2022 IEEE/ACM 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP) | 978-1-6654-9590-5/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSE-SEIP55303.2022.9793952

Bug Tracking Process Smells In Practice

Erdem Tuna
erdem.tuna@bilkent.edu.tr
Bilkent University
Ankara, Turkey

Vladimir Kovalenko
Vladimir.Kovalenko@jetbrains.com
JetBrains Research
Amsterdam, The Netherlands

Eray Tüzün
eraytuzun@cs.bilkent.edu.tr
Bilkent University
Ankara, Turkey

ABSTRACT

Software teams use bug tracking (BT) tools to report and manage bugs. Each record in a bug tracking system (BTS) is a reporting entity consisting of several information fields. The contents of the reports are similar across different tracking tools, though not the same. The variation in the workflow between teams prevents defining an ideal process of running BTS. Nevertheless, there are best practices reported both in white and gray literature. Developer teams may not adopt the best practices in their BT process. This study investigates the non-compliance of developers with best practices, so-called smells, in the BT process. We mine bug reports of four projects in the BTS of JetBrains, a software company, to observe the prevalence of BT smells in an industrial setting. Also, we survey developers to see (1) if they recognize the smells, (2) their perception of the severity of the smells, and (3) the potential benefits of a BT process smell detection tool. We found that (1) smells occur, and their detection requires a solid understanding of the BT practices of the projects, (2) smell severity perception varies across smell types, and (3) developers considered that a smell detection tool would be useful for six out of the 12 smell categories.

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**; *Software configuration management and version control systems*; **Maintaining software**.

KEYWORDS

bug tracking system, empirical study, developer perception, bug tracking smells, process smell

ACM Reference Format:
Erdem Tuna, Vladimir Kovalenko, and Eray Tüzün. 2022. Bug Tracking Process Smells In Practice. In *44nd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22), May 21–29, 2022, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3510457.3513080

1 INTRODUCTION

Developers inevitably introduce bugs to software. Quality assurance engineers and end users encounter these bugs, which brings the need for a communication channel to report the bugs and track progress on their resolution. Companies, communities, and developer teams use the process of bug tracking (BT) to govern the bugs. Dedicated bug tracking systems (BTS) provide a medium to run the BT process in an organized manner. Although bots and automated actors may take part in bug tracking [15, 30], the primary agent of a BTS is individuals. Most actions in the bug tracking process, such as submitting, triaging, assigning, prioritizing, linking, and resolving the bugs, are conducted by humans: either developers or end users reporting the issues.

To err is human, and the process and products of human work are often suboptimal. A growing body of research in software engineering is dedicated to identifying the so-called *smells*, the characteristics and patterns in software systems that are either suboptimal or dangerous per se or indicate underlying issues. The scope of research on smells is very broad [20]. It ranges from code smells [28], through higher-level architecture and design smells [17, 29], to the team- and ecosystem-level community smells [2]. Smells can be specific to a certain aspect, such as security [8].

The potential benefit of identifying smells in systems is multifaceted. First, the presence of smells indicates technical and organizational risks, which might help prioritize preventive measures such as maintenance activities. Moreover, once detected, some smells can be addressed with automatic approaches: for example, many code smells can be eliminated with reengineering tools [10] or automated suggestions from static analysis tools [23].

A less studied, yet emerging area is the identification of smells in *processes* rather than artifacts. Process smells are defined as the deviations from best practices in either the development process as a whole [25] or in its specific parts, such as code review [6] and bug tracking [18]. In a recent study, Qamar et al. [18] compile a taxonomy of 12 potential smells in the bug tracking process and propose algorithms for automated mining of these smells.

In this work, we put the taxonomy and the detection tools to the test in an industrial setting. Through a case study at JetBrains, a vendor of software engineering tools, we (1) investigate the occurrence of bug tracking smells in four industrial projects, (2) assess the perception of the smells by internal users of the bug tracking tool, and (3) gauge the potential added value of detecting and presenting the bug tracking process smells to the users.

The contributions of this work are:

- The first study on the occurrence of bug tracking smells in industrial software projects;
- An assessment of perception of relevance and severity by software professionals for each of the bug tracking smells;
- An evaluation of the potential added value of automatic detection of bug tracking smells.

The remainder of this paper is organized as follows. In the following section, we present the background information. In Section 3, the study design is described. Section 4 presents the results of the

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 the author(s) 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.
ICSE-SEIP '22, May 21–29, 2022, Pittsburgh, PA, USA
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9226-6/22/05…$15.00
https://doi.org/10.1145/3510457.3513080

empirical evaluation on four projects and the conducted survey. In Section 5, the empirical results and the survey results are discussed. Section 6 addresses validity threats of this study and finally, Section 7 presents our conclusion and future work.

2 BACKGROUND

2.1 Research on Bug Tracking

The process of issue tracking, and bug tracking in particular, is one of the pillars of modern software engineering. Numerous studies in the software research community have focused on understanding the bug tracking process and proposing automated approaches to enhance the bug tracking tools. Alipour et al. [1] developed a context-aware bug deduplication method to find duplicate bug reports. Zimmermann et al. [33] investigated the components of a comprehensive bug report. They surveyed both developers and bug reporters identified that stack traces and steps to reproduce are the most helpful information in bug reports. Soltani et al. [21] checked how significant different information provided in a bug report. To curate developer views, they conducted interviews. They validated their findings by surveying developers. It is concluded that developers consider crash description and reproducing steps are the most important information in a bug report (similar to [33]). Tamrawi et al. [24] developed a fuzzy set approach combined with caching technique to predict bug assignee. Besides, Xuan et al. [31] found that more effective assignee prediction is possible if the used number of data points is reduced. Tian et al. [26] introduced a new framework for predicting the priority field in bug reports. They extracted features from data and metadata of bug reports such as bug description, author name, and severity level. Linear regression-based classifier operates based on the generated features and predicts the priority of a bug report. Umer et al. [27] studied priority field prediction by considering deep learning techniques. They utilize textual features obtained from the description of the bug report and emotion value. Priority classification is realized with a convolutional neural network model. Lamkanfi et al. [13] explored the applicability of the machine learning concept into the severity field prediction problem for the first time. They used Naïve Bayes classifier using the textual features extracted from the text in bug report descriptions.

There are also studies focusing on the reassignment of values in different fields of bug reports [12] and classification of bug types [5]

The studies focus on specific parts or fields in the bug reports. They provide ways to increase and enhance the accuracy or quality of choosing a value of the related fields. Adopting better methods is important to enhance a process. Observing the current status enables recognizing the hotspot in a process. Hotspots could be a problem of the present, but they could also hint at long-living problems. Buse and Zimmermann [3] found that managers and developers think that, related to a software process, the answer of "what happened (in the past)" is more important than the answer of "what will happen (in the future)". Thus, interpreting the past and current status of a BT process would provide the relevant analytics to practitioners.

2.2 Bug Tracking Process Smells

The literature investigating the smells in the BT process is new. Qamar et al. [18] conducted the first study that systematically collects and analyzes the potentially suboptimal patterns in the BT process through a review of both white and grey literature. Qamar's study proposes a taxonomy of 12 *bug tracking process smells*, which they define as "deviations from best practices". Along with the taxonomy, they suggest methods to detect each of the smells. Moreover, the study explores the frequency of each smell in six open source projects. Their results suggest that the smells (1) do occur in practice, and (2) each smell has a different occurence ratio across different projects, and (3) some smells decrease in frequency over time in specific projects. We refer the reader to [18] for the complete definitions of the smells.

In this study, we want to observe the smells in an industrial context (more specifically, projects of JetBrains) instead of open source projects.

2.3 Industry Case Studies in Software Engineering

Software engineering researchers conduct case studies in private companies involving developers to explore the applicability and relevance of academic research results in practice [4, 9, 14, 16, 19, 22, 32]. Empirical studies in industry are the ultimate means to gauge the perception of the concepts and methods from academia by software professionals.

Strand et al. [22] deployed and evaluated their new code reviewer recommendation tool at Ericsson. They employed surveys and interviews to evaluate the tool's performance and understand the perception of practical feasibility of the tool by developers. Sadowski et al. [19] analyzed the current practices and the driving motives for the code review process at Google. The authors surveyed 44 developers to understand the value and effect of code reviews. Moreover, they interviewed 12 developers about their motivation for participation in the code review activity. Lewis et al. [14] explored the benefits of a bug prediction system for developers. They investigated the usefulness of two well-known bug prediction algorithms in practice. By interviewing 19 developers at Google, they found that the algorithms do not appear very useful to developers. Yan et al. [32] analyze the performance of a state-of-the-art defect identification system on Alibaba projects. Their study reveals that tool recommendations are less accurate than prior studies involving open-source studies. They conclude that the algorithms should be customized to account for project-specific factors.

Industrial studies that involve techniques and tools originating in academia are an invaluable source of information to validate the correctness of assumptions and understand the potential barriers to adoption of the techniques. Similarly to these studies, with this one we are looking to confirm that the results generated by the algorithms for bug tracking smells are indeed aligned with the intention behind them, and to understand the developers' perspective on BT process smells.

3 STUDY DESIGN

Our study consists of several steps. First, we selected the target projects and collected a historical dataset from the bug tracking

system. Then, we adapted the smell detection algorithms [18] to the YouTrack platform. After that, we detected the smells in the target projects[1] and obtained the data on prevalence of the smells. Finally, we designed and ran a survey serving several purposes: (1) to validate the smell detection algorithms and learn if developers consider smells as a deviation from the ideal process; (2) to understand the importance and severity of the smells from the developers' point of view; (3) to explore the demand and benefits of a potential smell detection tool. This section expands the details of each step.

3.1 Research Questions

We build this study around the following research questions.

RQ1. How common is each of the bug tracking smells in industrial projects?

The foundational study on bug tracking smells [18] inspects the frequency of BT process smells in open source projects. We are looking to explore the prevalence of smells in proprietary projects to complement their results.

RQ2. Do developers agree on the definitions and detection methodologies of BT process smells?

Generalization of the BT process smell detection algorithms could lead to inaccurate analyses due to differences between practices in teams and the bug tracking systems they use. Every BTS brings its own data format, and the details of smell detection depend on it. The practices followed in different teams may influence the existence of smells. A practice that is undesirable in one team might be intentional in another. Even individual developers on the same team may perceive the smells differently. With this research question, we want to explore the understanding of the smells across different teams and individuals.

RQ3. How do developers perceive the severity of each smell?

Even if a bug tracking process involves suboptimal patterns, and developers accept them as smells, the smells should not necessarily be addressed. For example, if fixing a smell is too cumbersome, the optimal solution could be to continue the process as is. Thus, the smells should be prioritized based on their severity. With this research question, we want to understand the developers' perception of smells severity.

RQ4. Can a tool detecting and presenting smell results to developers be helpful?

The bug tracking smells could potentially be automatically detected and presented to the developers either within the bug tracking platform or in a standalone tool. With this research question, we are looking to gauge the developers' opinions on the potential value of a smell detection tool in practice.

3.2 Scope and Data

3.2.1 Target Company. Our target company, JetBrains, is a world-renowned vendor of software engineering tools. The company employs over 1,500 people in multiple locations across the globe, and offers over 20 products for individual software engineers and software engineering teams.

One trait of JetBrains that makes it a particularly suitable target for our case study is the lack of strictly imposed company-wide process policies. This means that every team tailors their process to their own needs, which allows us to assess the perception and potential usefulness of bug tracking smells in several diverse environments.

Table 1: Overview of target projects. The numbers are approximate.

Project	Number of Issues	Number of Bugs	Number of Developers
Project-A	15,000	5,000	90
Project-B	25,000	15,000	100
Project-C	55,000	20,000	30
Project-D	230,000	150,000	150

3.2.2 Target Platform. The primary platform for issue tracking (and bug tracking in particular) at JetBrains is YouTrack. It has been in use for over 10 years, and is used by both internal reporters and end users who can report bugs directly to development teams. Each ticket in YouTrack belongs to a particular project. The format of tickets (e.g. field set, requested fields, default automated workflows) is customized per project.

3.2.3 Dataset. We selected four target projects based on the two criteria: the number of bugs and developers. The number of bugs is essential to make meaningful observations on patterns followed in the BT process of a project. The results obtained from a small number of bugs would limit generalizability. On the other hand, selecting projects with more developers would increase the participation in surveys. We present the information for the selected projects in Table 1. For industrial secrecy and privacy reasons, we do not disclose the project names and refer to them as *ProjectA..D* in the paper.

We utilized the YouTrack REST API[2] to download the complete issue histories of the selected projects. The projects have varying types of issues such as *task*, *bug*, or *feature*. The *bug* type is shared among all selected projects, and we included the issues with the type *bug* in our dataset and ignored the rest.

3.3 Smells and Detection

3.3.1 Smell Categories. As indicated in Section 2.2, we follow the taxonomy of BT process smells defined by Qamar et al. [18]. The taxonomy consists of 12 potentially suboptimal bug tracking practices. We group them into two categories, detailed below.

Smells from Fields. The smells in this category are detected by inspecting the field values of the bug report. It includes five smells: *Unassigned Bugs*, *Bugs Assigned to a Team*, *Missing Environment Information*, *Missing Priority*, and *Missing Severity*. All these smells can be detected by checking the corresponding field values.

Smells from Activities. The smells in this category are detected by inspecting the bug's activities and, if required, some field values. The remaining seven smells are included in this category are *Reassignment of Bug Assignee*, *No Link to Bug-Fixing Commit*, *Not Referenced Duplicates*, *No Comment Bugs*, *Non-Assignee Resolver of Bug*, *Closed-Reopen Ping Pong*, and *Ignored Bugs*.

[1] In this paper, *project* refers to the BT projects in YouTrack (Section 3.2.2)

[2] https://www.jetbrains.com/help/youtrack/devportal/youtrack-rest-api.html

3.3.2 Customized Detection Methodology. Qamar et al. [18] present a methodology for the detection of the smells in addition to the taxonomy. Originally designed to operate with the data format of Jira[3], the detection methodologies are directly compatible with the data format of YouTrack with exceptions for some smells that we could however address with minor customizations. In this section, we detail these customizations.

Jira keeps the state and the resolution status of a bug report in two separate fields. Also, resolution status can take several values. However, YouTrack keeps only the state of a bug report. The resolution status is embedded inside states as either *true* or *false*. This fact prevents us from directly using the detection methodologies presented. We adapted the detection methodologies accordingly by checking the definition and the detection for the Jira platform. For every project, we analyzed the states and their resolution values. In this analysis, we also considered the state diagram of the projects and the state transitions in practice in YouTrack. We clustered the states into four and note that *States-A* and *States-B* are mutually exclusive and constitute the whole *States* set of a project.

States-A: Every state in the project setting with resolution status *unresolved*.

States-B: Every state in the project setting with resolution status *resolved*.

States-C: Bug reports may require any kind of developer work such as visual inspection, attempt to reproduce, commit, etc. After developers conduct the required work, the report is resolved. Thus, we include the states that involve any kind of work and have resolution status *resolved* in this type. *States-C* is a subset of *States-B*.

States-D: Similar to *States-C*, some states definitely require commit activity by developers in version control systems. Thus, we include the states that involve commit activity and have resolution status *resolved* in this type. *States-D* is a subset of *States-C*.

We explain the use of the identified state types for the detection of the related smells below. We follow the detection methodologies for the rest of the smells as described in [18].

Unassigned Bugs: If the bug is in one of the states in *States-C*, the bug is eligible for the smell analysis. If so, we check if the assignee field is null or not.

No Link to Bug-Fixing Commit: If the bug is in one of the states in *States-D*, the bug is eligible for the smell analysis. If so, we check if there exists any version control activity in the bug history.

No Comment Bugs: If the bug is in one of the states in *States-B*, the bug is eligible for the smell analysis. If so, we check if there exists any comment activity in the bug history.

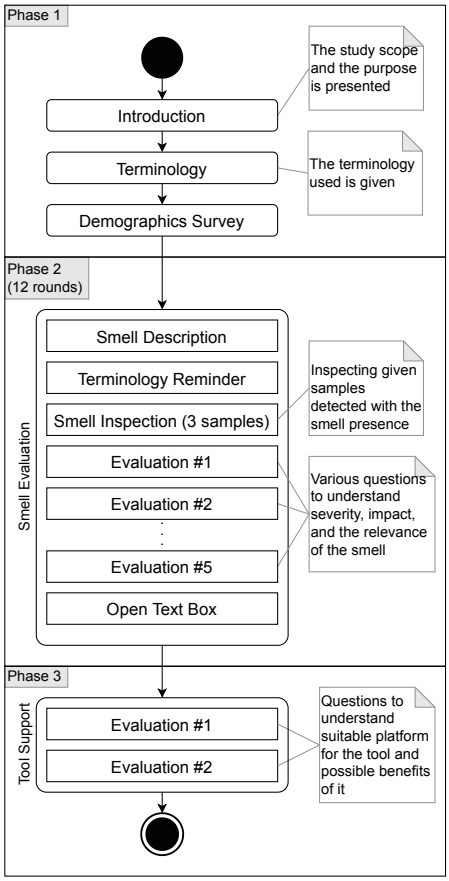
Non-Assignee Resolver of Bug: If the bug is assigned and in one of the states in *States-B*, the bug is eligible for the smell analysis. If so, we check if the assignee is the person who resolved the bug.

Closed-Reopen Ping Pong: We inspect the bug history and check if there are any transitions from a state in *States-B* to a state in *States-A*. This transition is classified as a reopen activity.

We share the source code of the implementation for the sake of reproducibility of the work[4].

[3]https://www.atlassian.com/software/jira

[4]https://doi.org/10.6084/m9.figshare.16822024



Rounded Box: Page, Rectangle Box: Section in a page.

Figure 1: The Survey Design.

3.4 Survey Design

We conducted a survey to understand the BT smell concept from developers' point of view. It was designed to provide answers for the research questions RQ2, RQ3, and RQ4. We present the details of the survey in this section.

3.4.1 Survey Content. We aimed to understand the perception of developers related to the smells as indicated in the RQ2. More specifically, we wanted to collect developer views on the importance and the severity of the smells. We prepared the survey for this purpose with the available smell results from the selected projects. We designed the survey as in Figure 1 (inspired from [11]). We share the anonymized survey[5] to ensure the transparency of our work as much as possible.

Introduction Page - We present the context of the work we are running. Also, we include brief information related to the BT process on this page, along with our reasons for conducting the survey.

Terminology Page - This page contains the terminology that we use in our study. We expect a participant to read the text to get familiar with the terminology and better understand the questions

[5]https://doi.org/10.6084/m9.figshare.16821358

Table 2: Smell occurences in bug reports for each project.

Smell Name	Project-A			Project-B			Project-C			Project-D		
	False	True	NA	False	True	NA	False	True	NA	False	True	NA
Bugs Assigned to a Team	93.9%	0.0%	6.1%	59.5%	0.1%	40.4%	89.9%	0.0%	10.1%	91.9%	0.0%	8.1%
Closed-Reopen Ping Pong	95.0%	5.0%	0.0%	96.2%	3.8%	0.0%	90.9%	9.1%	0.0%	95.9%	4.1%	0.0%
Ignored Bugs	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%
Missing Environment	93.3%	6.7%	0.0%	89.9%	10.1%	0.0%	73.8%	26.2%	0.0%	80.5%	19.5%	0.0%
Missing Priority	100.0%	0.0%	0.0%	40.2%	59.8%	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%
Missing Severity	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%	0.0%	100.0%	0.0%
No Comment Bugs	39.9%	36.4%	23.7%	58.6%	24.0%	17.4%	60.1%	35.3%	4.6%	53.7%	24.3%	22.0%
No Link to Commit	34.4%	22.9%	42.8%	29.8%	12.5%	57.7%	44.5%	12.8%	42.7%	23.1%	14.1%	62.8%
Not Referenced Duplicates	5.5%	0.0%	94.5%	16.2%	0.0%	83.8%	12.1%	0.1%	87.8%	15.2%	1.1%	83.7%
Reassignment of Bug	95.2%	4.8%	0.0%	96.6%	3.4%	0.0%	92.5%	7.5%	0.0%	94.3%	5.7%	0.0%
Non-Assignee Resolver	62.9%	10.1%	27.0%	42.0%	9.4%	48.6%	67.6%	19.4%	13.0%	43.4%	25.0%	31.6%
Unassigned Bugs	58.0%	1.4%	40.6%	38.2%	6.6%	55.1%	59.7%	1.4%	38.9%	40.5%	1.1%	58.4%

presented in Phase 2 of the survey. We acknowledge that it is not easy to remember new terms and concepts right away. So, we replicate the text from this page on every page of *Phase 2* to prevent misunderstandings.

Demographics Survey Page - A participant indicates their experience level (as months or years) in the software development domain and the company. This information is required to distinguish junior developers' answers from seniors. Besides, the participant indicates their job title in a free text format. Developers in managerial positions may have a different perception of the smells compared to individual contributors. Lastly, we ask participants to indicate the way they interact with YouTrack. Developers' perception of smells might depend on what kind of tasks they usually perform. A developer using the platform to verify bugs could find the smells related to bug report field values (e.g., *Missing Priority, Missing Severity*) more critical than *Missing Link to Commit* smell.

Smell Evaluation Page - We present the main content of the survey in *Phase 2*, as 12 consecutive pages. We customized the template page for each smell. It starts with a brief smell description and contents of the *Terminology Page*. Then, the participant is asked to inspect three bug report samples identified with the smell and decide whether the smell exists (binary response format). We provided screenshots and URL links of the samples. Afterwards, there are five evaluation questions to understand developers' perception of the smell. The first two questions ask whether the smell and the samples outline a deviation from the best practices in the BT process. The third one is a Likert scale question to rate the severity of the smell. In the fourth one, we want developers to think about the possible impacts of the smell and indicate their opinion via multiple selections or the free text field. Lastly, we ask about the usefulness of the automatic detection of the smell in a Likert scale question. We placed an optional text box at the end of the page to let developers state their opinions as they like.

Tool Support Page - In *Phase 3*, we designed two questions to measure the helpfulness of a tool that can detect smells. The former is to find a suitable context for the tool (i.e., whether detection should be within or out of the BTS). The latter one is to explore the potential benefits of such a tool.

3.4.2 Pilot Run. Before sending the survey to the target participants, we ran a pilot survey to correct and improve the survey

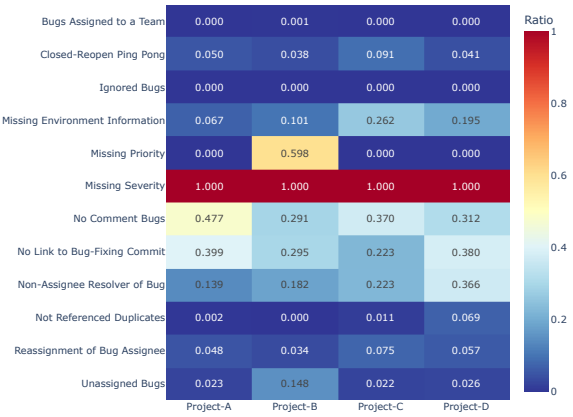


Figure 2: Incidence of smells in target projects.

instruments. The pilot survey included two graduate students at Bilkent University and one developer not working in the selected projects at YouTrack. The respondents reported their estimated completion times and grammar mistakes. We corrected some texts in the survey along with the received feedback.

3.4.3 Target Participants. Our target audience was the developers of the selected projects. The rough number (due to confidentiality) of developers are indicated in Table 1. We sent messages to Slack channels of the projects with a call to participate in our survey. In the message, we indicated that the survey takes approximately 20-30 minutes. We sent two reminders to increase the participation rate. We received 24 responses in total. The distribution of the responses are 1 (Project-A), 8 (Project-B), 4 (Project-C), and 11 (Project-D).

4 RESULTS

This section reports the numerical and statistical values that we obtained for all of the research questions.

4.1 Occurrence of the Smells (RQ1)

Our implementation of smell detection is based on the codebase by the authors of the taxonomy study [18]. They utilized Perceval [7] to download data and used the data format provided by the tool.

However, as the bug report data format of YouTrack is not the same as Perceval, we adapted the smell detection code to YouTrack. We use three labels for the smell existence in a bug report. *Not Applicable* (NA), if the bug report did not meet the condition for the smell analysis. If the bug report is eligible for the smell analysis, we label it *True* in case there is a smell, and *False* otherwise. We obtained the labels for each smell for every bug report in the selected projects and reported the values in percentage terms in Table 2. For confidentiality reasons, we cannot provide the exact number of labels for the smells.

Similar to [18], we created a heatmap representation of the values presented in Table 2, as in Figure 2. We constructed the heatmap according to the *Smell Ratio* parameter. This number is calculated for each project and smell by the ratio of the total number of bug reports identified with the smell (the values in the *True* column) to the total number of bug reports that are eligible for the particular smell analysis (the values in *False* and *True* columns). The calculation of the *Smell Ratio* ignores the values in the *NA* column because those values aggregate the number of bug reports that are not eligible for the related smell analysis. The *Missing Severity* smell occurs in every bug report of all the projects, as the severity field is not used at all. Some smells, in particular, *No Comment Bugs, No Link to Bug-Fixing Commit, Non-Assignee Resolver of Bug*, appear more frequently than the rest.

4.2 Developers' Views on the Smell Definition and Detection Methodology (RQ2)

4.2.1 About Smell Definition. We asked participants to answer in a binary reply whether the smell definition indicates a suboptimal practice. We show the answers for each project in Table 3 and aggregate the answers in Figure 3. For 7 out of 12 smells, 62.5 to 91.7 percent of the participants recognized that the smells outline a deviation from a best practice. The remaining smells, *Unassigned Bugs, Missing Severity, Non-Assignee Resolver of Bug, Reassignment of Bug Assignee,* and *No Comment Bugs* are rather subject to discussion whether they are actually undesirable. The participants rejected the definition of the *No Comment Bugs*'s smell in almost 60% of cases.

4.2.2 About Detection Methodology. The survey participants inspected the bug reports that were labeled with each of the smells. The responses consisted of either YES, NO, or OTHER choices. To develop a statistical interpretation, we classified the contents inside the OTHER answers into either YES, NO, or when we could not find explicit wording to N/A. By omitting the responses with the N/A category, we aggregated the YES and NO answers for each smell. Table 4 shows the ratio of agreement of the developers. We placed hyphens for the smells that do not occur in the projects. Since there was only one participant from Project-A, there is a 100% agreement score for this project.

Based on the samples shown in the survey, the participants' views vary according to the projects they are working in. The participants of Project-B approved the detection methodology of *Reassignment of Bug Assignee, Non-Assignee Resolver Bug, Bugs Assigned to a Team, Unassigned Bugs, No Link to Bug-Fixing Commit, Missing Severity,* and *Missing Priority* smells with YES ratio varying from 68.2 up to 87.5 percent. In contrast, Project-C participants did not find the detection of *Non-Assignee Resolver of Bug* smell accurate but agreed on the detection of *Not Referenced Duplicates* with a 100.0% ratio. The participants of Project-D found the detection of *Not Referenced Duplicates* inaccurate, unlike Project-C participants. These different thoughts on the detection of the same smell hint at the project-specific handling of the smells.

Table 3: Project-specific percentages of developers' view on the smell definition and its scope

Smell Name	Project-A		Project-B		Project-C		Project-D	
	Yes	No	Yes	No	Yes	No	Yes	No
Bugs Assigned to a Team	100.0%	0.0%	37.5%	62.5%	75.0%	25.0%	72.7%	27.3%
Ignored Bugs	100.0%	0.0%	75.0%	25.0%	75.0%	25.0%	72.7%	27.3%
Missing Environment	0.0%	100.0%	50.0%	50.0%	50.0%	50.0%	81.8%	18.2%
Missing Priority	0.0%	100.0%	87.5%	12.5%	75.0%	25.0%	72.7%	27.3%
Missing Severity	0.0%	100.0%	50.0%	50.0%	75.0%	25.0%	54.5%	45.5%
No Comment Bugs	100.0%	0.0%	25.0%	75.0%	25.0%	75.0%	54.5%	45.5%
No Link to Commit	100.0%	0.0%	75.0%	25.0%	100.0%	0.0%	90.9%	9.1%
Not Referenced Duplicates	100.0%	0.0%	87.5%	12.5%	100.0%	0.0%	90.9%	9.1%
Reassignment of Bug	0.0%	100.0%	37.5%	62.5%	25.0%	75.0%	63.6%	36.4%
Closed-Reopen Ping Pong	100.0%	0.0%	75.0%	25.0%	75.0%	25.0%	54.5%	45.5%
Non-Assignee Resolver	100.0%	0.0%	62.5%	37.5%	50.0%	50.0%	45.5%	54.5%
Unassigned Bugs	100.0%	0.0%	50.0%	50.0%	100.0%	0.0%	45.5%	54.5%

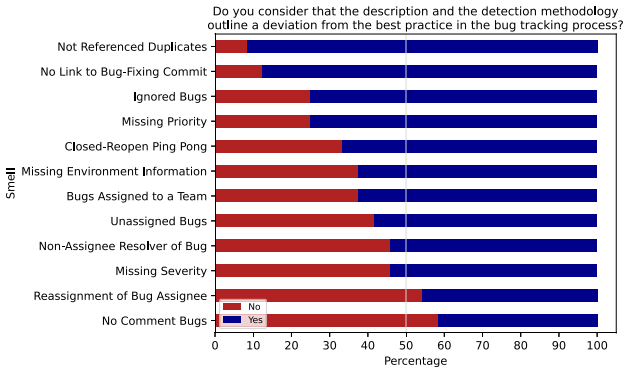


Figure 3: Cumulative representation of developers' view on the smell definition and its scope.

4.2.3 Highlights of Feedback. The smell evaluation pages in the survey consisted of two free text fields to receive developers' feedback related to the practices outlined by the smells and their detection. We observed that the participants presented their ideas sometimes separately and sometimes in a single comment. To provide a complete view, we combined the highlights related to the smell content and its detection methodology. The considerations related to the content were either about not finding the smell description as a deviation from the best practice, or doubts related to the applicability of the smell in practice. We present our findings for the applicable smells.

Unassigned Bugs - Some participants noted that this smell is quite natural in their workflow because the initial handlers of bug reports are support engineers. They explained that support engineers may resolve bugs that do not require a change in the product, and they do not assign themselves to the bugs. So, such occurrences are actually not considered as smells in their workflow. Another concern is about mass bug fixes. Participants note that sometimes mass bug

Table 4: Agreement on the smell detection method

Smell Name	Project-A		Project-B		Project-C		Project-D	
	Yes	No	Yes	No	Yes	No	Yes	No
Bugs Assigned to a Team	-	-	86.4%	13.6%	-	-	-	-
Closed-Reopen Ping Pong	100.0%	0.0%	50.0%	50.0%	75.0%	25.0%	78.8%	21.2%
Ignored Bugs	-	-	-	-	-	-	-	-
Missing Environment	100.0%	0.0%	70.0%	30.0%	72.7%	27.3%	48.5%	51.5%
Missing Priority	-	-	77.3%	22.7%	-	-	-	-
Missing Severity	100.0%	0.0%	77.8%	22.2%	66.7%	33.3%	57.6%	42.4%
No Comment Bugs	100.0%	0.0%	66.7%	33.3%	54.5%	45.5%	84.8%	25.2%
No Link to Commit	100.0%	0.0%	83.3%	16.7%	100.0%	0.0%	78.8%	21.2%
Not Referenced Duplicates	100.0%	0.0%	11.1%	88.9%	100.0%	0.0%	45.5%	54.5%
Reassignment of Bug	100.0%	0.0%	87.5%	12.5%	88.9%	11.1%	78.8%	21.2%
Non-Assignee Resolver	100.0%	0.0%	87.0%	13.0%	36.4%	63.6%	63.6%	36.4%
Unassigned Bugs	100.0%	0.0%	68.2%	31.8%	100.0%	0.0%	71.4%	28.6%

fixes or refactoring may resolve the bugs practically. They indicated that there is no added value in setting an assignee in such cases.

Bugs Assigned to a Team - Participants noted that assigning teams to bugs has a practical use. It is an intermediate step to determine which set of people (team) in the project should be pinged. They argued that the assignment of a team does not necessarily mean losing responsibility. Teams assign individuals related to the bug afterward.

Missing Priority and Missing Severity - We present the points related to these smells together. Priority and severity of a bug are distinct but relatable concepts. The responses revealed that the participants may use these terms interchangeably or provide joint insights to both. Some participants suggested to remove the priority field. They explained that the priority field can be abused by individuals who think their bugs are always of a high priority. Another point is about filling priority during sprint meetings. Some participants mentioned that picking bugs in Sprint meetings and prioritizing a small subset is a common practice. So, bug reports with empty priority fields do not necessarily mean they will never be filled.

Another comment warned about assigning a default value to the priority. It is noted that bug reports containing a priority field with a default value are indeed not prioritized. Moreover, the participant indicated that the detection consists of false negatives due to default priority values. The participant suggested that using default values could be a smell on its own.

Some participants were arguing that *Missing Severity* smell is irrelevant to their project because they do not use that field, and it is a decision of their team. A considerable number of participants noted that they use the priority field for both priority and severity. Some participants explained that they tried using severity, but it did not help to improve the process. It was also suggested that filling priority or severity fields should not be mandatory as a project setting since it may frustrate developers.

Missing Environment Information - Some participants made the distinction between customers and internal colleagues related to this smell. They noted that customers might file new bug reports, and they cannot be forced to fill in environment information. They argued it would be an annoying experience otherwise. On the other hand, in the case of a colleague filing a bug report, they indicated it is a better practice to fill environment information fields.

We received feedback related to the detection methodology of this smell as well. One comment suggested the smell is relevant but we should consider restricting the detection to the resolved bug reports. The idea behind it is that the environment information of a resolved bug report is known because someone should have worked on it. Simultaneously, one participant mentioned that if a bug is marked as *incomplete*, it is nonsense to check the environment fields since there is no sufficient information on the bug. Another comment suggested that our detection consisted of false positives because some of the environment information was present in the bug report's comment section.

Reassignment of Bug Assignee - Some participants indicated that reassignments happen due to the nature of bug management. One reason is based on the need for inspection of a bug by several developers due to the bug's complexity. Another cause is an adapted workflow — after a design of the solution is realized, the bug is transferred to a developer who will implement the design. One other reason is related to the workflow again. Some teams employ *dispatching* that bug report is first assigned to the related team's leader and passed to the target developer afterward. One participant made the distinction between bug analysis and fixing. The point was that reassignments during bug analysis are acceptable, but they are problematic amid bug fixing activity. Another participant underlined a tricky point about developers leaving the company. In such a situation, the bugs must be handed over to their successors.

No Link to Bug-Fixing Commit - Participants suggested that if a bug report has a link to another bug, such as *fixed by*, then it should not be identified as a smell. The main work is already conducted in the linked bug. Besides, some participants indicated that links to commits are sometimes mentioned in the bug report's comments section.

Not Referenced Duplicates - Participants indicated that the samples in the survey had duplicate links, but they were visible only to their team.

No Comment Bugs - A significant number of participants argued that if a bug report does not contain any comments, the bug is filed nicely. It is an indication of a high-quality bug report instead of a smell in the bug tracking process. Also, they claimed that expecting developers to leave comments to bug reports might be frustrating.

Related to the detection methodology, some participants suggested restricting the detection to the bugs in the states that require an explicit explanation for being in that state. They exemplified that the developer who realizes the bug state transition to *obsolete* or *declined* should comment on the reasoning.

Non-Assignee Resolver of Bug - Participants underlined that not every terminal state should be used in the detection of this smell. Even though someone else is the assignee of a bug report, they considered it all right when another developer (e.g., support engineer) resolves the bug as *duplicate*, *obsolete*, or any state that does not require a change in the codebase.

Closed-Reopen Ping Pong - Some participants considered that reopening is an inherent activity in the BT process, making it unavoidable. Another view is that the ping pongs could be an alert sign not for the BT process but the bug fixing process and project codebase. One participant noted that they reopened the bug report for the problems occurring in different environments. In this case, reopening is their BT process management practice.

Related to the detection methodology, one participant mentioned the number of reopenings. The participant found it fine to have a

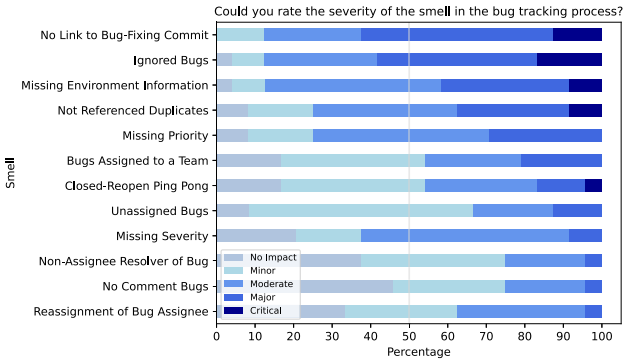


Figure 4: Developers' severity perception for each smell.

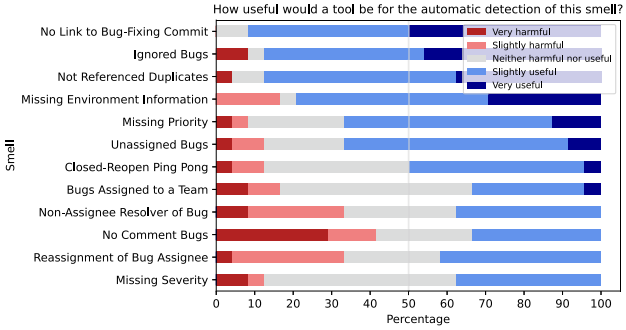


Figure 5: Usefulness rating of a potential tool with automatic smell detection feature.

single reopen. However, considered the excessive numbers a smell. Also, a significant number of participants suggested determining a time threshold between the state transition activities to be more conservative and prevent false-positive detections. Some participants suggested excluding the bug reports with resolution states that potentially indicate missing information related to the bug, such as *incomplete*.

Ignored Bugs - Participants' feedback about this smell was about the detection methodology. Some participants indicated that their project consists of some bugs that float in YouTrack. Even though such bugs exist, they criticized that our detection methodology raises false-negative results by only checking if an activity exists or not. They noted that the actions of support engineers mislead the detection methodology.

4.3 Developers' Severity Perception of Smells (RQ3)

The survey consisted of a Likert-scale question related to the perceived severity of each smell. The participants evaluated the severity by selecting one of *no impact (0), minor (1), moderate (2), major (3),* or *critical (4)* levels. Figure 4 presents the severity ratings for each smell. 62.5 and 58.3 percent of the participants identified the severity of *No Link to Bug-Fixing Commit* and *Ignored Bugs* smells either *major* or *critical*, respectively. On the other hand, 62.6 to 75.0 percent of the participants considered *Reassignment of Bug Assignee, No Comment Bugs, Non-Assignee Resolver of Bug,* and *Unassigned Bugs* smells the least severe by rating them as either *no impact* or *minor*. Another observation is that the median severity for 33.3 percent of the smells is *moderate* and for 50.0 percent of the smells is *minor*. So majority (10 out of 12) of the smells have median severity of either *minor* or *moderate*.

4.4 Usefulness of Smell Detection Tool (RQ4)

We asked participants to evaluate the usefulness of a hypothetical tool providing an automatic detection service of the smells. The evaluation was based on a Likert scale with five levels that are *very harmful (-2), slightly harmful (-1), neither harmful nor useful (0), slightly useful (1),* or *very useful (2)*. Figure 5 shows the percentages for each of the smells. We identify the most useful detection would be on *No Link to Bug-Fixing Commit*, and *Ignored Bugs* smells. Besides, the automatic detection of half of the smells is found to have *slightly useful* median value. On the other hand, 33.3 to 41.7 percent of the participants rated the detection of *Reassignment of Bug Assignee, Non-Assignee Resolver of Bug* and *No Comment Bugs* smells either *slightly* or *very harmful*.

We included a question in the survey asking about the suitable platform for a potential tool that provides an automatic smell detection feature. The question had three options. To account for a functionality that is included in BTS, we provided *within BT platform* option. A detection tool as a separate software in any environment is included in *out of BT platform* option. Lastly, *other* is provided for any ideas that we could not think of. The majority of the participants, 80.0 percent, found *within BT platform* suitable for such a tool. One participant indicated their own opinion in *other* option as the "embedded in business communication tools such as Slack".

Lastly, in the survey, we asked about the possible benefits of a tool detecting the BT process smells. We provided two options and received one distinct answer from the participants. Participants think that the potential benefit is two-fold. The first one is to *"gain insights about the BT process quality of the project"*. The second one is to *assess bottlenecks in the BT process*. Also, one participant indicated that some BT process smells indicate *smells* in other processes such as software development.

5 DISCUSSION

5.1 Improving Detection Methodology

We received feedback from the survey participants to potentially increase the smell detection accuracy. We discuss the comments related to the states of bug reports in Section 5.2. In this part, we are more interested in conceptual improvements that could be made to the smell detection methodologies. We curated the following points that could potentially improve the detection accuracy.

The smell detection for the linked bug reports should be based on the parent bug rather than the child one. In particular, checking the bug reports with link types *duplicate of* for *Missing Priority, Missing Severity,* and *Missing Environment Information* could lead to false positives. Developers may ignore filling the fields of the child bug or not mind some activities as the bug has a parent. Moreover, if a bug contains *fixed by* link, it seems unnecessary and incorrect

to check if the bug contains *No Link to Bug-Fixing Commit* because the problem is solved by the work conducted in the parent bug.

The detection of *Closed-Reopen Ping Pong* smell is subject to improvement as well. Incorporating the time threshold between consecutive state changes between resolved and unresolved states would decrease the false positive detections. Developers may unintentionally realize a state transition and revert it. Moreover, instead of a single ping pong, involving a maximum number of ping pongs in the analysis could lead to better detections. We note that determining the time threshold and a maximum number of ping pongs are subject to analysis and not easy to determine.

When developers leave a company, their bug reports are handed over to other developers. Detection methodology for *Reassignment of Bug Assignee* should be able to ignore those cases. A possible solution is integrating replacement lists into the detection algorithm and ignoring the related occurrences during the detection.

We think that participants misinterpreted the *Ignored Bugs* smell. The main point of this smell is to spot if developers conduct any activity on a bug in the first six months of its submission. The participants possibly perceived the smell coverage differently. They might have thought that the smell checks if a bug was forgotten after the last conducted activity and left unresolved. This perception probably hints at a new smell that we can call *Sleeping Bugs*.

5.2 Tailoring Smell Context and Detection Methodology

Participants acknowledged that some smells are actual deviations from the best practice. However, they indicated including or excluding some states in the detection methodology. Before running the smell analysis, we checked all of the available states in each of the projects. Besides, we created a map of all the state transitions in bug reports to see how developers use bug states in practice. We realized that the detection of smells according to the Jira conventions as presented in [18] is not applicable for YouTrack due to the merged use of state and resolution of a bug report. We came up with different clusters of states for accurate detection. We did not indicate our state clusters to the participants in the survey content since it would make the survey content-intensive.

According to the received feedback and statistical results, we concluded that differentiation of the state types was the correct approach for analysis in YouTrack. One should thoroughly understand every project and its particular paradigm before running a smell detection. Moreover, we deduce that in-person exploration of a project with its actual users is more beneficial and required. It is necessary to have a solid understanding of the use of bug states and bug report fields. The essence of smell detection is not to dictate every BT process to be in line according to a particular frame. Instead, it is to increase the process quality and traceability of any taken action in parallel with the principles. Thus, one should analyze the BT process with its authentic executors and tailor the detection methodologies accordingly.

5.3 Smell Detection Tool

We presented the participants' view on a tool for automatic smell detection in Section 4.4. We see a potential benefit of developing such a tool based on the survey data. We determined two essential features that tools detecting the BT process smells should possess. The first one is supporting to tailor the smell detection methodology. If this feature is missing, practices or habits specific to companies or teams would be ignored by the tool. Ignoring the context-specific information could cause generating false negative and false positive results. The second essential feature is providing a visual and statistical interpretation of the smell data. One benefit of this feature is that developers could gain insight into their BT process (see Section 4.4). Another use of visual or statistical interpretation is to reveal bottlenecks or suboptimal practices in the BT process.

6 THREATS TO VALIDITY

Internal validity reflects the potential threats related to the design and the results.

Bug States: Projects settings can be customized in YouTrack. The customization applies to the bug states as well. We observed different sets of states in the projects. We classified their semantic meanings into four groups to make accurate smell detections. During the classification, we checked the bug life cycles of the projects, analyzed the state transition diagrams happening in practice, and contacted team leads whenever required. Even though we conducted the classification activity with the utmost caution to mitigate this threat, there could still be differences between our perception of the states and their meaning from the developers' point of view in practice.

Team Practices: Several teams develop a single project, and each team may follow different rules while tracking bugs. Teams use the same project in the BTS. Different policies within the same project may cause false-positive smells. Some practices (e.g., not using priority or severity fields, assigning a team to bug reports) cannot be considered smells for some specific teams as those are the team's rules. However, using a single project in the BTS prevents developing team-specific detection methodologies. It is not clear to find out which teams work on which bug reports.

Survey: Misinterpreting the questions in the surveys is possible due to grammatical or semantic errors. We conducted a pilot survey to correct and clarify such cases. The participants might consider the repetitive nature of *Phase 2* of the survey exhausting and got bored over the iterations. This is a potential threat for us to receive inaccurate answers. Moreover, the participants may hesitate to state their opinions for a process that they are individually involved in. As a mitigation strategy, we anonymized the survey participation and left providing emails for further contact optional.

External validity reflects the threats on the general applicability of the results. We analyzed four projects to get a broader view of the prevalence of the smells in an industrial setting. We acknowledge that the analysis of four projects may not be enough and more investigation is required. Besides, the results may vary among different companies. However, we believe the results will be beneficial for both academia and industry.

7 CONCLUSION

Our empirical analysis revealed that the smells have visible occurrences in the inspected projects. We suggest that practitioners should evaluate the conformance of a BT process with the best practices according to the particular dynamics of the inspected process.

These dynamics include team-specific rules, company-wide conventions, and the used BTS. There is a consensus among developers that smells describe suboptimal processes, as our survey results suggest. However, some smells turned out to be controversial, such as No Comments Bugs and Reassignment of Bug Assignee. We concluded that there is room for improvement in the smell detection methodologies and stated our suggestions. Developers' perception of smell severity varies among smell types, and the survey revealed that the most severely perceived smell is No Link to Bug-Fixing Commit. We speculate that a BT process smell detection tool, possibly embedded in the BTS, would contribute to teams' and companies' understanding and evaluation of their BT processes.

Our future work agenda comprises investigating particular details about the smells, such as potential root causes, and interpreting differences in smell occurrences among projects. We also plan to conduct follow-up interviews with practitioners to investigate deviations from best practices.

8 ACKNOWLEDGEMENTS

This study was partially supported by The Scientific and Technological Research Council of Turkey (TUBITAK) 1505 program (Project Number:5200078). The authors thank Katerina Koshchenko for her comments on this work.

REFERENCES

[1] Anahita Alipour, Abram Hindle, and Eleni Stroulia. 2013. A contextual approach towards more accurate duplicate bug report detection. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 183–192. https://doi.org/10.1109/MSR.2013.6624026

[2] Nuri Almarimi, Ali Ouni, and Mohamed Wiem Mkaouer. 2020. Learning to detect community smells in open source software projects. *Knowledge-Based Systems* 204 (2020), 106201.

[3] Raymond P. L. Buse and Thomas Zimmermann. 2012. Information Needs for Software Development Analytics. In *Proceedings of the 34th International Conference on Software Engineering* (Zurich, Switzerland) *(ICSE '12)*. IEEE Press, 987–996.

[4] Jeffrey C. Carver, Oscar Dieste, Nicholas A. Kraft, David Lo, and Thomas Zimmermann. 2016. How Practitioners Perceive the Relevance of ESEM Research. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. 10 pages. https://doi.org/10.1145/2961111.2962597

[5] Gemma Catolino, Fabio Palomba, Andy Zaidman, and Filomena Ferrucci. 2019. Not all bugs are the same: Understanding, characterizing, and classifying bug types. *Journal of Systems and Software* 152 (2019), 165–181. https://doi.org/10.1016/j.jss.2019.03.002

[6] Emre Doğan and Eray Tüzün. 2021. Towards a taxonomy of code review smells. *Information and Software Technology* (2021), 106737. https://doi.org/10.1016/j.infsof.2021.106737

[7] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. 2018. Perceval: software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 1–4.

[8] Pascal Gadient, Mohammad Ghafari, Patrick Frischknecht, and Oscar Nierstrasz. 2019. Security code smells in Android ICC. *Empirical software engineering* 24, 5 (2019), 3046–3076.

[9] Vahid Garousi, Markus Borg, and Markku Oivo. 2020. Practical relevance of software engineering research: synthesizing the community's voice. *Empirical Software Engineering* 25, 3 (Mar 2020), 1687–1754. https://doi.org/10.1007/s10664-020-09803-0

[10] Marion Gottschalk, Mirco Josefiok, Jan Jelschen, and Andreas Winter. 2012. Removing energy code smells with reengineering services. *INFORMATIK 2012* (2012).

[11] Mrio Hozano, Alessandro Garcia, Baldoino Fonseca, and Evandro Costa. 2018. Are You Smelling It? Investigating How Similar Developers Detect Code Smells. *Inf. Softw. Technol.* 93, C (Jan. 2018), 130–146. https://doi.org/10.1016/j.infsof.2017.09.002

[12] Md Shariful Islam, Abdelwahab Hamou-Lhadj, Korosh Koochekian Sabor, Mohammad Hamdaqa, and Haipeng Cai. 2021. EnHMM: On the Use of Ensemble HMMs and Stack Traces to Predict the Reassignment of Bug Report Fields. In *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 411–421. https://doi.org/10.1109/SANER50967.2021.00045

[13] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. Predicting the severity of a reported bug. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 1–10. https://doi.org/10.1109/MSR.2010.5463284

[14] Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E. James Whitehead Jr. 2013. Does Bug Prediction Support Human Developers? Findings from a Google Case Study. In *Proceedings of the 2013 International Conference on Software Engineering*. 372–381.

[15] Dongyu Liu, Micah J. Smith, and Kalyan Veeramachaneni. 2020. Understanding User-Bot Interactions for Small-Scale Automation in Open-Source Development. In *Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems*. 1–8. https://doi.org/10.1145/3334480.3382998

[16] David Lo, Nachiappan Nagappan, and Thomas Zimmermann. 2015. How Practitioners Perceive the Relevance of Software Engineering Research. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 415–425. https://doi.org/10.1145/2786805.2786809

[17] Ran Mo, Yuanfang Cai, Rick Kazman, and Lu Xiao. 2015. Hotspot patterns: The formal definition and automatic detection of architecture smells. In *2015 12th Working IEEE/IFIP Conference on Software Architecture*. IEEE, 51–60.

[18] Khushbakht Ali Qamar, Emre Sülün, and Eray Tüzün. 2021. Towards a Taxonomy of Bug Tracking Process Smells: A Quantitative Analysis. In *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. 138–147. https://doi.org/10.1109/SEAA53835.2021.00026

[19] Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern Code Review: A Case Study at Google. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. 181–190.

[20] Tushar Sharma and Diomidis Spinellis. 2018. A survey on software smells. *Journal of Systems and Software* 138 (2018), 158–173.

[21] Mozhan Soltani, Felienne Hermans, and Thomas Bäck. 2020. The significance of bug report elements. *Empirical Software Engineering* 25, 6 (2020), 5255–5294.

[22] Anton Strand, Markus Gunnarson, Ricardo Britto, and Muhmmad Usman. 2020. Using a Context-Aware Approach to Recommend Code Reviewers: Findings from an Industrial Case Study. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice*. 1–10. https://doi.org/10.1145/3377813.3381365

[23] Gábor Szoke, Csaba Nagy, Lajos Jeno Fülöp, Rudolf Ferenc, and Tibor Gyimóthy. 2015. FaultBuster: An automatic code smell refactoring toolset. *2015 IEEE 15th International Working Conference on Source Code Analysis and Manipulation (SCAM)* (2015), 253–258.

[24] Ahmed Tamrawi, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2011. Fuzzy Set and Cache-Based Approach for Bug Triaging. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*. 365–375. https://doi.org/10.1145/2025113.2025163

[25] Ulisses Telemaco, Toacy Oliveira, Paulo Alencar, and Don Cowan. 2020. A Catalogue of Agile Smells for Agility Assessment. *IEEE Access* 8 (2020), 79239–79259.

[26] Yuan Tian, David Lo, and Chengnian Sun. 2013. DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis. In *2013 IEEE International Conference on Software Maintenance*. 200–209. https://doi.org/10.1109/ICSM.2013.31

[27] Qasim Umer, Hui Liu, and Inam Illahi. 2020. CNN-Based Automatic Prioritization of Bug Reports. *IEEE Transactions on Reliability* 69, 4 (2020), 1341–1354. https://doi.org/10.1109/TR.2019.2959624

[28] Eva Van Emden and Leon Moonen. 2002. Java quality assurance by detecting code smells. In *Ninth Working Conference on Reverse Engineering, 2002. Proceedings.* IEEE, 97–106.

[29] Stephane Vaucher, Foutse Khomh, Naouel Moha, and Yann-Gaël Guéhéneuc. 2009. Tracking design smells: Lessons from a study of god classes. In *2009 16th Working Conference on Reverse Engineering*. IEEE, 145–154.

[30] Mairieli Wessel, Igor Steinmacher, Igor Wiese, and Marco A. Gerosa. 2019. Should I Stale or Should I Close? An Analysis of a Bot That Closes Abandoned Issues and Pull Requests. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. 38–42. https://doi.org/10.1109/BotSE.2019.00018

[31] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou, Zhongxuan Luo, and Xindong Wu. 2015. Towards Effective Bug Triage with Software Data Reduction Techniques. *IEEE Transactions on Knowledge and Data Engineering* 27, 1 (2015), 264–280. https://doi.org/10.1109/TKDE.2014.2324590

[32] Meng Yan, Xin Xia, Yuanrui Fan, David Lo, Ahmed E. Hassan, and Xindong Zhang. 2020. Effort-Aware Just-in-Time Defect Identification in Practice: A Case Study at Alibaba. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1308–1319. https://doi.org/10.1145/3368089.3417048

[33] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643. https://doi.org/10.1109/TSE.2010.63