validity

In terms of the reliability of model settings, we configure the hyper-parameters using Grid Search, and we employ three different validation strategies which were proved stable by prior studies [29, 42]. We also report results of classical evaluation metrics, i.e., precision, re-

call, F-measure, and AUC-ROC. Furthermore, we apply statistical tests, e.g., Cliff's Delta and SK-ESD, to validate the significance of our conclusions.

The reliability of the feature importance algorithm is also a threat to conclusion validity. The reason why we replace the Information Gain algorithm exploited in the conference version of our paper [21] is described in Subsection 5.2. Since model explanation is an emerging topic [32], these solutions may need improvement.

Moreover, the approach of building a new model for Smelly Developer may be questionable, since we can use the unification of the prediction results of the three smells. However, compared with the unification approach, building the classifier improves AUC-ROC performance for 2%, 2%, and 6% for time-wise, cross-project, and within-project scenarios respectively. Meanwhile, it improves cross-project and within-project F-measure performance by 4% and 7% respectively, while there is no difference in time-wise performance. Thus, to achieve better performance, it is more ideal to build a new model.

Finally, the choice of implementation to build a time series based model may also be a threat. In terms of classical univariate models, DARTS serves as a wrapper of widely-used implementations; thus we believe it is reliable.

6.3 Internal Validity

A notable internal validity may be the unidentifiable offline communications among developers. We observe certain developers prefer informal offline communications, as they mentioned the experience of offline talks frequently in ITS comments. In most cases, they made long comments to directly state the conclusion of offline communications. Evaluation based on online systems may underestimate their communicability. Moreover, there exist some lower-level metrics related to code quality that we do not involve, e.g., the naturalness of code, coupling and cohesion metrics. Meanwhile, we do not involve code review metrics [30] due to the lack of information sources.

6.4 External Validity

Since different projects may involve developers and communities in various backgrounds [23, 28], the multi-faceted nature of software development and developer sentiment is an unavoidable threat to external validity. To address this issue, we perform our study in 12 established OSS projects of two major open-source ecosystems to maximize the generality of our conclusion. Such systems have been analyzed in prior studies of software engineering [3, 45]. Note that 11 out of 12 projects analyzed in our paper are registered in the ASF foundation, because ASF provides better support for a variety of software development infrastructures such as ITS and mailing lists, and the infrastructures support data extraction, making ASF projects ideal sources for software analytics research.

7 Conclusions

This paper investigated whether the individual-wide and the community-wide occurrence of community smells could be predicted by the developers' experience, sentiment, and process metrics. We mainly considered three community smells, i.e., Lone Wolf, Organization Silo, and Bottleneck. Moreover, we included Smelly Developer and Smelly Quitter for prediction. We built machine learning models to predict the community smells' occurrence, and to understand the features' interactions with community smells. The results showed our individual-wide model achieves mean F-measures ranging from 0.73 to 0.92 in individual-wide within-project, time-wise, and cross-project prediction. Moreover, good performance of 0.68 in terms of R^2 is achieved in community-wide Smelly Developer prediction. We also revealed that developers with less communicability and heavier workload tend to be predicted as smelly. Meanwhile, being less polite or less certain in comments is also an indicator of smell occurrence. To conclude, we suggest core developers with heavier workload should foster more frequent communication in a straightforward and polite way. Future work includes: 1) integrating more effort-aware process metrics, 2) interpreting the pattern of sentiments' interactions with community smells, and 3) using graph neural networks to capture more community smells.

References

[1] Tamburri D A, Palomba F, Kazman R. Exploring community smells in open-source: An automated approach. *IEEE Trans. Softw. Eng.*, 2021, 47(3): 630-652. DOI: 10.1109/TSE.2019.2901490.

[2] Johnson B, Song Y, Murphy-Hill E, Bowdidge R. Why don't software developers use static analysis tools to find bugs? In *Proc. the 35th IEEE/ACM Int. Conference on Software Engineering*, May 2013, pp.672-681. DOI: 10.1109/ICSE.2013.6606613.

[3] Pecorelli F, Palomba F, Khomh F, De Lucia A. Developer-driven code smell prioritization. In *Proc. the 17th Int.*

Conference on Mining Software Repositories, June 2020, pp.220-231. DOI: 10.1145/3379597.3387457.

[4] Sae-Lim N, Hayashi S, Saeki M. Context-based code smells prioritization for prefactoring. In *Proc. the 24th IEEE Int. Conference on Program Comprehension*, May 2016. DOI: 10.1109/ICPC.2016.7503705.

[5] Martin F, Kent B, John B, William O, Don R. Refactoring: Improving the Design of Existing Code (1st edition). Addison-Wesley, 1999.

[6] Conejero J M, Rodríguez-Echeverría R, Hernández J, Clemente P J, Ortiz-Caraballo C, Jurado E, Sánchez-Figueroa F. Early evaluation of technical debt impact on maintainability. *J. Syst. Softw.*, 2018, 142: 92-114. DOI: 10.1016/j.jss.2018.04.035.

[7] Tamburri D A. Software architecture social debt: Managing the incommunicability factor. *IEEE Trans. Comput. Soc. Syst.*, 2019, 6(1): 20-37. DOI: 10.1109/TCSS.2018.2886433.

[8] Palomba F, Tamburri D A, Arcelli Fontana F, Oliveto R, Zaidman A, Serebrenik A. Beyond technical aspects: How do community smells influence the intensity of code smells? *IEEE Trans. Softw. Eng.*, 2021, 47(1): 108-129. DOI: 10.1109/TSE.2018.2883603.

[9] Palomba F, Tamburri D A. Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach. *J. Syst. Softw.*, 2021, 171: Article No. 110847. DOI: 10.1016/j.jss.2020.110847.

[10] Catolino G, Palomba F, Tamburri D A, Serebrenik A, Ferrucci F. Refactoring community smells in the wild: The practitioner's field manual. In *Proc. the 42nd ACM/IEEE Int. Conference on Software Engineering: Software Engineering in Society*, June 27-July 19, 2020, pp.25-34. DOI: 10.1145/3377815.3381380.

[11] Catolino G, Palomba F, Tamburri D A, Serebrenik A. Understanding community smells variability: A statistical approach. In *Proc. the 43rd ACM/IEEE Int. Conference on Software Engineering: Software Engineering in Society*, May 2021, pp.77-86. DOI: 10.1109/ICSE-SEIS52602.2021.00017.

[12] Ferreira I, Stewart K, German D, Adams B. A longitudinal study on the maintainers' sentiment of a large scale open source ecosystem. In *Proc. the 4th IEEE/ACM Int. Workshop on Emotion Awareness in Software Engineering*, May 2019, pp.17-22. DOI: 10.1109/SEmotion.2019.00011.

[13] Tamburri D A, Kazman R, Fahimi H. The architect's role in community shepherding. *IEEE Softw.*, 2016, 33(6): 70-79. DOI: 10.1109/MS.2016.144.

[14] Yue Y, Yu X, You X, Wang Y, Redmiles D. Ideology in open source development. In *Proc. the 13th IEEE/ACM Int. Workshop on Cooperative and Human Aspects of Software Engineering*, May 2021, pp.71-80. DOI: 10.1109/CHASE52884.2021.00016.

[15] Ducheneaut N. Socialization in an open source software community: A socio-technical analysis. *Comput. Support. Coop. Work*, 2005, 14(4): 323-368. DOI: 10.1007/s10606-005-9000-1.

[16] Mäntylä M, Adams B, Destefanis G, Graziotin D, Ortu M. Mining valence, arousal, and dominance: Possibilities for detecting burnout and productivity? In *Proc. the 13th Int. Conference on Mining Software Repositories*, May 2016, pp.247-258. DOI: 10.1145/2901739.2901752.

[17] Cheruvelil J, Da Silva B C. Developers' sentiment and issue reopening. In *Proc. the 4th Int. Workshop on Emotion Awareness in Software Engineering*, May 2019, pp.29-33. DOI: 10.1109/SEmotion.2019.00013.

[18] Huq S F, Sadiq A Z, Sakib K. Understanding the effect of developer sentiment on Fix-Inducing Changes: An exploratory study on Github pull requests. In *Proc. the 26th Asia-Pacific Software Engineering Conference*, December 2019, pp.514-521. DOI: 10.1109/APSEC48747.2019.00075.

[19] Valdivia-Garcia H, Shihab E, Nagappan M. Characterizing and predicting blocking bugs in open source projects. *J. Syst. Softw.*, 2018, 143: 44-58. DOI: 10.1016/j.jss.2018.03.053.

[20] Ortu M, Murgia A, Destefanis G, Tourani P, Tonelli R, Marchesi M, Adams B. The emotional side of software developers in JIRA. In *Proc. the 13th International Conference on Mining Software Repositories*, May 2016, pp.480-483. DOI: 10.1145/2901739.2903505.

[21] Huang Z, Shao Z, Fan G, Gao J, Zhou Z, Yang K, Yang X. Predicting community smells' occurrence on individual developers by sentiments. In *Proc. the 29th IEEE/ACM Int. Conference on Program Comprehension*, May 2021, pp.230-241. DOI: 10.1109/ICPC52881.2021.00030.

[22] Magnoni S. An approach to measure community smells in software development communities [Master Thesis]. Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, 2016.

[23] Catolino G, Palomba F, Tamburri D A, Serebrenik A, Ferrucci F. Gender diversity and community smells: Insights from the trenches. *IEEE Softw.*, 2020, 37(1): 10-16. DOI: 10.1109/MS.2019.2944594.

[24] Jongeling R, Datta S, Serebrenik A. Choosing your weapons: On sentiment analysis tools for software engineering research. In *Proc. the 31st IEEE Int. Conference on Software Maintenance and Evolution*, September 29–October 1, 2015, pp.531-535. DOI: 10.1109/ICSM.2015.7332508.

[25] Ortu M, Destefanis G, Kassab M, Counsell S, Marchesi M, Tonelli R. Would you mind fixing this issue? — An empirical analysis of politeness and attractiveness in software developed using agile boards. In *Proc. the 16th Int. Conference on Agile Software Development*, May 2015, pp.129-140. DOI: 10.1007/978-3-319-18612-2_11.

[26] Ortu M, Adams B, Destefanis G, Tourani P, Marchesi M, Tonelli R. Are bullies more productive? Empirical study of affectiveness vs. issue fixing time. In *Proc. the 12th IEEE/ACM Working Conference on Mining Software Repositories*, May 2015, pp.303-313. DOI: 10.1109/MSR.2015.35.

[27] Bell R M, Ostrand T J, Weyuker E J. The limited impact of individual developer data on software defect prediction. *Empir. Softw. Eng.*, 2013, 18(3): 478-505. DOI: 10.1007/s10664-011-9178-4.

[28] Catolino G, Palomba F, Tamburri D A. The secret life of software communities: What we know and what we don't know. In *Proc. the 18th Belgium-Netherlands Software Evolution Workshop*, November 2019.

[29] Yang Y, Zhou Y, Liu J, Zhao Y, Lu H, Xu L, Xu B, Leung H. Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models. In *Proc. the 24th ACM SIGSOFT Int. Symp. Foundations of Software Engineering*, November 2016, pp.157-168. DOI: 10.1145/2950290.2950353.

[30] McIntosh S, Kamei Y. Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction. *IEEE Trans. Softw. Eng.*, 2018, 44(5): 412-428. DOI: 10.1109/TSE.2017.2693980.

[31] Spadini D, Aniche M F, Bacchelli A. PyDriller: Python framework for mining software repositories. In *Proc. the 26th ACM Joint Meeting on European Software Engineering Conference and Symp. the Foundations of Software Engineering*, November 2018, pp.908-911. DOI: 0.1145/3236024.3264598.

[32] Jiarpakdee J, Tantithamthavorn C, Grundy J. Practitioners' perceptions of the goals and visual explanations of defect prediction models. In *Proc. the 18th IEEE/ACM Int. Conference on Mining Software Repositories*, May 2021, pp.432-443. DOI: 10.1109/MSR52588.2021.00055.

[33] Jiarpakdee J, Tantithamthavorn C, Dam H K, Grundy J. An empirical study of model-agnostic techniques for defect prediction models. *IEEE Trans. Softw. Eng.*. DOI: 10.1109/TSE.2020.2982385.

[34] Rajbahadur G K, Wang S, Ansaldi G, Kamei Y, Hassan A E. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Trans. Softw. Eng.*. DOI: 10.1109/TSE.2021.3056941.

[35] Lundberg S M, Erion G, Chen H, DeGrave A, Prutkin J M, Nair B, Katz R, Himmelfarb J, Bansal N, Lee S I. From local explanations to global understanding with explainable ai for trees. *Nat. Mach. Intell.*, 2020, 2(1): 56-67. DOI: 10.1038/s42256-019-0138-9.

[36] Graziotin D, Fagerholm F, Wang X, Abrahamsson P. What happens when software developers are (un)happy. *J. Syst. Softw.*, 2018, 140: 32-47. DOI: 10.1016/j.jss.2018.02.041.

[37] Graziotin D, Wang X, Abrahamsson P. Software developers, moods, emotions, and performance. *IEEE Softw.*, 2014, 31(4): 24-27. DOI: 10.1109/MS.2014.94.

[38] Thelwall M, Buckley K, Paltoglou G. Sentiment strength detection for the social web. *J. Am. Soc. Inf. Sci. Tec.*, 2012, 63(1): 163-173. DOI: 10.1002/asi.21662.

[39] Danescu-Niculescu-Mizil C, Sudhof M, Jurafsky D, Leskovec J, Potts C. A computational approach to politeness with application to social factors. In *Proc. the 51st Annual Meeting of the Association for Computational Linguistics*, August 2013, pp.250-259.

[40] De Smedt T, Daelemans W. Pattern for Python. *J. Mach. Learn. Res.*, 2012, 13: 2063-2067.

[41] Islam M R, Zibran M F. Towards understanding and exploiting developers' emotional variations in software engineering. In *Proc. the 14th IEEE Int. Conference on Software Engineering Research, Management and Applications*, June 2016, pp.185-192. DOI: 10.1109/SERA.2016.7516145.

[42] Tantithamthavorn C, McIntosh S, Hassan A E, Matsumoto K. An empirical comparison of model validation techniques for defect prediction models. *IEEE Trans. Softw. Eng.*, 2017, 43(1): 1-18. DOI: 10.1109/TSE.2016.2584050.

[43] Scott A J, Knott M. A cluster analysis method for grouping means in the analysis of variance. *Biometrics*, 1974, 30(3): 507-512. DOI: 10.2307/2529204.

[44] Pedregosa F, Varoquaux G, Gramfort A *et al.* Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.*, 2011, 12: 2825-2830.

[45] Palomba F, Zanoni M, Fontana F A, De Lucia A, Oliveto R. Toward a smell-aware bug prediction model. *IEEE Trans. Softw. Eng.*, 2019, 45(2): 194-218. DOI: 10.1109/TSE.2017.2770122.

[46] Esteves G, Figueiredo E, Veloso A, Viggiato M, Ziviani N. Understanding machine learning software defect predictions. *Autom. Softw. Eng.*, 2020, 27(3): 369-392. DOI: 10.1007/s10515-020-00277-4.

[47] Shapley L S. A value for n-person games. In *Contributions to the Theory of Games II, Annals of Mathematics Studies*, Kuhn H W, Tucker A W (eds.), Princeton University Press, 1953, pp.307-317.

[48] Palomba F, Panichella A, Zaidman A, Oliveto R, Lucia A D. The scent of a smell: An extensive comparison between textual and structural smells. *IEEE Trans. Softw. Eng.*, 2018, 44(10): 977-1000. DOI: 10.1109/TSE.2017.2752171.

[49] Kirbas S, Caglayan B, Hall T, Counsell S, Bowes D, Sen A, Bener A. The relationship between evolutionary coupling and defects in large industrial software. *J. Softw.: Evol. Process*, 2017, 29(4): Article No. e1842. DOI: 10.1002/smr.1842.

[50] Chicco D, Warrens M J, Jurman G. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation. *PeerJ Comput. Sci.*, 2021, 7: Article No. e263. DOI: 10.7717/peerj-cs.623.

[51] Yu X, Bennin K E, Liu J, Keung J W, Yin X, Xu Z. An empirical study of learning to rank techniques for effort-aware defect prediction. In *Proc. the 26th IEEE Int. Conference on Software Analysis, Evolution and Reengineering*, February 2019, pp.298-309. DOI: 10.1109/SANER.2019.8668033.

[52] Saini M, Kaur K. Fuzzy analysis and prediction of commit activity in open source software projects. *IET Softw.*, 2016, 10(5): 136-146. DOI: 10.1049/iet-sen.2015.0087.

[53] Manzano M, Ayala C, Gómez C, Cuesta L L. A software service supporting software quality forecasting. In *Proc. the 19th IEEE Int. Conference on Software Quality, Reliability and Security Companion*, July 2019, pp.130-132. DOI: 10.1109/QRS-C.2019.00037.

[54] Ahammed T, Asad M, Sakib K. Understanding the involvement of developers in missing link community smell: An exploratory study on Apache projects. In *Proc. the 8th Int. Workshop on Quantitative Approaches to Software Quality*, December 2020, pp.64-70.

[55] Hofmann H, Wickham H, Kafadar K. Letter-value plots: Boxplots for large data. *J. Comput. Graph. Stat.*, 2017, 26(3): 469-477. DOI: 10.1080/10618600.2017.1305277.

[56] Graziotin D, Wang X, Abrahamsson P. Happy software developers solve problems better: Psychological measurements in empirical software engineering. *PeerJ*, 2014, 2: Article No. e289. DOI: 10.7717/peerj.289.

[57] Müller S C, Fritz T. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In *Proc. the 37th IEEE/ACM Int. Conference on Software Engineering*, May 2015, pp.688-699. DOI: 10.1109/ICSE.2015.334.

[58] Lin B, Zampetti F, Bavota G, Di Penta M, Lanza M, Oliveto R. Sentiment analysis for software engineering: How far can we go? In *Proc. the 40th IEEE/ACM Int. Conference on Software Engineering*, May 27–June 3, 2018, pp.94-104. DOI: 10.1145/3180155.3180195.

[59] Jiarpakdee J, Tantithamthavorn C, Treude C. AutoSpearman: Automatically mitigating correlated software metrics for interpreting defect models. In *Proc. the 34th IEEE Int. Conference on Software Maintenance and Evolution*, September 2018, pp.92-103. DOI: 10.1109/ICSME.2018.00018.

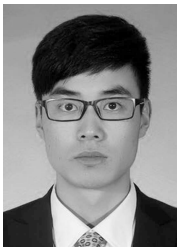
Zi-Jie Huang received his B.S. and M.E. degrees from Shanghai Normal University, Shanghai, in 2016 and 2020 respectively, both in computer science. He is currently pursuing his Ph.D. degree in computer science with the East China University of Science and Technology, Shanghai. His research interests include code smell, software quality assurance, and empirical software engineering.

Zhi-Qing Shao received his B.S. degree in mathematical logic from Nanjing University, Nanjing, in 1986, M.S. degree in pure mathematics from the Institute of Software, Chinese Academy of Sciences, Beijing, in 1989, and Ph.D. degree in computer software from Shanghai Jiao Tong University, Shanghai, in 1998. He is currently a professor at East China University of Science and Technology, Shanghai. His research interests include software engineering and network computing.

Gui-Sheng Fan received his B.S. degree from Anhui University of Technology, Ma'anshan, in 2003, M.S. degree from East China University of Science and Technology (ECUST), Shanghai, in 2006, and Ph.D. degree from ECUST in 2009, all in computer science. He is presently a research assistant of the Department of Computer Science and Engineering, ECUST, Shanghai. His research interests include formal methods for complex software system, service-oriented computing, and techniques for analysis of software architecture.

Hui-Qun Yu received his B.S. degree from Nanjing University, Nanjing, in 1989, M.S. degree from East China University of Science and Technology (ECUST), Shanghai, in 1992, and Ph.D. degree from Shanghai Jiao Tong University, Shanghai, in 1995, all in computer science. He is currently a professor of computer science with the Department of Computer Science and Engineering at ECUST, Shanghai. From 2001 to 2004, he was a visiting researcher in the School of Computer Science at Florida International University, Miami. His research interests include software engineering, high confidence computing systems, cloud computing, and formal methods. He is a member of ACM, and a senior member of CCF and IEEE.

Xing-Guang Yang received his B.S. degree in information security from China University of Mining and Technology, Xuzhou, in 2016, and Ph.D. degree in computer science from East China University of Science and Technology, Shanghai, in 2021. He is currently a lecturer at NingboTech University. His research interests include software engineering, software analysis, and software defect prediction.



Kang Yang received his B.S. degree in information and computing science from Shanghai Ocean University, Shanghai, in 2017. He is currently pursuing his Ph.D. degree in computer science with the East China University of Science and Technology, Shanghai. His research interests include software engineering and natural language processing.