Exploring the Role of Automation in Duplicate Bug Report Detection: An Industrial Case Study

Malte Götharsson
Karl Stahre
University of Gothenburg
Gothenburg, Sweden
[gusgothama,gusstahrka]@student.gu.se

Gregory Gay
Francisco Gomes de Oliveira Neto
Chalmers and University of Gothenburg
Gothenburg, Sweden
greg@greggay.com,francisco.gomes@cse.gu.se

ABSTRACT

Duplicate bug reports can increase technical debt and tester workload in long-running software projects. Many automated techniques have been proposed to detect potential duplicate reports. However, such techniques have not seen widespread industrial adoption. Our objective in this study is to better understand how automated techniques could effectively be employed within a tester's duplicate detection workflow. We are particularly interested in exploring the potential of a human-in-the-loop scenario where tools and humans work together to make duplicate determinations.

We have conducted an industrial case study where we characterize the current tester workflow. Based on this characterization, we have developed Bugle—an automated technique based on a complex language model that suggests potential duplicates to testers based on an input bug description that can be freely reformulated if the initial suggestions are irrelevant. We compare the assessments of Bugle and testers of varying experience, capturing how often—and why—opinions might differ between the two, and comparing the strengths and limitations of automated techniques to the current tester workflow. We additionally examine the influence of knowledge and biases on accuracy, the suitability of language models, and the limitations affecting duplicate detection techniques.

CCS CONCEPTS

• **Software and its engineering → Software post-development issues**; **Collaboration in software development**; **Software verification and validation**; **Software maintenance tools**.

KEYWORDS

Bug Reports, Duplicate Bug Reports, Automated Duplicate Bug Report Detection, Natural Language Processing, Software Testing

ACM Reference Format:
Malte Götharsson, Karl Stahre, Gregory Gay, and Francisco Gomes de Oliveira Neto. 2024. Exploring the Role of Automation in Duplicate Bug Report Detection: An Industrial Case Study. In *Proceedings of 5th International Conference on Automation of Software Test (AST 2024).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

AST 2024, April 2024, Lisbon, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

During the development and evolution of a software project, developers, testers, and users of the project[1] will inevitably witness *bugs*—cases where the actual behavior of the software deviates from expectations [6]. These incidents are then documented by filing a *bug report* describing the deviation and the context that it occurred within [21]. Bug reports play an essential role in the improvement of software quality [21].

As the lifespan and popularity of a project grows, so does the number of bug reports—making the process of evaluating, prioritizing, and assigning bug reports time-consuming and laborious [1]. Often, there can be multiple bug reports describing the same issue, known as *duplicate bug reports* [11]. Duplicate reports are not always harmful, since the information contained in duplicates is often complementary [2, 7]. However, their continuing presence adds to the technical debt of the project—making it harder to manage the collective repository of bug reports, adding noise to analyses and prioritization efforts, and potentially wasting the time of testers in forcing them to examine potentially-solved issues. Therefore, there is a natural desire to *automatically* identify potential duplicates.

Over the past two decades, a number of automated detection approaches have been proposed in research literature (e.g., [5, 8, 9, 11, 15, 20]). However, we have observed that the use of such tools has not become standard in an industrial setting—instead, testers tend to make manual assessments based on keyword searches, existing knowledge, and intuition.

Our objective in this study is to better understand how automated techniques could effectively be employed within a tester's duplicate detection workflow. We are particularly interested in exploring the potential of a human-in-the-loop scenario where tools and humans work together to make duplicate determinations [5]. Therefore, we are interested in characterizing the current tester workflow, comparing the assessments of automated techniques and testers, capturing how often—and why—automated techniques and testers' opinions might differ, and comparing the strengths and limitations of automated techniques to the current tester workflow.

To reach this objective, we conducted a case study at Test Scouts—a testing consulting company—where we performed an interview study to understand the typical workflow of a tester evaluating duplicate bug reports. Based on the results of this interview study, we have developed Bugle—a tool for duplicate bug report detection[2]. Bugle utilizes language models, based on the SBERT architecture [14], to identify bug reports with a high semantic textual similarity to the bug description provided as a query by the tester.

[1]For simplicity, we refer to all bug report submitters as "testers" in this study.
[2]Bugle is available from https://github.com/Maltegos/Thesis/.

Testers are then able to confirm or reject the identified potential duplicates and reformulate their query at any time. In this study, we deploy Bugle in the described human-in-the-loop scenario—offering the potential efficiency benefits of automation through Bugle while also taking advantage of testers' knowledge and intuition in the loop to further improve the effectiveness and efficiency of the duplicate detection process.

We evaluated Bugle in an observation study where we assessed the accuracy of human testers and Bugle in identifying duplicate reports. We also assessed how often—and why—testers and Bugle made different assessments. On average, Bugle retrieves the seeded duplicate report based on the participants' queries in 94.44% of cases. The participants' average accuracy in correctly identifying the duplicate from the retrieved reports was 75.00%. However, on average, the participants do not agree with Bugle's recommendations of potential duplicates in 38.90% of cases. The most common reasons for disagreement are semantic differences in descriptions, insufficient information in either the provided or recommended descriptions, and overlooking recommendations made by the tool. We additionally examine the influence of knowledge and other biases on the accuracy of duplicate detection, the suitability of language models for duplicate detection, and the limitations affecting duplicate detection techniques.

This study contributes to our understanding of current duplicate detection workflows employed in industry, the strengths and limitations of automated assessment tools, and the reasons why human judgement may differ from automated assessments. We also offer Bugle as an example of a human-in-the-loop detection tool that combines the benefits of automation and human experience and intuition. These contributions offer important lessons to inform future automated approaches to duplicate detection as well as future studies on duplicate detection workflows and processes.

2 BACKGROUND AND RELATED WORK

In the context of software, a "bug" refers to an unexpected defect, fault, flaw, or imperfection [6]. The objective of a bug report is to describe a bug accurately enough so that it can provide testers with sufficient information for it to be resolved. Typically, a bug report consists of a subset of the following items: *steps to reproduce*, *observed* and *expected behavior*, *stack traces*, *test cases*, *screenshots*, *code examples*, and a *summary* [21]. Once a bug report has been filed, it is typically stored in a repository system, such as Bugzilla or Jira [9]. Due to the absence of widely-accepted industry standards for bug report creation and their natural language format, identical issues may be described differently, resulting in *duplicate* bug reports. Duplicates can comprise a significant portion of filed reports; for instance, the Mozilla and Eclipse projects have experienced duplicate rates of up to 30% and 20%, respectively [2].

The problem of detecting duplicate bug reports has been investigated since the early 2000s. A systematic mapping study covering research from 2007 to 2017 categorized approaches into three types: top-N/ranking-based methods providing ranked lists of potential duplicates, classification approaches predicting based on historical data, and decision-making approaches comparing pairs of bug reports [11]. Notably, the study revealed that top-N/ranking-based approaches generally demonstrated superior performance.

Many early approaches were based on similarity measurements. Hiew proposed a recommendation system based on tf-idf and cosine similarity for duplicate detection [9]. Runeson et al. compared similarity metrics such as cosine, Dice, and Jaccard distance, finding that cosine similarity and focusing on textual summaries and descriptions and discarding other attributes of the reports yielded the highest accuracy [15]. Wang et al. report improved accuracy by adding execution information to the textual summaries and descriptions [19]. However, the feasibility of this approach was limited, since special tools are often needed to collect the required information. Sun [17] echoes that execution information is often hard to collect or not available.

Recent approaches incorporate machine learning and natural language processing. Kang improved recall over tf-idf using the model Doc2Vec [11], as the neural network could infer context from text segments rather than comparing individual words. Xie et al. combined convolutional neural networks with domain-specific features extracted from bug report repositories [20]. Recerntly, Isotani [10] and Carneiro [4] both perform duplicate detection using complex language models based on the SBERT architecture [14]. Haering et al. also use deep learning to map app reviews to bug reports to incorporate user feedback into the bug fixing process [8].

Chaparro et al. suggested that testers should be involved in duplicate bug report detection, proposing an approach where—after the top-N duplicate suggestions are retrieved—a tester can reformulate the report description and re-query the set of reports [5]. This human-in-the-loop approach yielded significantly improved accuracy after reformulation of queries.

This study builds on, and extends, past research. Based on Kang's findings [11], we have developed a a top-N/ranking-based approach. As suggested by Runeson [15], Wang [19] and Sun [17], we focus on natural language elements of bug reports, as those yield the most context for duplicate detection. Based on the success of ML-based approaches [4, 8, 10, 11, 20], as well as cosine similarity [8, 9, 15], we utilize both SBERT models and cosine similarity in our approach. We particularly extend past work by examining the role and judgements of human testers in the process of duplicate detection, with a particular focus on cases where testers and tools disagree and cases where automated methods result in false recommendations.

3 METHODS

Our aim is to enhance the integration of automated duplicate bug report detection techniques into testers' workflows. To achieve this, we examined existing literature and typical tester workflows, extracting requirements for an effective automated detection tool. We propose Bugle, a natural language processing-based tool, and assess its performance by having a group of testers evaluate its suggestions. Our analysis focuses on understanding the strengths and limitations of automated tools, as well as the reasons behind discrepancies between automated techniques and testers' opinions. We specifically seek to address the following questions:

- RQ1: What is the typical workflow of testers when evaluating potential duplicate bug reports?
- RQ2: How often do testers disagree with the recommendations made by Bugle?

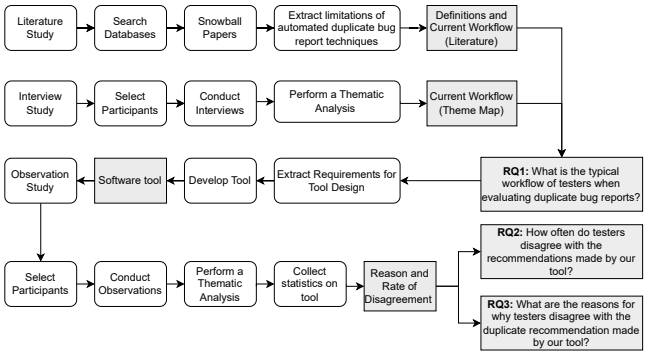

Figure 1: Overview of the study, including activities (rounded rectangles) and corresponding deliverable (shaded).

Table 1: Overview of interview participants.

ID	Current and Past Roles	Yrs. Experience
P1	Test Lead, Test Architect, Project Manager	24
P2	Test Lead, Embedded Test, Automated Test	12
P3	Test Lead, Test Architect	19
P4	Test Lead, Manual test, Automated Test	9
P5	Test Lead, Requirement Analyst	22
P6	User Acceptance Testing, Test Management	16
P7	Test Lead, Automated Test, Test Education	26

- RQ3: What are the reasons why testers disagree with Bugle's recommendations?

This study is conducted as an exploratory case study at Test Scouts, a consultant company with approximately 20 employees that has provided software testing services since 2015. Test Scouts works with agile test management, test automation, and development of test environments. An exploratory case study is suitable since our objective is to gain new insights from a realistic situation [16]. An outline of the study is shown in Figure 1. This section discusses the methodology for addressing the research questions. An overview of the tool, Bugle, is provided in Section 4.

3.1 Duplicate Detection Workflow (RQ1)

Data Collection: To characterize the typical workflow of testers when evaluating duplicate bug reports, we: (i) analysed past research, and (ii) performed semi-structured interviews with seven testers at Test Scouts who have taken part in duplicate bug report assessment. An overview of the participants is provided in Table 1, whereas our interview guide is shown in Table 2. Participants were sought for consent prior to recording interviews, with explicit information provided about anonymization procedures, and they retained the option to withdraw from the study at any point.

Data Analysis: We conducted a thematic analysis of the transcribed interviews, following the approach of Braun and Clarke [3]—familiarising ourselves with the data, generating initial codes, searching for themes, reviewing themes, defining and naming themes, and producing the report. The first two authors worked individually to extract relevant parts of the transcripts (i.e., codes). Then, together, those authors checked for agreement to ensure consistency in the interpretation. After analyzing our consistency across interviews, we had 71% agreement, which all authors agreed was acceptable to move forward with independent analysis of the remaining transcripts. We iteratively grouped those codes into themes until no additional logical groupings could be made.

Table 2: Interview questions.

No.	Question
Q1	How many years of experience do you have in the IT industry?
Q2	How many years of experience do you have as a software tester?
Q3	What is your role as a tester?
Q4	In your experience, what is the typical workflow of working with bug reports?
Q5	In your experience, what is the typical workflow of working with duplicate reports?
Q6	What tools and techniques do you use for detecting duplicate bug reports?
Q7	What does the workflow of that tool and/or techniques look like?
Q8	How are the results given by your current technique evaluated?
Q9	What role does your intuition play in the process of evaluating duplicates?
Q10	Do you question the results given by the duplicate bug report tool/technique?

Table 3: Overview of observation participants.

ID	Current and Past Roles	Yrs. Experience
	Group 1 (Exposure to Dataset, Duplicate Exp.)	
H1	Test Engineer, Test Framework, Production Test	5 years
H2	Test Lead, Embedded Test, Automated Test	12 years
H3	Test Engineer, Performance tester	3 years
	Group 2 (No Exposure, Duplicate Exp.)	
H4	Test Lead, Manual test, Automated Test	9 years
H5	System tester	4 years
H6	User Acceptance Testing, Test Management	16 years
H7	Test Lead	7 years
	Group 3 (No Exposure, No Duplicate Exp.)	
H8	Test Automation	<1 year
H9	Test Automation	<1 year

3.2 Tester and Tool Consensus (RQ2, RQ3)

Data Collection: To answer RQ2 and RQ3, we conducted an observational study at Test Scouts to analyze the frequency of disagreement between testers and Bugle on whether a bug report description is a duplicate as well as the *reasons why* they disagree. Bugle was pre-loaded with a dataset of 1548 industrial bug reports from an issue tracking repository for one of Test Scouts' clients. We conducted this analysis with nine Test Scouts employees, described in Table 3. Participants were divided into groups based on whether they had past experience with (1) issues in the dataset and (2) with duplicate detection. Group 1 had prior experience with both. Groups 2 and 3 had no prior exposure to the dataset. Group 2 had previous experience with duplicate detection and Group 3 did not have any experience.

To simulate a scenario in which the tester was given information about an issue and could be tasked with searching for potential duplicates using the tool, we prepared four issue descriptions to

be used during the observations. For three of these, we chose real bug descriptions from the client dataset, and represented them in the format of (i) one short text outlining a matching requirement, (ii) one short text outlining the expected behavior, and (iii) one short text outlining the observed behavior. These textual issue representations were constructed to be vague enough not to give away the true duplicate, but still contained enough information to be able to identify them as duplicates if inspected carefully. We also constructed one issue description in the same format that did not have any matching duplicate bug report in the dataset to create a situation where the tool would give an incorrect recommendation.

These issue descriptions were printed on paper and given to the testers during the observations so they would not simply copy and paste the text into the tool. Rather, these served as visual input for the participant to construct their own semantically-similar input description based on the provided description. We asked the participants to (i) read and familiarize themselves with the issue descriptions one by one, (ii) formulate their own input to the tool, and (iii), judge the relevance of the recommendations given by the tool and provide an explanation for their reasoning.

We consider an issue description to be labelled as a duplicate if a participant (i) linked the issue description to a recommended bug report, (ii) added complementary information to a recommended bug reports, or (iii), expressed the need to investigate a recommended bug reports further (e.g. by communicating with the creator of a recommended duplicate). Conversely, we consider an issue description as being labelled as not being a duplicate if a participant recommended creating a new report.

Observations were made following a think aloud protocol. During the observations, we collected qualitative and quantitative data while letting the participants use the tool. We recorded the audio, video and screen of the participants while they solved each of the four tasks. During the observations, the participants had a high degree of awareness of being observed and the researchers had a high degree of interaction with the participants [16].

When participants judged the relevance of the recommendations, we collected quantitative data regarding the agreement or disagreement between the participant and the tool for each issue description. In addition, we measured the accuracy of the tool in retrieving correct duplicates based on the participants' input and the participants' accuracy in detecting duplicates based on the tool's output. The possible outcomes of the observations were labeled as:

(1) True Positive: tool gave a correct recommendation.
   (a) *TPA*: Participant agreed with the recommendation.
   (b) *TPD*: Participant disagreed with the recommendation
(2) False Positive: tool gave an incorrect recommendation.
   (a) *FPA*: Participant agreed with the recommendation.
   (b) *FPD*: Participant disagreed with the recommendation.

Since one of the issue descriptions did not correspond to any actual issue in the dataset, all *FPD* for that issue were marked as correctly identified since the participants successfully identified the issue as a new bug report.

Data Analysis: We calculated the accuracy of participants as:

$$accuracy_{participant} = \frac{TPA + FPD}{TPA + TPD + FPA + FPD} \quad (1)$$

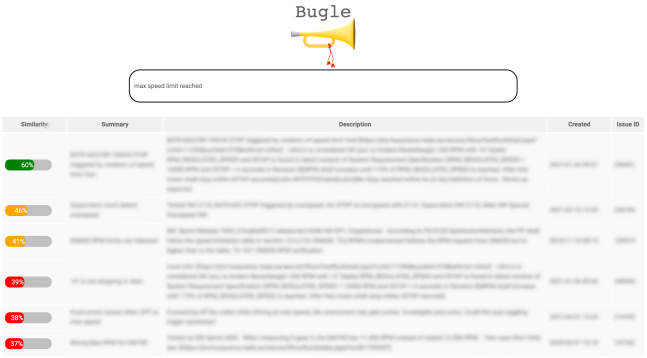

Figure 2: The user interface of Bugle. The output text is blurred to maintain confidentiality.

We also calculated the corresponding accuracy of Bugle in retrieving the correct duplicates as follows:

$$accuracy_{tool} = \frac{TPA + TPD}{TPA + TPD + FPA + FPD} \quad (2)$$

Furthermore, we measured the rate of disagreement between participants and the tool as:

$$disagreement = \frac{FPD + TPD}{TPA + TPD + FPA + FPD} \quad (3)$$

We conducted a thematic analysis of the qualitative data to answer RQ3, again following the approach outlined by Braun & Clarke [3]. The first two authors extracted quotes and behaviors relevant to answering the research question from recordings and notes from the observation sessions. During this process, we particularly paid attention to situations where the participants disagreed with the recommendations made by the tool. We individually extracted and compared relevant codes with 81% agreement. We then iteratively grouped codes into overarching themes.

4 BUGLE - DESIGN AND IMPLEMENTATION

Based on the results of RQ1 (Section 5.1) and key findings from related work (Section 2), we have developed Bugle—a stand-alone web application for automated duplicate bug report detection. Bugle is a ranking-based approach [11] that allows testers to provide a textual issue description as input and outputs the six most similar issue descriptions from a repository of bug reports.

Bugle uses Python and the Django web framework. Figure 2 displays Bugle's user interface, where a search bar is used to enter a query. Based on past findings [15, 17, 19] and the workflow derived from answering RQ1, the input is expected to conform to the textual *description* field of a bug report. Bugle lists the six bug reports from the dataset with the highest contextual similarity[3]. This result is updated in real-time based on the current contents of the query. As suggested by Chaparro et al. [5], the tester can reformulate the input query at any time.

Bug Report Data and Semantic Textual Similarity: Bugle is data-agnostic, i.e., it functions independently of the bug report dataset used. Common issue tracking systems, such as Bugzilla

[3]The choice to display exactly six reports was made based on suggestions from Test Scouts testers, who felt that more reports displayed at once would be distracting.

Table 4: Evaluation of open source datasets on pre-trained models.

Model	Eclipse			Mozilla Firefox		
	No. of duplicates	Identified Duplicates	Accuracy	No. of Duplicates	Identified Duplicates	Accuracy
paraphrase-Mini LM-L3-v2	14315	3016	21.07%	10808	754	6.98%
multi-qa-MiniLM-L6-cos-v1	14315	4724	33.08%	10808	1284	11.88%
all-mpnet-base-v2	14315	5011	35.26%	10808	1480	13.69%

or Jira [9], can export bug report data as CSV files that can be integrated into the datasets used by Bugle.

Input bug descriptions are encoded into sentence embeddings using SentenceTransformers, a Python framework based on SBERT [14]—a model architecture tuned for common natural language processing tasks such as semantic textual similarity. Similar embeddings have been employed successfully in past research on duplicate detection [4, 10]. The input embeddings are then measured in relation to the sentence embeddings produced for the entire dataset of bug reports, after which the shortest cosine similarity distance is used to retrieve the semantically closest bug reports for the user. Cosine similarity has been found to be more accurate for comparing bug reports than other similarity measurements [8, 9, 15].

Model Selection: We evaluated three contrasting pre-trained language models, chosen to reflect different combinations of size and performance. The first model, *paraphrase-MiniLM-L3-v2*, produces the embeddings in the shortest amount of time and is smallest in size, but is considered to have the lowest average accuracy of the considered models [13]. The second model, *multi-qa-MiniLM-L6-cos-v1*, is only 25% slower than *paraphrase-Mini LM-L3-v2* [13], but its accuracy in detecting semantic similarity in natural language is also improved by 25% on average. The third model, *all-mpnet-base-v2*, was largest in size and five times slower than *multi-qa-MiniLM-L6-cos-v1*. However, it is considered to have the highest accuracy in detecting semantic textual similarity in natural language.

We select models that correctly detect duplicate bug reports in two large open source bug report datasets, both containing a ground truth for duplicates. The datasets were the 115,814 bug reports filed between 1999 and 2013 in the Mozilla Firefox project, as well the 85,156 bug reports between 2001 and 2013 in the Eclipse project, with a duplicate percentage of 30.9% and 16.9% respectively [18]. We produced sentence embeddings for the bug descriptions for both datasets. Table 4 shows resulting accuracy.

Since Bugle recommends the six most similar bug reports, we considered Bugle to have detected a duplicate if either the *issue ID* or the linked *duplicate ID* for any of the six recommendations was the issue ID of the bug report whose issue description was used as input. In some cases, duplicates in these datasets were chained together—e.g., issue X was linked as a duplicate of issue Y, and issue Y was linked as duplicate of issue Z, but there was no direct link between issues X and Z. If Bugle recommended issue Z when evaluating the description of issue X, no duplicate would be marked as detected, even though Bugle potentially found one. This means that the resulting accuracy might be higher than listed, since the datasets do not link duplicates bidirectionally.

Before measuring the accuracy of Bugle, the datasets needed to be pre-processed. The bug reports for the Eclipse project followed a format where the textual description was concatenated with additional pieces of information (e.g., comments, URLs, etc.).

Table 5: RQ1: Main themes found in the interview data.

Theme	Description
Knowledge	The types of knowledge utilized to identify duplicates.
Duplicate Identification	Practices and techniques used to identify duplicates.
Duplicate Management	The main steps of the workflow for evaluating duplicates, from bug report creation to linking and closing duplicates.

This information was concatenated in a semicolon-separated string, which made it possible to extract only the textual description. However, not all bug reports included the textual issue description before the first semicolon, which led to noisy data being encoded into sentence embeddings. The bug reports in the Mozilla Firefox dataset also needed to be pre-processed, but the textual description field did not follow a specific format, making the extraction of the pure textual description difficult. We filtered out bug reports that were particularly noisy, e.g., where the description field was empty or only contained stack trace information or URLs.

As shown in Table 4, *all-mpnet-base-v2* achieved the highest accuracy—13.69% and 35.26%—for the two datasets. This accuracy is relatively low. The primary reason for this is imperfect pre-processing [8, 20]. Many of the bug reports did not have semantically coherent textual descriptions, introducing significant noise into the embeddings. In order to improve the accuracy of the models further on these particular datasets, further effort needs to be expended on improving the pre-processing process.

As we were using a new dataset for evaluationat TestScouts based on their own bug report data, we decided that the current results were sufficient as a basis for selecting the *all-mpnet-base-v2* model for use in our final evaluation without spending further time pre-processing the open source datasets. The client dataset used to address RQ3 was pre-processed similarly to the Eclipse dataset—extracting the pure textual description from a concatenated string. However, the formatting was consistent and we were able to verify that pre-processing was able to successfully clean the data.

5 RESULTS

5.1 Duplicate Detection Workflow (RQ1)

The analysis of the tester's workflow resulted in three main themes (Table 5): Knowledge, duplicate identification, and duplicate management. These were then divided into nine sub-themes. An overview of all themes and sub-themes is shown in Figure 3.

Knowledge: An overarching theme across all interviews was the role played by different facets of the knowledge of the testers in identifying duplicates. As shown in Figure 3, knowledge manifested as *system knowledge*—specific knowledge about the system and

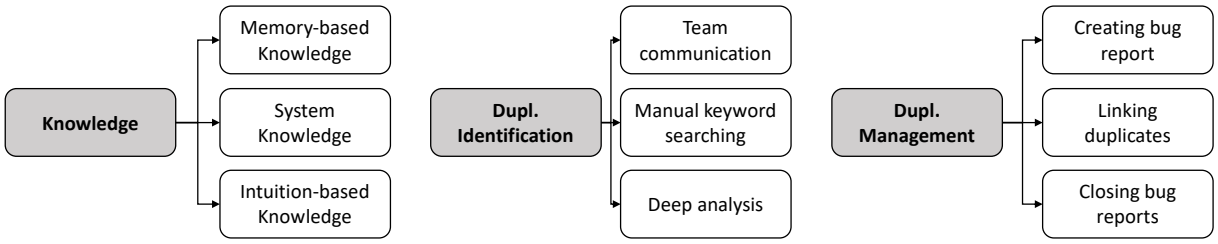

Figure 3: Themes and sub-themes pertaining the workflow of testers when analysing bug reports.

its functionality—*intuition-based knowledge*—gut feelings or intuitions that guided decision making—and *memory-based knowledge*—knowledge about previous bugs and specific behaviors derived from the experience of the tester.

The participants made frequent remarks about the importance of system knowledge in their workflow. Since bugs and the language used to describe them are highly system-specific, in-depth knowledge of the system-under-test provides the tester with the context and language necessary to evaluate duplicates.

> "You need to have a system understanding to realize that it is the same bug because people describe their issues differently, or in their own way." - Participant 7

In fact, participants also report remembering having seen similar bug reports in the issue tracking system, and let their memories and experiences guide them in their decision-making. Participants also often reported their basis for decision-making as being guided by intuition from prior experiences, but not necessarily specific to the current system-under-test (e.g., Participant 4).

> "The analysis team is the one that usually recognizes that we already have this issue, and it's really manual and from memory, which I must say is extremely fascinating." - Participant 7

> "Intuition, some kind of gut feeling, "I recognize this", and then you go home and search a bit." - Participant 4

Duplicate Identification: This theme relates to the practices and techniques used to identify duplicates in practice. We identified three high-level sub-themes: *manual keyword searching*, *deep analysis*, and *team communication* where the communication is mainly affected by the size of the team and the repository. One observation across all interviews was the absence of the use of any specific tool within the company for identifying duplicates. Instead, participants unanimously relied on manual searches using keywords or phrases in the issue tracking system to find duplicates. A quick keyword match typically sufficed to identify duplicates, and then the tester would avoid writing a new bug report.

> "Often in the tools I have worked with, you can search by titles, so then you think to yourself, "Hmm, I named this 'spelling mistake on the first page' " and then you start searching for 'spelling mistake'." - Participant 3

Sometimes, the participants reported a need for deeper analysis to be able to conclude whether the bug report is a duplicate, such as investigating the source code for a root cause that manifests throughout the system in different ways. Especially when the manual keyword search results in ambiguity, there may be a need for further analysis to establish certainty. The need for a deeper analysis is evaluated in relation to the potential risks associated with the bug under review—bugs considered high risk require deeper analyses before being dismissed as duplicates.

Team communication was also emphasized, recognizing that team members possess varying knowledge and insights into different parts of the system.

> "Before even going to the bug tool, you might write a quick message just to say: "Has anyone seen this issue?" " - Participant 6

Team and bug report repository size were recurring themes—as the system or team size grows, the reliability of testers' individual knowledge and ability to identify duplicates becomes more specific to their areas of responsibility. However, even in smaller teams, team communication is a common strategy. The importance of communication and the challenges introduced by large teams or large code or bug report repositories necessitates standards to be put into place regarding how team communication is conducted and—in some cases—how bug reports are written to ensure that duplicate detection is efficient.

> "The challenge is when there are more people, then you need to talk as a team and have a standard for how to report for the sake of simplicity for everyone." - Participant 5

Duplicate Management: This theme refers to the steps taken before and after duplicate identification. We identified the high-level sub-themes of *creating*, *linking*, and *closing bug reports*. Participants reported that the process used to create bug report matters when identifying duplicates. They stated that bug reports should be reported in a standardized manner or, at least, follow a pre-defined structure to proactively help in duplicate identification. Since the most useful aspects of a bug report for identifying duplicates are the text fields [15, 17, 19], standardizing the way that these textual inputs are formulated introduces consistency that helps the team.

> "If it's too difficult to find duplicates and it takes a lot of time, we sit down together and create some sort of standard for how we report. So that everyone follows the same way." - Participant 5

The general consensus across all participants was that confirmed duplicates should be linked together and the report containing the least amount of information should be closed. The issue tracking systems used in the organization allow different ways to do this—either by creating an actual link between two reports or by including the ID of the closed duplicate in the open report. Linking is important since a duplicate report might contain additional valuable information for addressing the bug.

> "Usually, you link them together, so if I say that A is a duplicate of B, I close A. But then on B, I can see that it has duplicates linked to it." - Participant 3

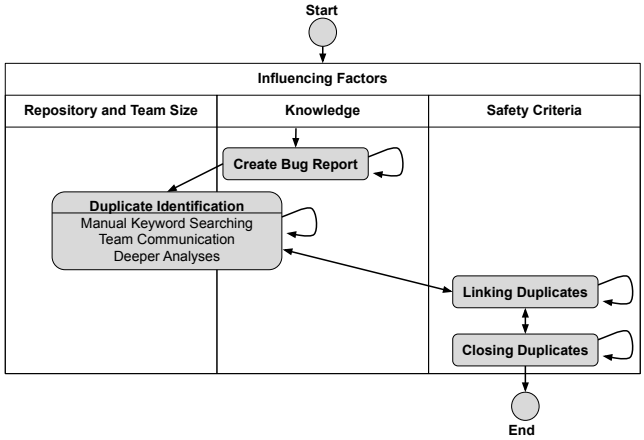

Figure 4: Typical workflow when evaluating duplicates.

Closing bug reports is encouraged as a way of maintaining the bug report repository—it is easier to keep track of open issues when there are fewer. As long as the closed duplicate is linked to the open report, closing is always preferred.

> "I like this aspect of closing things, because I think that if it is important, it will come up again, and I would rather have fewer tasks so that we can keep track of them." - Participant 3

The process of closing bug reports is dependent on the safety criteria within the organization—safety critical systems require more careful closing of duplicates, since a mistakenly closed bug report could create risk.

> "The reasons for keeping both defects open may vary, as they may not be exactly the same issue but very similar, and for various reasons, one may not want to close either of them because they are not the same." - Participant 4

There were differing views on who should be able to evaluate and close a duplicate. Some made the argument that only the initial creator of a report should be able to close it. Others argued that it is preferable to have few open issues, and that one should be able to close issues created by others as long as a link exists.

> **Duplicate Detection Workflow (RQ1):** The typical workflow—shown in Figure 4—consists of (i) creation of a bug report, (ii) using manual searches, team communication, or deep analysis to identify duplicates, (iii) linking duplicate reports together, and (iv) closing duplicates. Factors affecting the workflow include different forms of knowledge, team and repository size, and safety criteria of the organization.

Note that the individual steps of this process can be complex and involve iterative sub-processes. For example, bug reports can be iterated upon and refined by multiple testers before reaching their final form [12]. Similarly, duplicate detection, linking, and closing may involve more than one tester and may not always be unidirectional steps (i.e., previous steps may be revisited).

This workflow informed the design of Bugle. First, our results suggested that the bug description was the field that most aided testers in deciding if a report was a duplicate. In addition, that the current workflow primarily relied on keyword searches and tester knowledge suggested that Bugle should act as a "smart" keyword search based on semantic similarity rather than simple keywords.

Table 6: Average rate of disagreement, tool accuracy, and tester accuracy across all participants, and among each of the three groups (gr.) of participants.

	All Partic.	Gr. 1	Gr. 2	Gr. 3
Rate of Disagreement	38.90%	41.67%	43.75%	25.00%
Tool Accuracy	92.60%	91.67%	93.75%	100.00%
Tester Accuracy	75.00%	58.33%	75.00%	100.00%

Table 7: Performance of each participant and Bugle.

	Participant				Bugle	
ID	**Time to Eval.**	**Num. Queries**	**Acc.**	**Rate of Disag.**	**Time to Detect**	**Num. Queries**
H1	5:22	3.00	1/4	1/4	4:14	2.70
H2	2:55	2.75	3/4	2/4	1:32	1.70
H3	6:28	7.25	3/4	2/4	0:34	1.00
H4	2:37	5.00	3/4	1/4	1:42	4.70
H5	2:01	4.75	3/4	2/4	0:33	2.30
H6	2:31	3.75	4/4	1/4	0:25	2.00
H7	3:58	4.50	2/4	3/4	1:05	2.00
H8	2:13	3.75	4/4	1/4	0:53	2.70
H9	2:17	3.00	4/4	1/4	0:38	2.01
Avg.	**3:22**	**4.19**	**75.00%**	**38.90%**	**1:17**	**2.35**

5.2 Rate of Disagreement with Tool (RQ2)

In Table 7, we list data on each participant, including the average time to make an evaluation (minutes:seconds), average number of queries to the tool before the user decided that they had identified a duplicate, accuracy of their determinations, and rate of disagreement with Bugle. For Bugle, we also list the average time and number of queries before Bugle returned the duplicate in its recommendations. Table 6 then lists the average rate of disagreement, tool accuracy, and participant accuracy across the full group of participants as well as for the three groupings of participants.

> **Rate of Disagreement with Tool (RQ2):** On average, Bugle locates the duplicate report based on the participants' queries in 94.44% of cases. The participants' average accuracy in correctly identifying the duplicate was 75.00%. On average, the participants do not agree with Bugle's recommendations in 38.90% of cases.

Interestingly, Table 6 reveals large differences across groups, with the *most* experienced participants yielding the worst results. We will discuss this observation further in Section 6.2.

5.3 Reasons for Disagreement (RQ3)

The thematic analysis of the data gathered during the observations resulted in three main themes with nine high-level sub-themes and six low-level sub-themes that summarize the reasons for the disagreements made during the observations. As described in Table 8, the primary themes include *semantic differences in descriptions*, *insufficient information*, and *overlooking recommendations*.

Table 8: Description of main themes identified during the observational study.

Theme	Description
Semantic Differences	The provided issue description and recommended bug report description were reported to have different meanings.
Insufficient Information	The provided issue description or recommended bug report description were reported as not containing enough information to make a judgement.
Overlooking	Skipping or not fully reading a description.

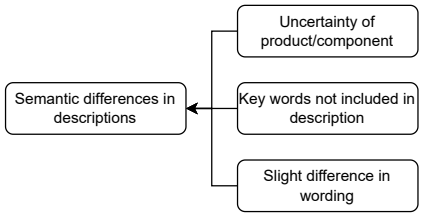

Figure 5: Sub-themes related to semantic differences.

Semantic Differences in Descriptions: The most common reason for disagreement was that the participants reported there to be semantic differences between the issue descriptions and the recommendations made by the tool. In Figure 5, we show the derived theme and corresponding sub-themes.

In some cases, the differences were subtle—e.g. one participant remarked that the first description mentioned "flipping" of a switch rather than "holding down"—as it was described in the highest ranked recommendation. The participant argued that there could be two slightly different behaviors and would have raised a new bug report rather than trusting the recommendation to be a duplicate.

Another description detailed a case where a motor ran at top speed, after which it slowed to a halt, which caused a user interface to shut down. The tool recommended a nearly-identical issue description with the difference being that—instead of coming to a halt—the motor nearly halted and then slightly sped up again, which also led to a user interface shutting down.

Most often, disagreements based on semantic differences were concerned with potentially important keywords not being present in the description of the recommended reports.

> "It didn't retrieve "temperature" *and* "LED" so that's why I saw that the combination didn't exist. The recommendations didn't give both parts of "LED" and "temperature"." - Participant 8

Insufficient Information: The second most common reason for disagreement was that the participants reported not having sufficient information to determine whether a recommendation was a duplicate or not, and therefore, opting to disagree with the recommendation. In Figure 6, we list the derived theme and its corresponding high-level and low-level sub-themes.

This remark was more frequently made by more experienced participants and those with in-depth system knowledge. Participants with previous experience working with evaluating duplicate bug reports also remarked that they would like more specific pieces of information to determine with certainty that a recommendation

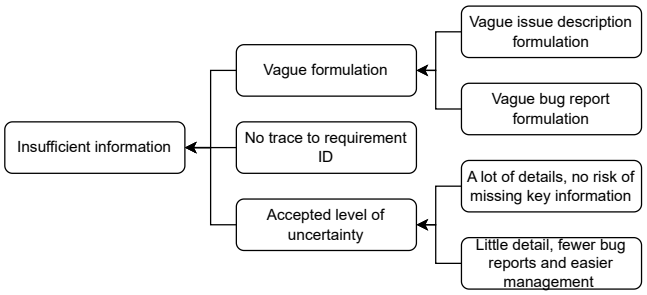

Figure 6: Sub-themes related to insufficient information.

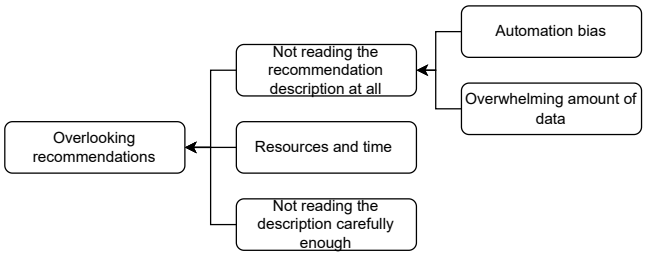

Figure 7: Sub-themes related to overlooking.

was a duplicate, including references to related requirements, the specific project the bug report belonged to, and the bug status.

Overlooking Recommendations: The third most common reason for disagreement was that participants overlooked a correct recommendation either because the participant skipped or missed important details from the recommendation. In Figure 7, we list the derived theme with its corresponding sub-themes.

On being asked why they disagreed with a correct recommendation, one participant re-read the description and acknowledged that it might actually be a duplicate and that the initial assessment had been made hastily. Other participants chose not to read the descriptions of certain recommendations at all, either because they were long or difficult to interpret or because the recommendation was not ranked as high as others retrieved by the tool.

> "Oh, this was a lot of text, that is useless. I don't want to read and understand that." - Participant 2

Reasons for Disagreement (RQ3): Common reasons for disagreement include *semantic differences in descriptions*, *insufficient information* in provided or recommended descriptions, and *overlooking recommendations* made by the tool.

6 DISCUSSION

6.1 Duplicate Detection Workflow (RQ1)

One insight was that—despite many automated techniques having been proposed in past literature—there is a lack of widespread commercial or open source tools available to testers working to identify duplicate reports. None of the participants, some of whom had more than two decades of experience in testing, reported having heard of such a tool. Instead, the typical activities conducted for duplicate identification are performed manually without tool support.

The interview data revealed a trade-off between the risk of simply creating a new bug report and the effort of first spending time looking for duplicates. The tolerance for technical debt in a team plays an important role in making that trade-off, as well as an assessment of the risk of falsely labeling a bug report as a duplicate. In safety-critical systems, there needs to be a high level of certainty before testers label a bug report as a duplicate due to the high risk associated with closing an prematurely. Some redundancy in the open bug reports may be deemed acceptable. For systems with lower risks associated with failures, lower technical debt may be more highly prioritized.

Participants expressed that such a tool would be of significant benefit to their workflow, with one estimating that the average cost per duplicate determination being around $1000. Effective tool support could reduce the time spent making determinations and increase the certainty in the determinations—impacting both the financial and technical debt impact of duplicate bug reports. It is not clear why effective tool support is not widely available. There could be a lack of awareness of existing tools, existing research tools may not effectively fit existing workflows, or existing tools may not be sufficiently robust. Future work should explore awareness and how to develop tools that effectively support developers.

6.2 Disagreement with Tool (RQ2, RQ3)

Contradictory findings regarding knowledge: One interesting observation from the study was an inverse relationship between a tester's knowledge and experience and successful duplicate detection using Bugle. More experienced testers tended to claim to need more information to determine whether a bug report recommended by Bugle was an actual duplicate or not, whereas their less experienced peers were successful in making the correct decision given only the limited amount of information they had been provided.

More experienced participants also commented on not having access to additional contextual information that they are used to having, impacting the level of certainty with which they could claim a recommendation to be an actual duplicate. More often than their less experienced peers, they would disagree with the recommendation of the tool and would have chosen to create a new bug report for their provided issue description due to uncertainty.

We hypothesize that the primary reason for the inverse relationship between knowledge and successful duplicate detection is that the more experienced participants tend to—by default—draw on their additional contextual knowledge and intuitions while less experienced participants can only base their opinion on the limited amount of information at their direct disposal.

In multiple cases, more experienced participants used keywords in their queries that were not derived from their provided issue description, but from their own domain knowledge. What is interesting is that—although the experienced participants had more information at their disposal—they were actually less successful than their least experienced peers in detecting true duplicates with the help of the tool, and had a higher average rate of disagreement.

More experienced participants were also potentially biased by their existing process of manual keyword searches. The input queries used by experienced participants tended to rely heavily on concatenating individual keywords, while less-experienced participants tended to formulated the query in the form of semantically coherent sentences. These participants tended to adapt their query formulations as free-flowing descriptions more naturally than their more experienced counterparts, which helps to explain their success in using the tool—whose model has been tuned specifically for semantic textual similarity. The tool design and intended use was explained to all participants, but the more experienced ones may have a more ingrained way of working.

These observations are based on a pool of only nine participants, split into three groupings. To confirm these findings and hypotheses, we would need a larger group of participants—something we plan to pursue in future work.

Bias: Participants made reference to additional biases during the observational study as well. In several instances, the similarity score and its coloring (green, yellow or red to indicate high or low similarity) was noted as affecting the participants' opinions about a recommendation being a likely duplicate or not. Some felt that particular similarity scores seemed to be too high or low relative to their expectations. In certain cases, this seemed to be a reason for not reading lower-ranked recommendations' descriptions, leading to participants choosing to create a new bug report for issues whose duplicate the tool had successful retrieved.

One participant stated that they had lost faith in Bugle after several attempts to find a duplicate had failed to retrieve anything of interest. The participant was asked for the reason behind their disagreement with the tool and subsequent choice to create a new bug report, after which they studied the recommended description more carefully and acknowledged that they had dismissed a very likely duplicate due to having underestimated Bugle's abilities after multiple failed attempts.

Suitability of language models for duplicate detection: Another topic for discussion is the question of whether a sentence transformer model such as *all-mpnet-base-v2* are actually necessary for detecting bug report duplicates—or if it is analogous to "killing a fly with a sledgehammer."

Some participants expressed skepticism that a tool such as Bugle could be more effective than their current process of keyword searches, while others expressed optimism. Participants suggested that Bugle would be potentially beneficial for less-experienced testers or when starting to work on a new project, as the ability to enter free-flowing text descriptions allows them to describe issues without the need to be familiar with domain-specific keywords.

Compared with static keyword searches, the power of semantic search lies in the ability of the model to detect not only synonyms, but also contextual similarity across sentences. This was demonstrated clearly in our study, as each issue description we provided to the participants was carefully reformulated from the original "duplicate" using multiple synonyms while still retaining the original semantic meaning. Duplicate detection would be unlikely under traditional techniques, e.g., those that compare vectors of terms.

Limitations of duplicate detection techniques: In conducting this study, we have observed several challenges to overcome in the design of future tools for duplicate detection.

One of the biggest current limitations for duplicate detection is the dependency on high-quality bug report data. In this study, we applied general pre-trained language models fine-tuned for semantic textual similarity search. However, a model trained specifically

on bug report data could potentially be much more accurate. Challenges in creating such a model include the amount, availability, and quality of data needed for training and the model's ability to generalize its performance across different product domains.

Commercial developers are often reluctant to share bug report data for reasons related to security, confidentiality, and reputation. Open source bug report datasets are available, such as those we used to evaluate models [18]. However, the quality of the bug report data varies widely both across and within projects and organizations, and the lack of a standard or well-structured format for bug reports—and lack of consistency in following a particular format even within some datasets—results in a need for extensive data pre-processing.

Improving organizational standards regarding bug reporting would likely improve model accuracy regardless of whether general models or models specifically trained on bug report data are used. Bug descriptions in natural language are extremely useful. However, guidance for how to structure a bug report and how to phrase discussions about bugs in a more consistent and clear manner could be useful for improving both tool and tester accuracy in identifying duplicates as well as improving the utility of the bug reports for the testers attempting to fix a bug.

Our study also highlights the power of a combination of an automated tool and tester opinion, in comparison to solely relying on automated duplicate detection. The combination of tool and tester eliminates false recommendations being automatically marked as duplicates, a common risk in past research [9], by allowing a tester to evaluate the recommendations made by Bugle. In addition, allowing testers to reformulate queries enabled effective and nuanced exploration of the bug report repository not possible purely through the use of automated tools, further confirming the findings of Chaparro et al. [5].

There is an important design decision to be made regarding the similarity threshold at which a bug report should be marked as a duplicate in a fully automated system. If the threshold is too high, true duplicates with lower similarity scores may be ignored, whereas a lower threshold may result in unique bug reports being marked as duplicates. Determining this score requires considering the safety criteria and tolerance for technical debt in the organization.

In a human-in-the-loop system, like Bugle, the actual decision on whether a report is a duplicate is outsourced to the tester. In such cases, there is an additional design decision to be made—how many bug reports to show the human tester. If a tester is presented with too many results, they may become overwhelmed and overlook important details in particular bug reports. However, if too few are shown, then a tester may never see the duplicate they are looking for. In our case, we elected to always show six recommendations, regardless of the actual score. However, there were cases where testers lost confidence because they had seen too many irrelevant reports. It is possible that a different setting would have been better—we will explore this topic further in future work.

6.3 Threats to Validity

Construct Validity: Testers could be too trusting of Bugle's recommendations, leading to a failure to fully utilize their own intuition. We mitigated this threat by explaining to all participants that the tool is merely providing suggestions and that we are interested in their ability to judge the relevance of the recommendations. We further explained to all participants that potential duplicates could be in any position in the ranked list, even with a low similarity score, and that the recommendations are based on the input description. Further, we utilize different participants for addressing RQ1 and RQ2–3. The latter participants could have a different workflow than those in the former group. However, as all participants are at the same company, we believe differences are minor.

Internal Validity: Bugle updates recommendations in real time while the tester enters a query. This may influence how testers formulate their query, leading to a different description than what would be entered into a static search. We mitigated this threat by recording sessions and asking participants to explain why and how they reformulated their queries.

External Validity: We had a relatively small pool of participants. Further, the number of participants in each group is unbalanced. Unfortunately, we were unable to involve more participants from Test Scouts. However, many of the testers had extensive experience and have worked at multiple companies. In addition, Test Scouts works with many partner organizations. Therefore, we believe that their perspectives and workflow are broadly representative of other testers. In addition, the contents of the dataset of bug reports may influence the generalizability of the results. However, (i) we used alternative open-source datasets to select the model used in Bugle, and (ii), the dataset employed in the observation study contains real-world industrial bug reports. Therefore, we believe that the results were sufficiently realistic. Finally, the actual number of duplicate issues was small in order to limit the time commitment of participants. However, we believe that we gathered sufficient data for an initial exploration. Future work should extend the study with additional participants from multiple companies and a larger pool of duplicate issues.

Reliability: The first two authors were present during all interviews and observations and coded data independently to avoid biased conclusions. All authors discussed and confirmed the codes, themes, and sub-themes to ensure coherency and agreement.

7 CONCLUSION

Our study aimed to enhance the integration of automated duplicate bug report detection techniques into a tester's workflow. Proposing Bugle as an automated tool to present relevant bug reports, we achieved an average success rate of 94.44% in locating duplicates based on participants' queries. While participants exhibited a 75.00% accuracy in identifying duplicates from the retrieved reports, disagreements with Bugle's recommendations occurred in 38.90% of cases, primarily due to semantic differences, inadequate information, and oversight.

These findings provide insights for future automated duplicate detection approaches and studies on duplicate detection workflows. Future research will expand our study to a broader tester group, focusing on how tool support can benefit both inexperienced and experienced testers. Further investigations will explore the impact of thresholds on the number of reports and minimum similarity scores, along with methods to enhance automated technique accuracy through data pre-processing and structured report writing.

Acknowledgements: We thank Test Scouts for their participation.

REFERENCES

[1] Thangarajah Akilan, Dhruvit Shah, Nishi Patel, and Rinkal Mehta. 2020. Fast detection of duplicate bug reports using LDA-based topic modeling and classification. In *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 1622–1629.

[2] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann, and Sunghun Kim. 2008. Duplicate bug reports considered harmful... really?. In *2008 IEEE International Conference on Software Maintenance*. IEEE, 337–345.

[3] Virginia Braun and Victoria Clarke. 2012. Thematic analysis. *APA handbook of research methods in psychology* 2 (2012), 57–71.

[4] Guilherme Carneiro, José Ferreira, Franklin Ramalho, and Tiago Massoni. 2023. Similar Bug Reports Recommendation System using BERT. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*. 378–387.

[5] Oscar Chaparro, Juan Manuel Florez, Unnati Singh, and Andrian Marcus. 2019. Reformulating queries for duplicate bug report detection. In *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 218–229.

[6] dictionary [n. d.]. Bug, Definition & Meaning, Merriam-Webster. https://www.merriam-webster.com/dictionary/bug

[7] Maximilian Flis. 2020. *Support Scrub Meetings in Distributed Teams by Detecting Duplicates of Software Defect Reports in Issue Management Systems*. Master's thesis. Dept. of Informatics, Technical Univ. of Munich, Munich, Bavaria, Germany.

[8] Marlo Haering, Christoph Stanik, and Walid Maalej. 2021. Automatically matching bug reports with related app reviews. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 970–981.

[9] Lyndon Hiew. 2006. *Assisted detection of duplicate bug reports*. Ph. D. Dissertation. University of British Columbia.

[10] Haruna Isotani, Hironori Washizaki, Yoshiaki Fukazawa, Tsutomu Nomoto, Saori Ouji, and Shinobu Saito. 2021. Duplicate bug report detection by using sentence embedding and fine-tuning. In *2021 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 535–544.

[11] Li Kang. 2017. Automated duplicate bug reports detection-an experiment at axis communication ab.

[12] Senthil Mani, Seema Nagar, Debdoot Mukherjee, Ramasuri Narayanam, Vibha Singhal Sinha, and Amit A Nanavati. 2013. Bug resolution catalysts: Identifying essential non-committers from bug repositories. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 193–202.

[13] Reimers Nils. 2022. SentenceTransformers Documentation. https://www.sbert.net/

[14] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. http://arxiv.org/abs/1908.10084

[15] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. 2007. Detection of duplicate defect reports using natural language processing. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 499–510.

[16] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14 (2009), 131–164.

[17] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 45–54.

[18] LogPAI Team. 2018. BugRepo. https://github.com/logpai/bughub

[19] Xiaoyin Wang, Lu Zhang, Tao Xie, John Anvik, and Jiasu Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *Proceedings of the 30th international conference on Software engineering*. 461–470.

[20] Qi Xie, Zhiyuan Wen, Jieming Zhu, Cuiyun Gao, and Zibin Zheng. 2018. Detecting duplicate bug reports with convolutional neural networks. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 416–425.

[21] Thomas Zimmermann, R. Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36 (09 2010), 618–643. https://doi.org/10.1109/TSE.2010.63