Bug Report Priority Prediction Using Developer-Oriented Socio-Technical Features

Zijie Huang
Zhiqing Shao*
hzj@mail.ecust.edu.cn
zshao@ecust.edu.cn
East China University of Science and Technology
Shanghai, China

Guisheng Fan*
Huiqun Yu
gsfan@ecust.edu.cn
yhq@ecust.edu.cn
East China University of Science and Technology
Shanghai, China

Kang Yang
Ziyi Zhou
15921709583@163.com
zhouziyi@mail.ecust.edu.cn
East China University of Science and Technology
Shanghai, China

ABSTRACT

Software stakeholders report bugs in Issue Tracking System (ITS) with manually labeled priorities. However, the lack of knowledge and standard for prioritization may cause stakeholders to mislabel the priorities. In response, priority predictors are actively developed to support them. Prior studies trained machine learners based on textual similarity, categorical, and numeric technical features of bug reports. Most models were validated by time-insensitive approaches, and they were producing sub-optimal results for practical usage. Moreover, they tend to ignore the developer and social aspects of ITS. Since ITS bridges users and developers, we integrate their sentiment- and community-oriented socio-technical features to perform 2- and multi-classed bug priority prediction and validate our model in within-project, cross-project, and time-wise scenarios. The proposed model outperforms the 2 baselines by up to 10% in AUC-ROC and 13% in MCC, and the significance of improvement is statistically confirmed. We reveal involving assignee and reporter features from socio-technical perspectives such as sentiment could boost prediction performance. Finally, we test statistically the mean and distribution of the features that reflect the differences in socio-technical aspects (e.g., quality of communication and resource distribution) between high and low priority reports. In conclusion, we suggest researchers should involve contributors' experience and sentiments in bug report priority prediction.

CCS CONCEPTS

• **Software and its engineering → Risk management**.

KEYWORDS

bug report priority, developer sentiment, socio-technical analysis, issue tracking system, empirical software engineering

*Corresponding authors: Zhiqing Shao, Guisheng Fan.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Internetware 2022, June 11–12, 2022, Hohhot, China
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9780-3/22/06…$15.00
https://doi.org/10.1145/3545258.3545288

ACM Reference Format:
Zijie Huang, Zhiqing Shao, Guisheng Fan, Huiqun Yu, Kang Yang, and Ziyi Zhou. 2022. Bug Report Priority Prediction Using Developer-Oriented Socio-Technical Features. In *13th Asia-Pacific Symposium on Internetware (Internetware 2022), June 11–12, 2022, Hohhot, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3545258.3545288

1 INTRODUCTION

Bug fixing is a key task in software maintenance. However, the limited Software Quality Assurance (SQA) resources cannot support the localization and elimination of all potential malfunctioning programs and sub-optimal implementations that lead to bugs. As a result, practitioners and researchers have been trying to develop automatic approaches to predict, localize, prioritize, and fix bugs.

Practitioners report and track bugs in ITSs such as JIRA and Bugzilla. ITS incorporates the life-cycle of a bug from being discovered to being fixed. Once a bug report is created, the reporter (triager) is required to complete a form including a field concerning the priority of the bug. Afterwards, the assignee (developer) is in charge of fixing the bug. Bugs with higher priority deserve to be resolved sooner by contributors (*i.e.*, reporters and assignees) with more SQA resources. Since bug triaging is time-consuming, bug report priority prediction has been actively studied to save efforts. Researchers constructed prediction models incorporating both contextual and technical information. Textual analysis was exploited to capture the context of bug reports including topics and emotion [41, 44]. Meanwhile, they measured experiences of reporters [42, 43] and categorical information [34] of similar bug reports to represent technical characteristics. However, most studies were not explained [14] and validated [3] with the present standard, *e.g.*, lacked explanation to model performance and not time-sensitive. Moreover, the ITS was not modeled as a social system, *i.e.*, the patterns of communication and collaboration of stakeholders were not sufficiently considered and involved.

Recent studies shed light on the impact of socio-technical patterns [1, 30] and sentiments [11] to the development community healthiness, *i.e.*, the effectiveness of communication and collaboration. Meanwhile, community smells features capturing unhealthy collaboration and communication patterns in development community were proved effective to improve bug prediction performance [2]. Since ITS is a social system bridging reporters and assignees, we intend to investigate if socio-technical features measuring their sentiment and community could improve priority prediction.

To the best of our knowledge, there lacks a bug report priority prediction model involving the perspective of socio-technical features. This paper builds a sentiment and development community aware bug report priority prediction model. We develop machine learners upon a sentiment dataset [11, 27] over ITS systems to predict high and low priority bug reports. Furthermore, we assess the features' contribution to our model, as well as the impact of different parameter and validation settings. Finally, we analyze the significance of difference in features' distributions to discriminate high and low priority bug reports, and to explain model performance.

The main contributions of our work are listed as follows:

(1) To our knowledge, we build the first machine learning based approach integrating various types of developer sentiment- and community-oriented socio-technical features into bug report priority prediction. Our model outperforms the state-of-the-arts in the JIRA dataset.

(2) We explain our models using SHAP (SHapley Additive exPlanation) [20] feature importance. We discover that reporter and assignee socio-technical features contribute significantly to our model, and they are top-ranked features.

(3) We evaluate statistically the difference in the distribution of features that produce high and low priority prediction by assessing their related SHAP values. We infer the features can reflect difference in several socio-technical aspects between high and low priority reports.

(4) The replication dataset of this paper is available in [10].

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

2 RELATED WORK

2.1 Bug Report Priority and Severity Prediction

Most of the bug severity prediction studies were developed and improved based on a report similarity measure called REP [39]. The authors integrated BM25f-based textual similarity and various categorical features and defined a KNN (K Nearest Neighbours)-driven severity measuring method. The authors also divided the features into textual and non-textual.

To improve textual features, Zhang *et al.* [47] extended REP using LDA-based topic model to predict severity and recommend bug fixers. Instead of extending REP, Yang *et al.* [44] developed their own bug severity prediction model based on semantic similarity of positive and negative emotions in report comments and descriptions. Umer *et al.* [41] also measured emotions using a lexicon-based approach called Senti4SD to evaluate positive, negative, and neutral sentiments. The authors used embedded words and sentiments as input to train CNN-based deep learning models.

As for non-textual features, the improvements include involving product and component features by Tian *et al.* [40], as well as stack traces and more categorical features (*e.g.*, operating system) by Sabor *et al.* [34]. Alternatively, Valvida-Garcia *et al.* [42, 43] integrated experience features of reporters to build blocking bug prediction models based on various classical machine learning classifiers.

The major differences of our work to prior studies are:

(1) We involve various types of sentiments of reporters and assignees instead of coarsely positive, negative, and neutral sentiments to build predictors. Moreover, we also introduce community-oriented socio-technical features;

(2) We explain the model by assessing the features' importance and interpreting the model's behavior, because recent work revealed inexplainable models are unacceptable for practitioners [48];

(3) We exploit the more practical time-sensitive validation [3] which is not exploited in up-mentioned studies;

(4) The up-mentioned studies are multi-classed prediction on Bugzilla ITS, which is different from our two-, three-, and five-classed prediction on JIRA ITS. Meanwhile, the involved features and datasets are also different. Moreover, our approach outperformed 2 baseline studies [41, 47].

2.2 Socio-Technical Analysis on Developers

Task context including the developers' perception (social) [9, 32] as well as their development task (technical) [7, 35] are non-negligible aspects to comprehend and interpret software artifacts. Thus, socio-technical analysis combining the up-mentioned perspectives is performed on these artifacts.

Ortu *et al.* [22, 24–27] constructed a contributor sentiment dataset based on JIRA comments and sentences. The dataset is widely studied and regarded as a state-of-the-art dataset [15]. Furthermore, Ortu *et al.* also found that sentiments could reflect the severity of bug reports and impose impact on bug fixing. For example, they found impolite comments [26] and bullies [25] are related to longer bug fixing time. Meanwhile, VAD [22] (Valence-Arousal-Dominance) scores may help to identify productivity issues such as burnout, which may lead to longer bug fixing time.

Researchers also focused on community smells, *i.e.,* unhealthy developer community structure measured by patterns of motifs over collaboration and communication graphs in open-source development community [37]. Tamburri *et al.* [37] implemented a community smell detection tool called Codeface4Smells which evaluates development mailing list and software repository history information to detect various community smells. Based on their study, we [11] improved the prediction of community smell occurrence on individual developers by involving their sentiments, and we also suggested developers should communicate in a straightforward and polite way to improve community healthiness.

3 DATASET CONSTRUCTION

This section describes how we revise and generate features based on Ortu *et al.*'s dataset [27] and our prior work [11].

3.1 Analyzed Projects

We perform the prediction over 12 projects analyzed in a dataset of our prior study [11], which are listed in Table 1. We first calculate socio-technical features based on an original sentiment dataset [27] consists of project comments with evaluated sentiments. Afterwards, we extend the dataset using Codeface4Smells and additional mailing lists data to calculate socio-technical metrics. We

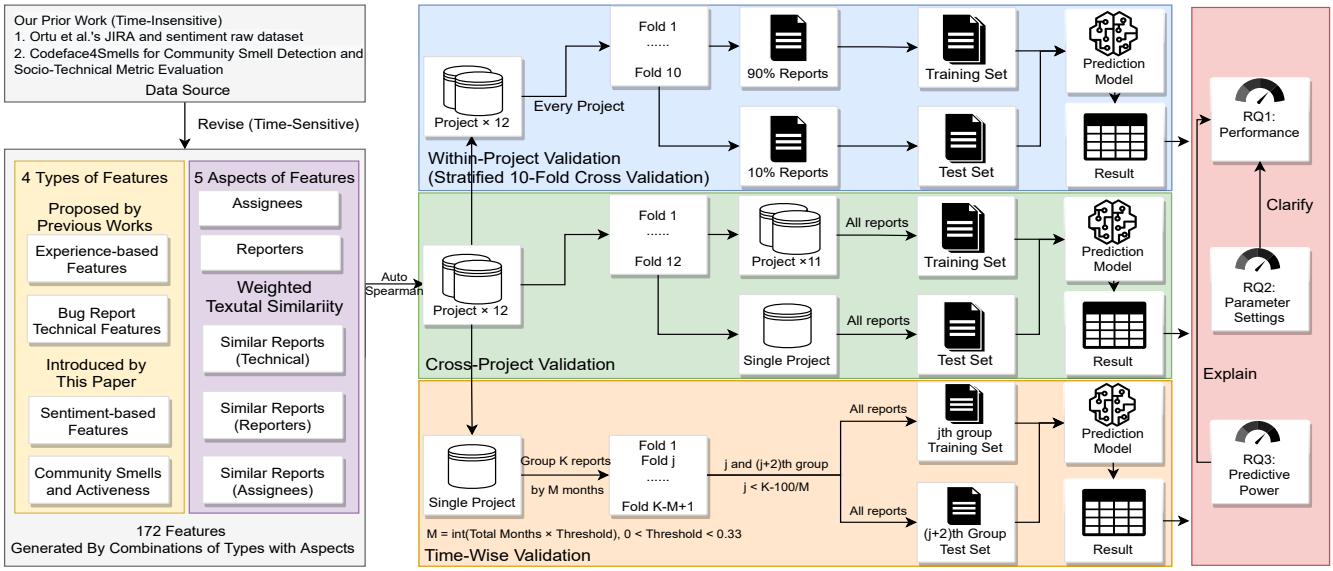

Figure 1: Overview of the prediction process.

Table 1: Overview of The Analyzed Projects

Project Name	Blocker	Critical	Major	Minor	Trivial
HBase	417	474	3295	953	221
Hadoop Common	712	245	2939	772	143
Hadoop HDFS	189	171	1627	492	57
Cassandra	39	137	1773	1357	284
Hadoop Map/Reduce	349	242	2256	425	77
Hive	131	160	2556	378	81
Harmony	712	111	3003	1044	123
OFBiz	27	80	1297	810	267
Hibernate ORM	164	307	3822	757	114
Camel	9	71	1619	567	46
Wicket	29	107	2297	757	137
Zookeeper	112	128	526	133	36

analyze these projects in [27] because their artifacts (*i.e.,* repository and mailing list) are acceptable by Codeface4Smells while others are not. We also revise our time-insensitive data with time series awareness, *i.e.*, data is sorted by the last updated timestamp, and no feature is generated based on future data.

3.2 5 Aspects of Feature Construction

Table 2 lists all 172 features extracted, and they will be processed by feature selection algorithm in the next section. The last 5 columns represent 5 aspects of the features, namely similar reports (S), reporters of similar reports (S_R), assignees of similar reports (S_A), reporter of the concerning report (R), and assignee of the concerning report (A). We combine the 5 aspects and 4 types of features in the first column to construct our dataset, *e.g.*, in the 3rd line we generate the sentiment valence in ITS comments for 5 aspects, producing 5 features, *i.e.*, S_SEN_VAL, S_R_SEN_VAL, S_A_SEN_VAL, R_SEN_VAL, and A_SEN_VAL. To clarify, S_SEN_VAL measures valence sentiment in comments of similar reports, while S_R_SEN_VAL measures valence sentiment in comments made by reporters of similar reports. Note that some combinations are inapplicable, so we do not generate them (*e.g.*, changelogs of reporters).

We use the features of similar reports instead of the features of the report being predicted because we intend to avoid a violation in time series, *i.e.*, we assume once a bug report is assigned to a developer, its priority should be predicted.

3.3 Measuring Textual Similarity

Textual similarity is included in most studies predicting bug report priority [4]. In this paper, we exploit Latent Semantic Indexing (LSI) and cosine similarity based approach to calculate the textual similarity between reports, which has been applied to calculate textual similarity between code fragments [28] and bug reports [35]. We use preprocessed (*i.e.*, tokenization, stop-words removal and stemming) text incorporating title and description as corpus for each bug report. We retrieve the top S similar reports and normalize their similarity to the range of [0,1] as the weight of every report. Then, we calculate the weighted summation of every feature of similar reports to construct features of similar reports (*i.e.*, columns named S, S_R, and S_A in Table 2) for every report.

3.4 Generating 4 Types of Features

We generate 4 types of features including sentimental, experience, community, as well as technical features. For sentimental features, based our prior study [11], our modification and improvement in this paper includes: (1) we calculate sentiments not only for reporters and assignees, but also for every bug report by measuring their comments, (2) the dataset used in this work is time-sensitive. For experience features, we measure the contributors' experience of commenting, serving as assignees, as well as reporting bugs and other types of issues (*e.g.*, new feature). For community smells

Table 2: The Extracted 172 Features (○ & ●) and The Selected 82 Features (●)

Features	Description	S	S_R	S_A	R	A
Sentiments [11, 24]						
SEN_VAL	Mean intensity of valence, *i.e.*, how developers enjoy a situation.	○	○	○	●	○
SEN_ARO	Mean intensity of arousal, *i.e.*, increased alertness.	○	○	○	○	●
SEN_DOM	Mean intensity of dominance, the extent that developers were feeling in control.	○	○	○	○	○
SEN_SAD	Mean intensity of all sadness expressions.	●	●	●	●	○
SEN_ANG	Mean intensity of all angry expressions.	●	●	●	●	●
SEN_LOV	Mean intensity of all love expressions.	○	○	○	○	●
SEN_JOY	Mean intensity of all joy expressions.	●	●	●	●	○
SEN_POS	Mean intensity of all sentiments greater than 0.	○	○	○	●	○
SEN_NEG	Mean intensity of all sentiments smaller than 0.	○	●	●	●	○
SEN	Summation of all positive and negative sentiments' intensities.	○	●	○	●	●
SEN_POL	Proportion of polite expressions in all commentary sentences of a developer.	●	○	○	●	○
SEN_IND	Proportion of indicative sentences that express fact or belief, *e.g.*, It is buggy.	○	○	○	●	●
SEN_IMP	Proportion of imperative sentences that express command, warning, *e.g.*, Do not produce bugs!	●	●	●	●	●
SEN_COND	Proportion of conditional sentences in the form like would, may, or will, *e.g.*, It might be buggy.	●	○	○	●	●
SEN_SUB	Proportion of subjunctive sentences in the form like wish, were, *e.g.*, I hope it works.	●	●	●	●	●
SEN_MOD (Modality)	The degree of uncertainty of a sentence, ranges between [-1,1].	○	○	○	○	○
Experience						
EXP_A	Count of unique issues assigned in the ITS regardless of their types.		○	○	○	○
EXP_A_BUG	Count of bug reports assigned in the ITS.		○	○	○	○
EXP_A_BUG_PROP	Ratio of EXP_A_BUG to EXP_A.		●	○	●	○
EXP_COM	Count of comments in issues regardless of their types.		○	●	○	○
EXP_COM_BUG	Count of comments in bug reports.		○	○	○	○
EXP_COM_BUG_PROP	Ratio of EXP_COM_BUG to EXP_COM.		●	●	●	●
EXP_R	Count of unique issues reported in the ITS regardless of their types.		○	○	○	●
EXP_R_BUG	Count of bugs reported in the ITS.		○	○	●	○
EXP_R_BUG_PROP	Ratio of EXP_R_BUG to EXP_R.		○	○	●	○
Community Smells and Activeness [11, 37]						
CS_ML (Missing Links)	Mean frequency that developer lacked communication with any co-committing developer.		○	○	○	○
CS_OS (Org. Silo)	Mean frequency that developer lacked communication with co-committing subgroups.		○	○	○	○
CS_RS (Radio Silence)	Mean frequency that contributor blocked communication between subgroups (boundary spanner).		○	○	○	○
CS (Is Smelly)	Mean frequency of a developer affected by any community smells in a given analysis window.		○	●	●	○
CS_QUIT	Mean frequency of a smelly developer quitted the community in a given analysis window.		●	●	●	●
ST_CD	Mean frequency of code commits.		○	○	○	○
ST_MD	Mean frequency of a developer commented in mailing list.		●	●	●	●
ST_COR	Mean frequency of acting as a core developer.		○	○	○	○
ST_SENT	Mean count of sentences commented in mailing list.		○	○	●	○
ST_SPO (Sponsored)	Mean frequency of a developer commited only in working hours in a given analysis window.		●	●	●	●
Bug Report Technical Features						
FEA_ATT	Weighted count of attachments.	●				
FEA_LOG	Weighted count of changelogs.	●				
FEA_COM	Weighted count of comments.	○				
FEA_COM_DUR	Weighted count of duration in months between the first and last comment.	●				
FEA_SUB	Weighted count of subtasks.	●				
FEA_VOTE	Weighted count of votes.	●				
FEA_WATCH	Weighted count of watchers.	●				
FEA_COM_SENT	Weighted count sentences in comments.	○				
FEA_FIX	Weighted count of fixed reports.	●				
FEA_PRIO	Weighted average of reports' original priority (originally ordinal numbers from 1 to 5).	●				
FEA_C_R_DUR	Weighted count of duration in months between the create and resolve timestamp.	●				
FEA_R_UPD_DUR	Weighted count of duration in months between the resolve and update timestamp.	●				
FEA_PROP_BLO	Proportion of blocking bugs in all similar reports.	●				
FEA_PROP_SEV	Proportion of severe bugs in all similar reports.	●				
FEA_PROP_MIN	Proportion of minor bugs in all similar reports.	●				
FEA_PROP_TRI	Proportion of trivial bugs in all similar reports.	●				

and activeness features, we apply the state-of-the-art tool Codeface4Smells [21, 37] to detect community smells. We generate data for every report and its stakeholders based on the last updated time of bug reports. For technical features in bug reports, we extract all features available for each bug report. We do not include any categorical features such as operating system or affected component [34], because they are not available in JIRA.

4 EXPERIMENTAL DESIGN

Fig. 1 depicts the overview of prediction process. The *goal* of our study is to evaluate how and to what extent bug report priority can be predicted by developer-oriented socio-technical features, with the *purpose* of providing practitioners with recommendations to automatically assign priorities or correct the mislabeled priorities (*e.g.*, high priority labeled low and *vice versa*). To these ends, we propose 3 research questions.

RQ1: *Can our model outperform the baselines?*
RQ2: *What is the impact of different parameter settings to our model?*
RQ3: *Which features contribute the most predictive power?*

4.1 RQ1-2: Defining and Validating the Proposed Model

4.1.1 Dependent Variables. Since there exist various treatments to the predicted priorities caused by the concern of the reliability of the manually assigned data, we follow the strategy of a prior study [38] and validate our model in 3 combinations of priorities, *i.e.,* two-, three-, and five-classed prediction.

The five-classed prediction refers to using the original priorities, namely blocking, critical, major, minor, and trivial.

In terms of the three-classed prediction, we follow the coarsely-grained setting of [4, 17, 18] that merges priorities into 3 classes. We leave the middle class untouched which contains the majority of bug reports. The 3 classes are presented as follows. High priority combines the reports originally labeled with blocking and critical. Medium priority refers to the reports originally labeled major. Low priority contains the reports originally labeled minor and trivial.

In terms of the two-classed prediction, we exclude the medium priority class in the two-classed prediction (*i.e.*, reports originally labeled major) because they share the same characteristics (*e.g.*, large sample size) with normal priority reports in Bugzilla [36]. Thus, we only predict the high and low priorities.

4.1.2 Independent Variables and Feature Selection. We use Autospearman [13] which is capable of completely removing multicollinearity while preserving most performance. We apply Autospearman to the features in Table 2 in order to (1) address potential multicollinearity problem, (2) improve the stability of feature importance interpretation [33].

4.1.3 Data Balancing. Bug report priority datasets are imbalanced [34, 39, 43], which may hinder model performance. Therefore, we preprocess our data with SMOTE, Random Under Sampling, and Random Over Sampling if they lead to better performance.

4.1.4 Training Machine Learners. We apply machine learners that have been used in prior studies [29, 31, 32, 39], including KNN, Random Forest, Decision Tree, Support Vector Machine, Multilayer Perceptron, Adaboost, Naive-Bayes, and Logistic Regression. We configure the hyper-parameters of the classifiers by exploiting Exhaustive Grid Search with a 10-Fold Cross-Validation strategy to select the best parameters instead of using default settings.

4.1.5 Performance Assessment. We build models separately in cross-project, within-project, and time-wise validation scenarios (see Fig. 1). Afterwards, we compute performance metrics including AUC-ROC and MCC (Matthews Correlation Coefficient) to pick the best classifier in the 3 scenarios. The range of AUC-ROC is 0.5 to 1, and the range of MCC is -1 to 1. We use only these 2 metrics because they are insensitive to data distribution and summarize the overall performance of a model which is more interpretable, while metrics such as F-Measure applied in prior studies [34, 41, 43, 47] are biased [46], and metrics such as Precision and Recall only focus on specific aspects. To clarify, for severely imbalanced datasets such as bug report priority whose middle class always account for the majority of data, it is inappropriate to use F-Measure to evaluate performance since a very high F-Measure score will be generated if the model classifies all instances to the class that contains the majority of instances (*e.g.,* the middle class). To rank the model performance and assess the significance of improvement, we also involve the Scott-Knott Effect Size Difference (SK-ESD) test.

4.1.6 Model Validation. (1) For cross-project validation, we apply a strategy similar to [29], which is a project-wide Leave-One-Out Cross-Validation, *i.e.*, we use 1 out of 12 projects as the test set, and the others as the training set to build our model. Such a process is performed 12 times. (2) For within-project validation, we apply Stratified 10-Fold Cross Validation in each project, *i.e.*, we build models separately for each project, and we randomly use 10% of all reports for testing and the rest for training. Such a process is performed 10 times. Since this strategy is proved reliable [32] for software engineering, we still acknowledge the potential flaw of such a method, as it use future data to predict earlier targets [45]. Thus, we involve time-wise validation to eliminate the drawbacks of this strategy, and to capture the drift of concept [23] in terms of bug report priority. (3) For time-wise validation, a demonstration of this process is available in Fig. 1. First, we use the last updated time for the timestamp to order each reports because it represents the final state of a report. Second, assume that we have K months of reports, we group the data by T% of months, each group contains reports in M=K×T% months. Third, we extract reports from month j to month M+j-1 as a group. Finally, due to the characteristic of version iteration [45], we train our models using data in group j, and validate the model in group j+2M. Such a process is performed for K-3M+1 times in each project. We do not use a fixed M because a fixed threshold (*e.g.*, 2 months in [45]) may not derive enough data and is inadaptive to projects. Meanwhile, we also assess the impact of different T selections to pick the best-performed one.

4.1.7 Baselines to Compare. We compare our model with two state-of-the-arts which achieved good performance in BugZilla datasets, namely cPur [41] and REPTopic [47]. cPur is a CNN-based predictor which enhanced paragraph vectors of bug reports with the extracted word vectors of positive, neutral, and negative sentiment, and REPTopic is a REP- [39] and KNN-based model which uses topic modeling to enhance the bug triaging performance. We use these two baselines because they are recent studies which outperformed

Table 3: Mean Performance and SK-ESD Rankings.

High and Low Priority (Two-Classed)						
	Within-Project		Time-Wise		Cross-Project	
	MCC	AUC	MCC	AUC	MCC	AUC
Our Approach	**0.34(1)**	**0.70(1)**	**0.20(1)**	**0.61(1)**	**0.24(1)**	**0.59(1)**
cPur	0.21(2)	0.60(2)	0.05(3)	0.51(3)	0.20(2)	**0.58(1)**
REPTopic	0.07(3)	0.51(3)	0.17(2)	0.57(2)	0.10(3)	0.53(2)
High, Medium, and Low Priority (Three-Classed)						
	Within-Project		Time-Wise		Cross-Project	
	MCC	AUC	MCC	AUC	MCC	AUC
Our Approach	**0.25(1)**	**0.60(1)**	**0.16(1)**	**0.56(1)**	**0.14(1)**	**0.58(1)**
cPur	0.13(3)	0.55(3)	0.04(2)	0.51(3)	0.10(2)	0.53(2)
REPTopic	0.23(2)	0.57(2)	**0.16(1)**	0.55(2)	0.07(3)	0.52(3)
The Original 5 Priorities (Five-Classed)						
	Within-Project		Time-Wise		Cross-Project	
	MCC	AUC	MCC	AUC	MCC	AUC
Our Approach	**0.19(1)**	**0.60(1)**	**0.16(1)**	**0.58(1)**	**0.07(1)**	**0.54(1)**
cPur	0.10(2)	0.53(2)	0.03(2)	0.51(3)	0.02(3)	0.50(3)
REPTopic	-0.11(3)	0.47(3)	0.07(3)	0.52(2)	0.04(2)	0.51(2)

other classical methods such as REP [39] and DRONE [40]. We compare the performance of our model with the baselines in RQ1, and we demonstrate the parameter settings in the best-performed prediction class in RQ2.

4.2 RQ3: Explaining the Predictive Power of Features

To address this RQ, we need to explain the extent of predictive power that each independent variable contributes to the best classifier in the best-performed class. We apply the SHAP algorithm, which has been studied empirically in a recent software engineering paper [33] 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 [19] to distribute the credit for a classifier's output among its features [33] using the game-theory based Shapley values [20].

Then, we take a closer look at the relationships of features' distribution and the prediction results for high and low bug priority, which is positive (Shapley value > 0) for high priority and negative for low priority. Since our data are not normally distributed, we apply the non-parametric Spearman's Rank Correlation Test [28]. Given two sets of values equal in length, the test produces a correlation coefficient ρ with a p-value to measure the significance level. We use the conventional threshold of p-value, *i.e.*, 0.05, to measure the correlation between the features' values and their Shapley values calculated. We consider the rank of correlation is trivial if $|\rho| < 0.10$, low if $0.10 \leq |\rho| < 0.30$, moderate if $0.30 \leq |\rho| < 0.50$, high if $0.50 \leq |\rho| < 0.70$, very high if $0.70 \leq |\rho| < 0.90$, and perfect if $|\rho| \geq 0.90$ [16]. Meanwhile, we calculate Cliff's Delta (δ) to measure the effect size (*i.e.*, the extent of the difference) for each pairs of feature values that produce positive and negative ϕ_i values. The effect size is negligible if $|\delta| < 0.147$, small if $0.147 \leq |\delta| < 0.33$, medium if $0.33 \leq |\delta| < 0.474$, and large if $|\delta| \geq 0.474$. Additionally, we also report the mean and variance of the features leading to positive and negative prediction results.

5 RESULT AND DISCUSSION

In this section, we answer the proposed research questions by demonstrating and discussing the results of our experiment.

5.1 RQ1: Model Performance

Table 3 lists the performance of our approach and the 2 compared baselines validated by SK-ESD. Our approach appears at the top rank in all 3 validation scenarios of the two-, three-, and five-classed predictions.

Generally, within-project prediction is the best-performed scenario since sufficient within-project data is provided for training. The low performance in cross-project predictions may be caused by the different characteristics of socio-technical patterns in projects. Meanwhile, lacking enough data in groups of the projects with fewer reports are harming the performance of time-wise models.

In terms of the performance of the baselines, cPur could achieve similar cross-project performance in the two-classed prediction, while REPTopic performs almost as good as our approach in the three-classed time-wise prediction. We find that the performance of cPur may be related to the scale of data, and it performs badly in time-wise validation since it may not produce enough data for training. The performance of REPTopic is inferior since many features in BugZilla are not available in the JIRA dataset, *e.g.,* categorical features. Meanwhile, we believe the reason why our approach works better is that a finer-grained approach to measure the characteristics of the title and description of bug reports is more feasible and adaptive if extra features such as operating system and component types are not available.

Finding 1. Our prediction model outperforms the baseline models by up to 10% in AUC-ROC and 13% in MCC, and it achieves better prediction result in within-project prediction. Meanwhile, it is always the top-ranked classifier in all prediction scenarios.

5.2 RQ2: Parameters and Settings

5.2.1 Selecting Classifier. We train multiple classifiers on the dataset and use the best performed one (*i.e.*, Random Forest) for cross-project, within-project, and time-wise prediction. We also demonstrate the medians of weighted average performances of the models ranked by SK-ESD in our online appendix [10].

5.2.2 Data Balancing. We apply Random Under Sampling in all validation scenarios. The improvement of performance is 2% to 7% in AUC-ROC and 2% to 6% in MCC.

5.2.3 Number of Similar Reports Selected. The impact of different S selection is not significant, and a greater S does not necessarily lead to better performance, which in line with prior studies [42, 43]. The figure which depicts the details is available in our online appendix [10]. We pick 10 as our parameter because except for MCC in within-project scenario, this selection leads to the best performance.

5.2.4 Months for Grouping in Time-Wise Validation. For time-wise validation, we also test the impact of selecting different T on model performance. Grouping data with 10% of available months produces

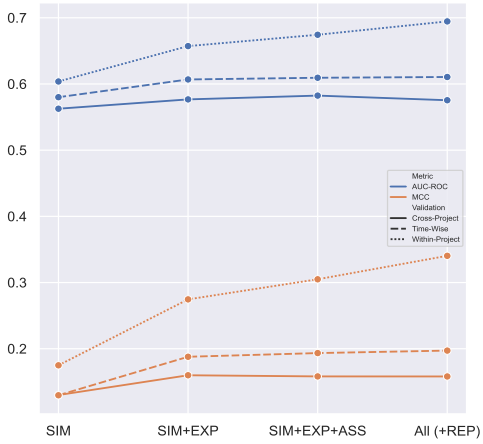

Figure 2: Model performance with different aspects of appended features.

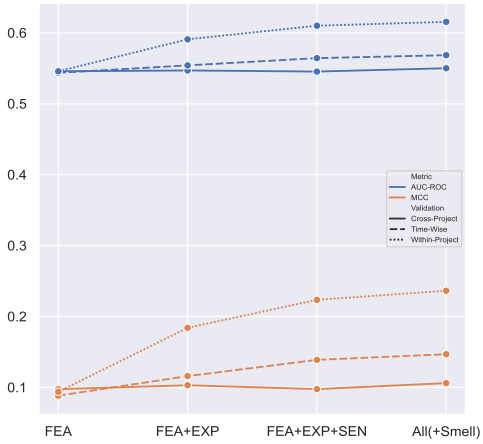

Figure 3: Model performance with different types of appended features.

the best performance. The impact of T is even more trivial than S (smaller than 0.01), so we do not present such a trivial impact.

5.2.5 Performance Variation Based on Subsets of Features. In terms of the performance variation of models built with features in different aspects, `SIM+EXP` are features proposed in prior studies [39, 40, 42, 43]. To figure out the impact of novel features proposed in this paper, we perform predictions based on sets of incrementally appended features, *i.e.*, `SIM` for features derive from similar reports, `EXP` for assignee and reporter experience features, `ASS` for all assignee features, and `REP` for all reporter features. Result in Fig. 2 shows similarity features are effective to some extent, but adding reporter- and assignee-related features gradually improves the model performance.

Similarly, Fig. 3 depicts the performance variation of models built with features of different types. `FEA` refers to conventional technical features of bug reports, `SEN` refers to sentiments, and `Smell` refers to community smell features. `FEA+EXP` features were proposed in prior studies. Sentiments and community smell features are gradually boosting the model performance in most cases.

Finding 2. The selection of classifiers and data balancing techniques impacts the model performance to the greatest extent. The newly involved developer-oriented sentiment and community features could improve model performance based on experience and technical features.

5.3 RQ3: Feature Importance and Model Behavior

To reveal the importance of features and the behavior of the prediction model, we intend to find the most discriminating features from the perspective of statistical analysis. Fig. 4 depicts a SHAP beeswarm plot displaying the feature values' impact on the correct prediction cases of high and low priorities in all 3 scenarios. Darker (lighter) points represent higher (lower) feature values. Meanwhile, data points in the right (left) represent higher (lower) Shapley values that lead to the prediction results of high (low) priority. The SK-ESD rankings of feature importance values are also presented in the Y-axis.

From Fig. 4, we can find some obvious and comprehensible trends. The row of `R_EXP_R_BUG` and `R_EXP_A_BUG_PROP` indicate bugs reported by an individual who reported or assigned more bugs tend to be a more severe one. `S_FEA_PROP_MIN` indicates a report having similar reports with higher proportion of minor priority tend to be predicted as less severe, while `S_FEA_PROP_BLO`, `S_FEA_PROP_SEV`, and `S_FEA_PRIO` show opposite trends. `A_EXP_COM_PROP` reveals reports assigned by individuals with more comments tend to be predicted as severe. `S_FEA_R_UPD_DUR` reveals that longer duration between the time when bug reports are marked as resolved and when they are updated later indicates they are more severe, which reflets the difficulty in diagnosing the bugs or preventing them from re-occurring. `S_FEA_FIX` shows that bug reports with more resolved similar bug reports are tend to be predicted as non-severe. `A_SEN_ANG` indicates that bug reports assigned by individuals leaving angry comments are tend to be predicted as severe.

Meanwhile, there exists also some trends which are more difficult to discriminate, especially for the sentimental features. Thus, to explain Fig. 4 in detail, we demonstrate the statistical relationships between the values of the hard-to-interpret features and their Shapley values in Table 4. Note that we only present Spearman's ρ and Cliff's δ if the results are statistically significant. Effect sizes in {Large, Medium, Small, Negligible} are mapped to {L, M, S, -}. Correlation ranks in {Perfect, High, Moderate, Low, Trivial} are mapped to {++++, +++, ++, +, -}.

Negative sentiments have always been regarded as problematic in developer sentiment analysis studies [5, 6]. Our findings in line with their observations since `S_A_SEN_ANG`, `R_SEN_ANG`, `R_SEN_SAD`, and `R_SEN_NEG` show similar trend as `A_SEN_ANG` in Fig. 4. However, our observation is different from a prior study since it reported negative sentiments appear more frequently in low priority bugs [41].

Our prior study showed certainty is an indicator of effective communication [11]. However, the characteristics of behaviors of dealing with certainty features differ in our model. In terms of

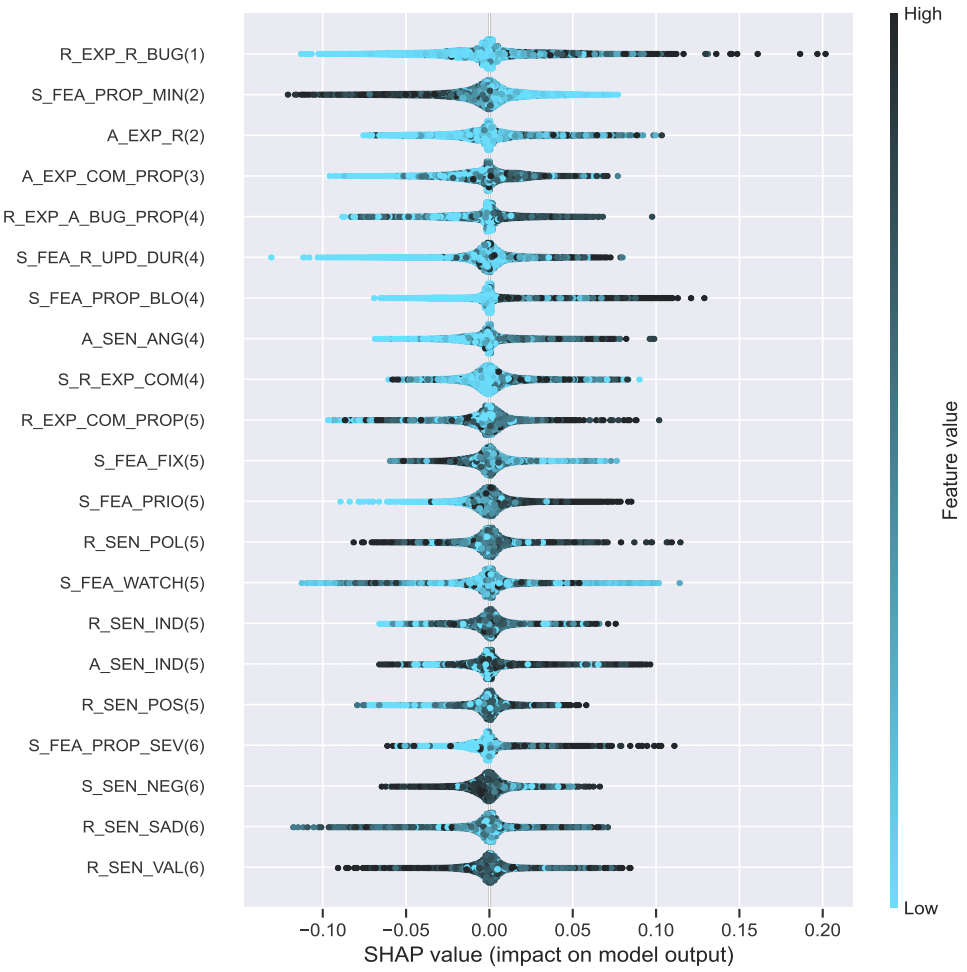

Figure 4: Feature values' impact to the correct prediction cases of our model.

indicative expression, R_SEN_IND and A_SEN_IND show that bug reports containing indicative expressions tend to be predicted as non-severe ones. In terms of imperative expression, A_SEN_IMP shows bug reports with more imperative expressions tend to be predicted as severe. S_R_SEN_SUB also shows that bug reports with more subjunctive expressions are more easily to be predicted as trivial ones.

In terms of positive sentiments, they are interpreted by prior study as a sign of more constructive collaboration [41]. Similarly, S_SEN_POL, R_SEN_POS, and A_SEN show more positive expressions lead to the prediction of lower priority. However, the S_A_SEN_LOV and R_SEN_POL features demonstrate an opposite trend. Since their effect sizes are small, we cannot infer any single sentiment leads to the prediction of high or low priority.

Apart from the sentimental features, there also exist other features to interpret. In terms of the proportion of comments in bug reports, S_A_EXP_COM_PROP and R_EXP_COM_PROP follow similar trend in Fig. 4 as A_EXP_COM_PROP. S_R_EXP_COM indicates reporter of higher priority bugs make more comments, while R_ST_SENT shows they leave less comment in developer mailing lists.

In terms of community smells, we find that their contribution to the model is trivial. Community smells are evaluated on mailing lists [37]. However, related study [8] pointed out that mailing lists are no longer acting as hubs for communication. Measuring community smells directly over ITS may improve the performance of such features. Nevertheless, we still discover the trials of less SQA resource allocation in low priority bugs which is reported in a prior study [26], *e.g.*, more severe community smells in low priority bugs, indicating a reduction in effective cooperation.

The proposed features can partly reflect differences in socio-technical aspects (*e.g.*, quality of communication and collaboration, SQA resource distribution) of high and low priority bug reports. Thus, we suggest researchers should involve socio-technical perceptions of bug reports and the feautres of assignees and reporters while preserving classical technical metrics.

Finding 3. Reporter and assignee features contribute significantly to our model, and they are among the top-ranked features.

Table 4: The Power of The Unmentioned Features to Discriminate High (High P.) and Low (Low P.) Priority Bug Reports

Feature	Spearman's ρ		Cliff's δ		Mean		Variance	
	Value	Rank	Value	Effect Size	High P.	Low P.	High P.	Low P.
R_SEN_IND	-0.67	+++	-0.63	L	0.15	0.35	0.02	0.04
S_FEA_COM_DUR	0.54	+++	0.58	L	0.21	0.08	0.02	0.01
R_ST_SENT	-0.47	++	-0.52	L	13.83	24.87	104.45	106.53
S_A_SEN_ANG	0.45	++	0.45	M	0.09	0.05	0.00	0.00
R_SEN_SAD	0.35	++	0.33	M	0.55	0.44	0.03	0.04
R_SEN_NEG	-0.39	++	-0.40	M	-0.11	-0.08	0.01	0.01
R_SEN_ANG	0.42	++	0.40	M	0.26	0.18	0.01	0.01
A_SEN_IMP	0.41	++	0.38	M	0.67	0.37	0.45	0.23
S_R_EXP_COM	0.37	++	0.35	M	57.15	45.73	6735.95	11667.98
S_SEN_POL	-0.31	++	-0.29	S	0.06	0.09	0.01	0.01
R_SEN_POS	-0.32	++	-0.23	S	0.14	0.17	0.04	0.03
A_SEN_IND	-0.30	++	-0.28	S	1.37	1.49	0.07	0.13
S_R_EXP_A_BUG_PROP	-0.30	++	-0.32	S	0.00	0.01	0.00	0.00
S_R_SEN_SUB	0.28	+	0.26	S	0.08	0.06	0.01	0.01
S_A_SEN_LOV	0.28	+	0.24	S	0.19	0.16	0.01	0.01
R_SEN_POL	0.25	+	0.22	S	0.15	0.13	0.02	0.03
R_SEN	-0.14	+	-0.20	S	0.01	0.01	0.00	0.00
A_SEN	-0.27	+	-0.33	S	1.14	1.55	0.47	0.67
S_FEA_C_R_DUR	0.17	+	0.17	S	0.04	0.03	0.00	0.00
S_A_EXP_COM_PROP	0.27	+	0.25	S	3.17	2.71	2.03	1.51
R_EXP_COM_PROP	0.22	+	0.19	S	0.67	0.63	0.01	0.01

Meanwhile, classical technical features are also among the most important features. While experience features are more powerful predictors than the sentiment features, sentiments also reflect certain aspects of bug triaging and cooperation.

6 THREATS TO VALIDITY

Construct Validity. The major threat to construct validity is the way we adjust bug report priority labels. In the two-classed prediction, we discard bug reports labeled major. A similar process is exploited in other studies discarding normal reports [17, 18, 34, 39, 47] because they contain high noise. Empirical research [36] showed the cause of noises in normal class were (1) the unwillingness to prioritize, (2) the absence of knowledge. Such problems occurred so common that normal priority bugs usually account for the largest population. However, normal priority is not presented in JIRA, and bug triagers are forced to choose between major and minor, *i.e.*, JIRA removed normal priority, but it did not eliminate the triagers' motivation to introduce noise. Since the reports labeled major share similar characteristics with the normal ones in size (see Table 1), we discard them to avoid noisy data for two-classed prediction.

Conclusion Validity. The major threat is that we evaluate our approach in RQ2-3 in merged classes. First, two-classed prediction is a common setting in priority prediction studies [4]. Second, multi-classed predictions [39, 40] were producing less ideal results for practical usage in our study and related works (*e.g.*, F-Measure < 0.4 in most classes [34, 39, 40, 47]), and interpreting unpromising models is less helpful [12]. Thus, we demonstrate the results in the most practical two-classed prediction.

External Validity. The advantage of our model may not be reproducible in other ITS since they may contain more stronger indicators than textual features (*e.g.*, operating system, component type, and categorical features in BugZilla). However, our approach is validated on a widely used JIRA dataset [27], and JIRA is a less studied trending ITS system. Thus, we think it is practical and constructive to perform prediction on JIRA.

7 CONCLUSION

This paper predicted bug report priority by developer-driven socio-technical features related to developer sentiments and community. Result showed our model outperformed the baseline models by up to 10% in AUC-ROC and 13% in MCC. Moreover, we inferred that the proposed features could reflect the differences in resource distribution as well as the quality of communication and collaboration of high and low priority reports. To conclude, we can improve bug report priority prediction by involving socio-technical features of reporters and assignees.

Future work includes: (1) the integration of more process- and assignee-related socio-technical metrics, (2) improving multi-class and cross-project prediction performance, (3) interpreting the sentiments' interactions with SQA activities.

ACKNOWLEDGMENTS

We wish to thank the three anonymous reviewers for their comprehensive and constructive comments. We also thank Xingguang Yang from NingboTech University for his proofreading and corrections.

This work was partially supported by the National Natural Science Foundation of China (No. 61772200), and the Natural Science Foundation of Shanghai (No. 21ZR1416300).

REFERENCES

[1] Manuel De Stefano, Fabiano Pecorelli, Damian A. Tamburri, Fabio Palomba, and Andrea De Lucia. 2020. Splicing Community Patterns and Smells: A Preliminary Study. In *42nd International Conference on Software Engineering Workshops (ICSEW)*. 703–710.

[2] Beyza Eken, Francis Palma, Ayse Basar, and Ayse Tosun. 2021. An empirical study on the effect of community smells on bug prediction. *Software Quality Journal* 29, 1 (2021), 159–194.

[3] Davide Falessi, Jacky Huang, Likhita Narayana, Jennifer Fong Thai, and Burak Turhan. 2020. On the need of preserving order of data when validating within-project defect classifiers. *Empirical Software Engineering* 25, 6 (2020), 4805–4830.

[4] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. 2019. Bug report severity level prediction in open source software: A survey and research opportunities. *Information and Software Technology* 115 (2019), 58–78.

[5] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson. 2017. Consequences of Unhappiness while Developing Software. In *2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. 42–47.

[6] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. 2018. What happens when software developers are (un)happy. *Journal of Systems and Software* 140 (2018), 32–47.

[7] L. Gren, P. Lenberg, and K. Ljungberg. 2019. What Software Engineering Can Learn from Research on Affect in Social Psychology. In *4th International Workshop on Emotion Awareness in Software Engineering (SEmotion)*. 38–41.

[8] A. Guzzi, A. Bacchelli, M. Lanza, M. Pinzger, and A. van Deursen. 2013. Communication in open source software development mailing lists. In *10th Working Conference on Mining Software Repositories (MSR)*. 277–286.

[9] M. Hozano, A. Garcia, N. Antunes, B. Fonseca, and E. Costa. 2017. Smells Are Sensitive to Developers! On the Efficiency of (Un)Guided Customized Detection. In *25th International Conference on Program Comprehension (ICPC)*. 110–120.

[10] Zijie Huang. 2022. *Replication dataset.* https://github.com/SORD-src/Internetware22Replication

[11] Zijie Huang, Zhiqing Shao, Guisheng Fan, Jianhua Gao, Ziyi Zhou, Kang Yang, and Xingguang Yang. 2021. Predicting Community Smells' Occurrence on Individual Developers by Sentiments. In *29th International Conference on Program Comprehension (ICPC)*. 230–241.

[12] J. Jiarpakdee, C. Tantithamthavorn, H. K. Dam, and J. Grundy. 2020.. An Empirical Study of Model-Agnostic Techniques for Defect Prediction Models. *IEEE Transactions on Software Engineering* 48, 2 (2020.), 166–185.

[13] Jirayus Jiarpakdee, Chakkrit Tantithamthavorn, and Christoph Treude. 2020. The impact of automated feature selection techniques on the interpretation of defect models. *Empirical Software Engineering* 25, 5 (2020), 3590–3638.

[14] Jirayus Jiarpakdee, Chakkrit Kla Tantithamthavorn, and John Grundy. 2021. Practitioners' Perceptions of the Goals and Visual Explanations of Defect Prediction Models. In *18th International Conference on Mining Software Repositories (MSR)*. 432–443.

[15] R. Jongeling, S. Datta, and A. Serebrenik. 2015. Choosing your weapons: On sentiment analysis tools for software engineering research. In *31st International Conference on Software Maintenance and Evolution (ICSME)*. 531–535.

[16] Serkan Kirbas, Bora Caglayan, Tracy Hall, Steve Counsell, David Bowes, Alper Sen, and Ayse Bener. 2017. The relationship between evolutionary coupling and defects in large industrial software. *Journal of Software: Evolution and Process* 29, 4 (2017), e1842.

[17] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals. 2010. Predicting the severity of a reported bug. In *7th Working Conference on Mining Software Repositories (MSR)*. 1–10.

[18] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck. 2011. Comparing Mining Algorithms for Predicting the Severity of a Reported Bug. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*. 249–258.

[19] Scott M Lundberg, Gabriel Erion, Hugh Chen, Alex DeGrave, Jordan M Prutkin, Bala Nair, Ronit Katz, Jonathan Himmelfarb, Nisha Bansal, and Su-In Lee. 2020. From local explanations to global understanding with explainable AI for trees. *Nature Machine Intelligence* 2, 1 (2020), 56–67.

[20] Scott M. Lundberg and Su-In Lee. December 2017. A Unified Approach to Interpreting Model Predictions. In *31st International Conference on Neural Information Processing Systems (NIPS)*. 4768–4777.

[21] Simone Magnoni. 2020. *An approach to measure community smells in software development communities.* https://github.com/maelstromdat/codeface4smells_TR

[22] Mika Mäntylä, Bram Adams, Giuseppe Destefanis, Daniel Graziotin, and Marco Ortu. 2016. Mining Valence, Arousal, and Dominance: Possibilities for Detecting Burnout and Productivity?. In *13th International Conference on Mining Software Repositories (MSR)*. 247–258.

[23] S. McIntosh and Y. Kamei. 2018. Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction. *IEEE Transactions on Software Engineering* 44, 5 (2018), 412–428.

[24] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do Developers Feel Emotions? An Exploratory Analysis of Emotions in Software Artifacts. In *11th Working Conference on Mining Software Repositories (MSR)*. 262–271.

[25] M. Ortu, B. Adams, G. Destefanis, P. Tourani, M. Marchesi, and R. Tonelli. 2015. Are Bullies More Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. In *12th Working Conference on Mining Software Repositories (MSR)*. 303–313.

[26] Marco Ortu, Giuseppe Destefanis, Mohamad Kassab, Steve Counsell, Michele Marchesi, and Roberto Tonelli. 2015. Would you mind fixing this issue?. In *16th International Conference on Agile Software Development (XP)*, Vol. 212. 129–140.

[27] M. Ortu, A. Murgia, G. Destefanis, P. Tourani, R. Tonelli, M. Marchesi, and B. Adams. 2016. The Emotional Side of Software Developers in JIRA. In *13th Working Conference on Mining Software Repositories (MSR)*. 480–483.

[28] Fabio Palomba, Annibale Panichella, Andy Zaidman, Rocco Oliveto, and Andrea De Lucia. 2018. The Scent of a Smell: An Extensive Comparison Between Textual and Structural Smells. *IEEE Transactions on Software Engineering* 44, 10 (2018), 977–1000.

[29] Fabio Palomba and Damian Andrew Tamburri. 2021. Predicting the emergence of community smells using socio-technical metrics: A machine-learning approach. *Journal of Systems and Software* 171 (2021), 110847.

[30] F. Palomba, D. A. Tamburri, F. Arcelli Fontana, R. Oliveto, A. Zaidman, and A. Serebrenik. 2021. Beyond Technical Aspects: How Do Community Smells Influence the Intensity of Code Smells? *IEEE Transactions on Software Engineering* 47, 1 (2021), 108–129.

[31] F. Palomba, M. Zanoni, F. A. Fontana, A. De Lucia, and R. Oliveto. 2019. Toward a Smell-Aware Bug Prediction Model. *IEEE Transactions on Software Engineering* 45, 2 (2019), 194–218.

[32] Fabiano Pecorelli, Fabio Palomba, Foutse Khomh, and Andrea De Lucia. 2020. Developer-Driven Code Smell Prioritization. In *17th International Conference on Mining Software Repositories (MSR)*. 220–231.

[33] G. K. Rajbahadur, S. Wang, G. Ansaldi, Y. Kamei, and A. E. Hassan. 2021. The impact of feature importance methods on the interpretation of defect classifiers. *IEEE Transactions on Software Engineering* (2021), Early Access.

[34] Korosh Koochekian Sabor, Mohammad Hamdaqa, and Abdelwahab Hamou-Lhadj. 2020. Automatic prediction of the severity of bugs using stack traces and categorical features. *Information and Software Technology* 123 (2020), 106205.

[35] Natthawute Sae-Lim, Shinpei Hayashi, and Motoshi Saeki. 2018. Context-based approach to prioritize code smells for prefactoring. *Journal of Software: Evolution and Process* 30, 6 (2018), e1886.

[36] R. K. Saha, J. Lawall, S. Khurshid, and D. E. Perry. 2015. Are These Bugs Really "Normal"?. In *12th Working Conference on Mining Software Repositories (MSR)*. 258–268.

[37] D. A. Tamburri, F. Palomba, and R. Kazman. 2021. Exploring Community Smells in Open-Source: An Automated Approach. *IEEE Transactions on Software Engineering* 47, 3 (2021), 630–652.

[38] Yuan Tian, Nasir Ali, David Lo, and Ahmed E. Hassan. 2016. On the Unreliability of Bug Severity Data. *Empirical Software Engineering* 21, 6 (2016), 2298–2323.

[39] Y. Tian, D. Lo, and C. Sun. 2012. Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction. In *19th Working Conference on Reverse Engineering (WCRE)*. 215–224.

[40] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated Prediction of Bug Report Priority Using Multi-Factor Analysis. *Empirical Software Engineering* 20, 5 (2015), 1354–1383.

[41] Q. Umer, H. Liu, and I. Illahi. 2020. CNN-Based Automatic Prioritization of Bug Reports. *IEEE Transactions on Reliability* 69, 4 (2020), 1341–1354.

[42] Harold Valdivia-Garcia and Emad Shihab. 2014. Characterizing and Predicting Blocking Bugs in Open Source Projects. In *11th Working Conference on Mining Software Repositories (MSR)*. 72–81.

[43] Harold Valdivia-Garcia, Emad Shihab, and Meiyappan Nagappan. 2018. Characterizing and predicting blocking bugs in open source projects. *Journal of Systems and Software* 143 (2018), 44–58.

[44] Geunseok Yang, Tao Zhang, and Byungjeong Lee. 2018. An Emotion Similarity Based Severity Prediction of Software Bugs: A Case Study of Open Source Projects. *IEICE Transactions on Information and Systems* E101.D, 8 (2018), 2015–2026.

[45] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. 2016. Effort-Aware Just-in-Time Defect Prediction: Simple Unsupervised Models Could Be Better than Supervised Models. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. 157–168.

[46] Jingxiu Yao and Martin J. Shepperd. 2021. The impact of using biased performance metrics on software defect prediction research. *Information Software Technology* 139 (2021), 106664.

[47] Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, and Xiapu Luo. 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software* 117 (2016), 166–184.

[48] W. Zou, D. Lo, Z. Chen, X. Xia, Y. Feng, and B. Xu. 2020. How Practitioners Perceive Automated Bug Report Management Techniques. *IEEE Transactions on Software Engineering* 46, 8 (2020), 836–862.