CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Case Study on the Limitations of Automated Duplicate Bug Report Detection

Bachelor of Science Thesis in Software Engineering and Management

MALTE GÖTHARSSON
KARL STAHRE

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

A Case Study on the Limitations of Automated Duplicate Bug Report Detection.

Supervisor: FRANCISCO GOMES & GREGORY GAY
Examiner: RICHARD BERNTSSON SVENSSON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023

A Case Study on the Limitations of Automated Duplicate Bug Report Detection

Malte Götharsson
University of Gothenburg
Software Engineering and Management BSc
Gothenburg, Sweden
Email: gusgothama@student.gu.se

Karl Stahre
University of Gothenburg
Software Engineering and Management BSc
Gothenburg, Sweden
Email: gusstahrka@student.gu.se

***Abstract*—Identifying duplicate bug reports is crucial in software development as it helps streamline the debugging process, reduce redundancy, and enhance overall efficiency. By addressing the challenges associated with existing automated techniques and leveraging testers' expertise, the tool proposed in this study aims to improve the accuracy of duplicate detection, saving valuable time and resources while ensuring that potential duplicates are not overlooked. While various automated techniques for identifying duplicate bug reports have been previously described in literature, they often result in false positives or identified duplicates that testers would not consider as such [8]. To address this we propose Bugle, a software tool that incorporates a state-of-the-art large language model and involves the opinions of testers when evaluating potential duplicate bug reports in real-time. Our approach leverages testers' tacit knowledge and intuition to improve the accuracy of duplicate detection and reduces the amount of false positives by letting the tester evaluate a ranked list of recommended duplicates with the highest semantic textual similarity. We evaluate the tool in a case study at TestScouts, a testing consulting company, and analyze the recommendations of the tool against the judgement of a group of testers at the company. Besides Bugle, we contribute to the state-of-the-art by incorporating testers' opinions and provide insights into the limitations of automated techniques for duplicate bug report detection.**

I. Introduction

Maintenance is an ongoing process in the software development life cycle (SDLC) where bug reports are used as input for testers to solve reported issues. As software grows, so does the number of reported bugs which makes the process of evaluating, prioritizing, and assigning bug reports to testers, known as bug triaging, time-consuming and laborious—leading to billions of dollars in cost of maintenance [1]. The problem of triaging large numbers of bug reports describing the same issue, referred to as duplicates throughout this study, has been an oft-treated subject in the software engineering literature since the early 2000s [9]. Large numbers of duplicates are not harmful per se, since the pieces of information contained in different duplicates often prove complementary to each other [2], [6]. The problem is rather one of accurately linking duplicates together in order to minimize the number of open issues in the repository—avoiding redundant work of bug triaging the same issue more than once. Several techniques exist to automate the identification of duplicate bug reports—drawing from the fields of information retrieval and machine learning. Even though the techniques used vary across the different approaches described in the literature, all utilize some variation of the following three-step process:

1) Extracting data from either bug report repositories or immediately from bug reports produced by reporters (the people submitting a bug report, referred to as testers throughout this study).
2) Using various systems, such as neural networks, and statistics, such as term frequency-inverse document frequency, in order to automate the identification of potential duplicates.
3) Using the identified duplicates in order to alleviate the workload of bug triagers.

Even though several automated techniques following these general steps have been described in the literature, they are not common practice in industry—instead, testers in industry usually rely on manual practices such as keyword searching, e.g. Ctrl+F on Windows, coupled with their contextual domain knowledge in order to find duplicates. In this study we seek to better understand what potential limitations there are in fully automated duplicate detection as compared to the manual practices commonly used today in industry. We propose a solution for the duplicate bug report problem based on a combination of the two approaches, that not only improves the identification process in practice through automation but that also further involves testers to take advantage of their professional opinions and tacit knowledge to further improve the duplicate detection process.

Our proposed solution is Bugle, a bug reporting tool that involves testers further by letting them evaluate duplicates automatically identified by a large language model specifically fine-tuned for detecting semantic textual similarity in the second step of the above process. In particular, the testers will be able to confirm or reject the identified duplicate in real-time. As testers possess a unique insight into the bug at the time of submitting the report, our approach will take advantage of their tacit knowledge and intuition in evaluating the relevance of the recommendations made by Bugle. The reason for this is that testers often have in-depth knowledge about the system-under-test and additional contextual information useful to be able to determine if an issue is a duplicate that is external to

the information usually contained in the bug reports that the automated system uses to make the decision.

We evaluated Bugle in a case study at TestScouts—a testing consulting company based in Gothenburg, Sweden—where we found that not only did it provide state-of-the-art accuracy in duplicate detection, it also seemingly eliminated the need for extensive contextual domain knowledge by the tester in the duplicate detection process, making it easier for less experienced testers to perform the task of successfully identify duplicates.

Our discussion focuses on what insights can be gained regarding the reliability of both the tool as well as the general ability of testers in detecting duplicates. We further investigate the reasons for inconsistencies in duplicate detection ability between Bugle and the tester as well as the resulting limitations of automated solutions in the domain of duplicate bug report detection.

The expected scientific contribution of this study is to expand the literature with a study involving the tacit knowledge and intuition of testers in automated duplicate bug report detection—utilizing state-of-the-art techniques for duplicate detection with ramifications for future tool design considerations in the domain. Our technical contribution is Bugle, a software tool with the purpose of providing testers with the most probable duplicate bug reports based on the semantic similarity of their own textual formulation of an issue. Rather than focusing on improving the state-of-the-art of automated duplicate detection techniques, the objective of this study is to better understand the limitations of automated techniques in the domain of duplicate bug reports and the reasons for why automated techniques and testers' opinions on duplicates might differ.

II. Background and Related Work

A. Bug Reports

In the context of software, the term bug refers to an unexpected defect, fault, flaw, or imperfection [5]. 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. A bug report typically consists of various items, the most widely used ones being *steps to reproduce*, *observed* and *expected behavior*, *stack traces*, and *test cases*. Other common items include *screenshots*, *code examples* and *summary* [16]. Once a bug report has been filed, it is typically stored in a repository system, such as Bugzilla or Jira [8].

Since a bug report can consist of various items and there currently is no widely accepted industry standard for creating them, bug reports describing the same bug often end up rather different from each other. This also stems from the fact that many of the various items of a bug report are represented by natural language, with its innate contextual nuances and room for ambiguity. This leads to *duplicates*—bug reports that describe problems that have already been filed. For large repositories, duplicates often make up a large percentage of the bug reports filed; e.g., for the Mozilla and Eclipse projects the number has historically been as high as 30% and 20% respectively [2].

B. Duplicate Bug Report Detection

The problem of triaging large numbers of duplicate bug reports has been an oft-treated subject in the software engineering literature during the past decade. The problem space has been investigated since the early 2000s [8]. Hiew [8] provides an early account of an automated approach which uses a recommendation system based on techniques from the field of information retrieval for suggesting potential duplicates to bug triagers processing bug reports. The solution converts the textual description of bug reports into word vectors using the tf-idf measurement (term frequency-inverse document frequency), and uses the cosine similarity to determine how similar two bug reports are. The approach improved the accuracy of duplicate detection with a precision and recall of 29% and 50% respectively—however, in about 40% of the cases the approach resulted in false recommendations as bug reports were often related in their textual description but not considered actual duplicates after manual analysis by the author. In one of the samples of 100 duplicates, none of the top seven recommended duplicates were correct.

Runeson et al. [11] describe methods that focus on the attributes of a bug report described in natural language and found that a combination of textual summary and description resulted in being able to find 40% of marked duplicates. The study also compared different similarity metrics such as cosine, Dice and Jaccard and evaluated the approach in a case study at Sony Ericsson, where they found the cosine similarity resulting in the highest accuracy. Wang et al., [14] report similar findings regarding the usefulness of a combination of textual summary and description, but also that the addition of execution information further improves the duplicate detection result—leading to being able to find between 67% and 93% of duplicates. However, the study discusses the limited general feasibility of the approach, since special tools are often needed to collect the required execution information and in many instances execution traces are missing from bug reports. Sun [13] also echoes Wang's findings regarding the difficulty in considering execution information, since the information is often hard to collect and is often not available.

Kang [9] extends the literature with a systematic mapping study in order to categorize and structure all research between 2007 and 2017 on automated duplicate bug report detection. The study categorizes the approaches for automated duplicate detection into three distinct categories based on different problem formulations: top-N/ranking-based problems, which return ranked lists of recommended duplicates; classification problems, which test if a bug report is classified as duplicate or not; or decision-making problems, where prediction decisions are made for any pair of bug reports. The study finds that among the three categories, top-N/ranking-based problems achieve the best performance. Kang evaluated the effectiveness of an approach based on tf-idf and the deep learning algorithm Doc2Vec, resulting in a recall rate improving the state of

the art by 9%. The Doc2Vec algorithm uses document-level embedding, which makes up for the inability of tf-idf in dealing with synonyms. Doc2Vec generates representation vectors out of larger pieces of text rather than single words, and can therefore be used to represent semantics. In contrast to tf-idf, which deals with individual terms, Doc2Vec is able to deal with context.

Xie et al. [15] present a generalized approach combining convolutional neural networks (CNN), with domain-specific features extracted from bug report repositories. The approach improves the accuracy remarkably in comparison to both vector space and clustering-based solutions, with an average accuracy of 91.9%. In turn, Chaparro et al. [4] make the case that testers should be involved in more than one step of the process of duplicate bug report detection, one of submitting an initial report, and another of reformulating the report description to allow for re-querying the top-N retrieved duplicate suggestions. The approach allows for the retrieval of more duplicates and increases the testers' confidence in there not being any duplicates if none are found. In an empirical evaluation of the approach they report an average improvement of 56%-78% in retrieving duplicates by allowing the testers to reformulate the description compared to only using the initial description.

Haering et al., [7] aim to map informal, colloquially-written app reviews with matching technically-written bug reports, integrating user feedback into the bug fixing process. The researchers removed all user feedback of newly submitted bug reports containing less than ten words as previous research has shown that such feedback is most likely praise or spam. They use text embedding deep learning techniques to match the app reviews to bug reports by transforming the texts into the same vector space and identify matches by using the cosine similarity distance metric. The approach managed to find 167 relevant matches with bug reports of 200 total app reviews. However, upon manual inspection, in 44 instances the approach failed to identify a potential match.

Finally, Reimers and Gurevych [10] introduce SBERT, a modification of the language representation model BERT. BERT is one of the earliest and most influential large language models and was released by Google in 2018, and SBERT is a modified version of the pre-trained BERT network optimized for performance on a range of natural language processing tasks, including sentence-pair regression tasks such as semantic textual similarity, that can be used to produce semantically meaningful text embeddings for measuring and comparing semantic similarity between sentences.

In this study we build on Hiew's discussion on cases where automated methods result in false recommendations [8] since, in past work, the causes were not examined further. Our approach relies on the findings of Runeson [11], Wang [14] and Sun [13], who found that the parts of a bug report expressed in natural language yield the best accuracy for duplicate detection. Our tool utilizes a top-N/ranking-based approach based on the findings presented by Kang [9]. Due to the success of more recent applications of deep learning-based approaches [7], [9], [15] in the problem domain, we use deep learning techniques for detecting potential bug report duplicates, in particular the state-of-the-art Python framework SentenceTransformers[1], which is based on the work of Reimers and Gurevych [10]. Finally, our approach is inspired by the work of Chaparro et al. [4] in extending the involvement of the testers, taking advantage of their human faculties in detecting duplicates. We will in a similar manner take advantage of testers' tacit knowledge and intuition in evaluating the recommendations made by Bugle, our tool—emphasizing the limitations of automated duplicate detection and the potential causes for false duplicate recommendations.

III. Research Methodology

The objective of this study is to better understand the limitations of state-of-the-art automated duplicate bug report techniques as well as the reasons for why automated techniques and testers' opinions on duplicates might differ. To reach the objective, we analyze the relevant literature as well as the typical workflow of testers in practice of evaluating duplicate bug reports in order to derive requirements for a state-of-the-art tool for automated duplicate detection. We present Bugle, a tool for duplicate bug report detection based on state-of-the-art natural language processing techniques, and analyze the ability of a group of testers in evaluating the relevance of its automatically identified duplicates, using their tacit knowledge and intuition in judging the semantic similarity of issue descriptions. Furthermore, we define the typical workflow of evaluating duplicate bug reports as well as analyze the inconsistencies between automated duplicate recommendations and the judgement of a group of testers. We state our research questions as:

- RQ1: What is the typical workflow of testers when evaluating duplicate bug reports?
- RQ2: How often do testers disagree with the recommendations made based on semantic similarity of textual issue descriptions?
- RQ3: What are the reasons for why testers disagree with duplicate recommendations?

This study is conducted as an exploratory case study with a unit of analysis of how TestScouts evaluate duplicate bug reports. We aim to understand the typical workflow of evaluating duplicate bug reports in order to better be able to develop a state-of-the-art software tool for automated duplicate detection. We evaluate the rate at which testers disagree with the duplicate recommendations made by Bugle as well as analyze the reasons for those disagreements. An exploratory case study is the most suitable methodology to use for this study since our objective is to gain new insights, ideas and hypotheses from a realistic situation [12]. Since we develop a software tool, design science research would be an alternative

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

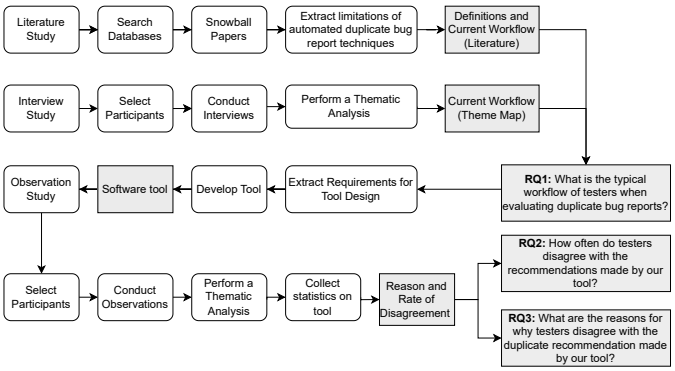

Fig. 1. Overview of the Case Study, including the Activities (Rounded Rectangles) and Corresponding Deliverable (Rectangles).

approach. However, rather than focusing on tool development, our research is concerned with gaining new perspectives on the limitations of automated duplicate bug report detection in practice.

A. Context

TestScouts is a consultant company with approximately 20 employees that has provided software testing services since 2015. TestScouts works with many aspects of testing such as agile test management, test automation and development of test environments. The company deals with bug reports on a daily basis and has expressed interest in developing a software tool for automated duplicate bug report detection. Next, we explain the data collection and methods of analysis for our research questions.

B. RQ1: Eliciting the Triaging Workflow

*1) **Data Collection**:* To answer RQ1, what the typical workflow of testers when evaluating duplicate bug reports is, we used interviews with seven testers at TestScouts together with the findings from related work described in Section II-B. The participants in the interviews are employees at TestScouts who possess knowledge of their current duplicate bug report workflow and have between 9–24 years of experience within a wide range of different software testing roles, as shown in Table I. The interviews were semi-structured and, with a written consent from the participants, recorded for later access and analysis. The interviews were in the first degree level, meaning we were in direct contact with the participants and collected data in real time [12]. We structured the interviews to be made up of the same 10 questions, as listed in Table II—some more overarching in scope to provide context for the study and to get the participants comfortable talking, and others more directly related to the participants' insights into the typical workflow surrounding duplicate bug reports based on their experience. We also left room to ask potential follow-up questions as they arose naturally as part of the conversation.

*2) **Data analysis**:* We conducted a thematic analysis of the qualitative data gathered from the interviews to answer RQ1, following the six-phase approach outlined by 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. After having conducted and transcribed the interviews, we each studied the interview transcripts individually in order to extract parts of the transcripts relevant to the stated research question. During this process, we individually identified the interview extracts that we found relevant in isolation to each other in order to ensure coherent interpretation of the data. We marked the interview extracts that we both found relevant, as well as the ones identified as relevant by only one or the other of us.

After analysing our consistency of individual codes across all seven interviews, we gained a 71% agreement, meaning that 71% of the identified interview extracts were identical between us and 29% were only identified by one or the other of us. Next, we generated initial codes relevant to answer our research question based on the interview extracts together. We iterated over all interview extracts, discussing the codes and interpretations of each extract together. During this process some overarching themes started to appear. For example, one participant noted the effects their prior experience and intuition had on their ability to identify a bug report duplicate:

> "you know, I've leaned on having that sort of real deep knowledge and that can be as much about the product but also the people and that's the thing" - Participant 6

TABLE I
SUMMARY OF THE INTERVIEW PARTICIPANTS. THE ROLES COLUMN SHOWS PAST AND CURRENT ROLES. THE EXPERIENCE COLUMN IS SHOWN IN YEARS.

No.	Roles	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

TABLE II
INTERVIEW QUESTIONS.

No.	Question
1	How many years of experience do you have in the IT industry?
2	How many years of experience do you have as a software tester?
3	What is your role as a tester?
4	In your experience, what is the typical workflow of working with bug reports?
5	In your experience, what is the typical workflow of working with duplicate bug reports?
6	What tools and techniques do you use for detecting duplicate bug reports?
7	What does the workflow of that tool and/or techniques look like?
8	How are the results given by your current tool or technique for detecting duplicate bug reports evaluated?
9	What role does your intuition play in the process of evaluating duplicates?
10	Do you question the results given by the duplicate bug report tool/technique?

> "nine times out of ten I wouldn't even need to get that validated, I would just say 'this is a duplicate, close the newer one, keep the older one open'." - Participant 6

We derived codes out of all identified interview extracts by grouping similar interview extracts together. Once all interview extracts had been assigned to a specific code, we grouped similar codes together such as *System knowledge* and *Intuition-based knowledge* for the example quotes above, and formulated an overarching theme of *Knowledge*. However, we arrived at the final thematic structure after several iterations of rearranging and renaming the codes and themes. The final themes and their corresponding description are outlined in detail in Section V-A.

C. RQ2 & RQ3: Rate and Reasons for Disagreement

1) ***Data collection****:* To answer RQ2 and RQ3, how often and why testers disagree with duplicate recommendations, we developed Bugle based on the interview findings and conducted an observational study at TestScouts to be able to analyze the frequency of disagreement between testers and Bugle on whether a given bug report description is a duplicate, as well as the reasons for why they disagree. For the sake of the observations, Bugle had been pre-loaded with industry data representing 1548 bug reports from an issue tracking repository for one of TestScouts' clients. Three of the participants in the observations had been acquainted with the dataset in their recent working experience, as testing consultants working on some of that client's projects. The other six participants had no prior acquaintance with the bug report dataset, but four of them did have previous experience working with evaluating duplicate bug reports, and two of them had no prior experience working with evaluating duplicate bug reports, as shown in Table III.

In order to simulate a scenario in which the tester was given information about an issue and could be tasked with searching for potential duplicates using our 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 to not give away the true duplicate too easily, but still containing enough information in order to be able to deduce them as being 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, in order to create the circumstance in which the tool would give an incorrect recommendation. These issue descriptions were printed on paper and given to the testers during the observations in order to avoid for them to simply copy and paste the text into the tool, but rather as visual input for them to construct their own semantically similar input description based on their given textual issue description.

TABLE III
SUMMARY OF THE OBSERVATION PARTICIPANTS. THE ROLES COLUMN SHOWS PAST AND CURRENT ROLES. THE EXPERIENCE COLUMN IS SHOWN IN YEARS.

No.	Roles	Experience
	Group 1	
P1	Test Engineer, Test Framework, Production Test	5 years
P2	Test Lead, Embedded Test, Automated Test	12 years
P3	Test Engineer, Performance tester	3 years
	Group 2	
P4	Test Lead, Manual test, Automated Test	9 years
P5	System tester	4 years
P6	User Acceptance Testing, Test Management	16 years
P7	Test Lead	7 years
	Group 3	
P8	Test Automation	<1 year
P9	Test Automation	<1 year

The observations were of first degree, meaning observations with "think aloud protocols". During the observations we were in direct contact with the participants and collected qualitative and quantitative data in real time while letting the participants use the tool in order to identify duplicate bug reports. The participants were able to judge the correctness of the recommendations made by the tool as well as to provide an explanation for the reasoning behind their decision. With consent from the participants, we recorded the audio, video and screen of the participants while they solved the four tasks of the observation. 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 [12].

We asked the participants to do the following—(i) to read and familiarize themselves with the given issue descriptions one by one, (ii) formulate their own input to the tool based on the given issue description, and (iii) judge the relevance of the recommendations given by the tool based on the input and label the recommendations as either duplicate or not duplicate of the issue description provided by us. Participants were regarded as labeling an issue description as duplicate if they reported wanting to take any of the following actions—(i) linking the given issue description to any of the recommended bug reports, (ii) adding complementary information to any of the recommended bug reports, and (iii) expressing the need to investigate any of the recommended bug reports further, e.g. by communicating with the creator of a recommended duplicate. Conversely, participants were regarded as labeling an issue description as not being a duplicate if they reported wanting to create a new bug report. During the last step where participants judged the relevance of the recommendations, we collected

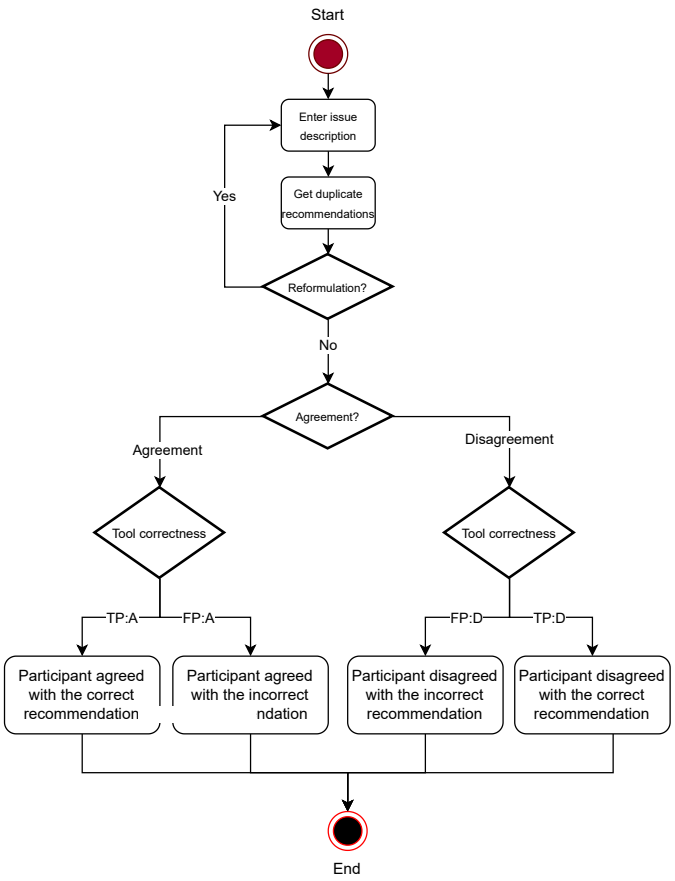

Fig. 2. Quantitative Data Collection for RQ2.

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) Agreement (*TPA*): participant agreed with the recommendation
   b) Disagreement (*TPD*): participant disagreed with the recommendation
2) False Positive: tool gave an incorrect recommendation.
   a) Agreement (*FPA*): participant agreed with the recommendation.
   b) Disagreement (*FPD*): participant disagreed with the recommendation.

*2) **Data analysis**:* During the observations, we collected the quantitative data of all nine participants based on their answers on all four issue descriptions—following the activities described in Figure 2. 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 in those cases successfully identified the issue as a new bug report. We calculated the average accuracy for the participants in detecting duplicate bug reports:

$$accuracy_{participant} = \frac{TPA + FPD}{TPA + TPD + FPA + FPD}$$

We also calculated the corresponding average accuracy of Bugle in retrieving the correct duplicates base on any of the input description:

$$accuracy_{tool} = \frac{TPA + TPD}{TPA + TPD + FPA + FPD}$$

Furthermore, we measured the mean rate of disagreement with the participants and the tool in detecting duplicate bug reports:

$$disagreement = \frac{FPD + TPD}{TPA + TPD + FPA + FPD}$$

We conducted a thematic analysis of the qualitative data gathered from the observations to answer RQ3—following the six-phase approach outlined by Braun & Clarke [3] of familiarising ourselves with the data, generating initial codes, searching for themes, reviewing themes, defining and naming themes, and producing the report. We each studied the recordings of the observations as well as the notes taken during the observations in order to extract quotes and behavior relevant to the research question. During this process we particularly paid attention to cases during the observations in which the participants disagreed about the recommendations made by the tool as being actual duplicates. We marked the quotes and notes from the observations that each of us regarded as relevant to the research question individually, and compared our marked extracts for consistency after which we reached an 81% agreement between the two of us. Based on the quotes and notes from the observations we then started to generate initial codes from our marked extracts. We reviewed and grouped similar codes together in iterations as overarching themes started to emerge, after which three main themes eventually stood out.

Those main themes and corresponding sub-themes are discussed at length in Section V-B. Along with the themes for answering RQ3 we identified additional themes that were of interest for sake of discussion but not immediately connected to RQ3. These qualitative findings are discussed in detail in Section VI-B.

Fig. 3. The user interface of Bugle - a centered search bar where testers input an issue description in order to get Bugle's recommendations of potential duplicate bug reports.

IV. Tool Design

Based on the results from the interviews, see Section V-A, together with the key findings from the related work in Section II-B, we derived requirements for a stand-alone web application tool for automated duplicate bug report detection. The tool, Bugle, lets testers provide an issue description as input and will output the top six similar issue descriptions from a given repository of bug reports, a design decision made based on the findings of Kang [9] regarding the superior performance of top-N/ranking-based approaches. Based on the findings of Runeson [11], Wang [14] and Sun [13], the input is expected to conform to the textual *description* field of a bug report, and for every input query, Bugle outputs the bug reports from the dataset with the description containing the highest contextual similarity in real time. Bugle lets the testers evaluate the relevance of the output with regards to the input as well as allow the tester to reformulate the input indefinitely, based on the successful domain-specific application of such an approach presented by Chaparro et al. [4].

A. User Interface

We designed Bugle to be a stand-alone web application written in Python and based on the Django web framework. We chose Python because of its popularity in deep learning applications and the frameworks, e.g. SentenceTransformer, available for that language. We chose the Django web framework because of its built-in MVC architecture and simplicity that aids in rapid development. Due to the search-based nature of Bugle, inspiration for the user interface was taken from the Google Search homepage, with its minimal look and simple input form centered on the page being easily recognized patterns for the user interface design. Figure 4 displays a screenshot of Bugle's user interface when recommending six suggestions based on a input description.

B. Bug Report Data and Semantic Textual Similarity

Bugle is data-agnostic, i.e. it functions independently of the bug report dataset used. Most organizations use issue tracking systems such as Bugzilla or Jira [8], that allows for exporting bug report data as csv files which allows for easy integration of new datasets into Bugle.

The input bug descriptions get handled asynchronously, and are encoded into sentence embeddings using Sentence-Transformers, a Python framework for state-of-the-art sentence embeddings based on Sentence-BERT [10], a pre-trained transformer network fine-tuned for common natural language processing tasks such as semantic textual similarity. The input

Fig. 4. Recommendations are displayed and ranked in a table as the user enters an issue description. Semantic similarity is normalized and color-coded in the first column to showcase the likelihood of a duplicate in the repository.

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 top-N bug reports to the user.

We based the choice of deep learning models to transform the textual data of bug report descriptions into vectorized text embeddings on the prior reports of Kang [9], Xie [15] and Haering [7] in the same domain. Furthermore, we derived the decision to use the cosine similarity metric on the prior findings by Hiew [8] and Runeson et al. [11].

C. Large Language Models

We evaluated the accuracy of three pre-trained sentence transformer models in their ability of correctly detecting duplicate bug reports across two large open source bug report datasets, both containing a ground truth with regards to duplicates. The three models were selected due to their differing qualities. The first model, *paraphrase-MiniLM-L3-v2*, produces the embeddings in the shortest time, is smallest in size and has the lowest average accuracy. The second model, *multi-qa-MiniLM-L6-cos-v1*, is only 25% slower than *paraphrase-Mini LM-L3-v2*[2], but the accuracy in detecting semantic similarity in natural language is improved by 25%. The last model, *all-mpnet-base-v2*, was largest in size and, compared to *multi-qa-MiniLM-L6-cos-v1*, 5 times slower. However, *all-mpnet-base-v2* performs best in detecting semantic similarity in natural language.

The datasets used to evaluate the models 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. We produced the sentence embeddings for the bug descriptions for both datasets and evaluated the accuracy of the three pre-trained sentence transformer models. Since Bugle recommends the top 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

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

recommendations was in fact 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 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 had actually found one. This means that the resulting accuracy for the two open source bug report datasets might be higher than listed below in Table IV, since the open source datasets don't link duplicates bidirectionally.

Before being able to measure the accuracy of Bugle on the two open source datasets, they needed to be pre-processed. The bug reports for the Eclipse project followed a certain format for their description fields, where the textual description was concatenated with additional pieces information such as comments, URLs etc. These pieces of information were all concatenated in a string separated by semicolons with the textual description before the first semicolon, which made it possible for us to extract only the textual description of a given bug report from the data field. However, not all bug reports in the dataset followed the format of having the textual issue description before the first semicolon—which led to noisy data being encoded into sentence embeddings thus misrepresenting the semantic content of the actual description leading to inaccurate similarity scores.

The bug reports in the Mozilla Firefox project also needed to be pre-processed, but the textual description field of bug reports within the dataset did not follow any certain format—making the extraction of the purely textual description difficult. Instead of extracting the textual description of all bug reports, we filtered out duplicate bug reports that where particularly noisy, e.g. bug reports where the description field was empty or only containing stack trace information or URLs.

The client dataset pre-loaded into the tool for the observational part of the study was pre-processed in a way similar to the Eclipse dataset, extracting the purely textual descriptional information out of concatenated pieces of information in the description field. This pre-processing made sure that not only were bug descriptions displayed on the user interface in a semantically coherent format, they also produced less noisy embeddings—improving the model's accuracy in detectiing semantic textual similarity.

After the datasets had undergone pre-processing to remove undesired bug reports and description fields, we could conclude that the model *all-mpnet-base-v2* managed to identify the highest number of duplicates out of the three models with an accuracy of 13.69% and 35.26% respectively for the two datasets. This meant that for the Eclipse dataset the model correctly identified one out of every three marked duplicates based on one textual issue description. Bugle is designed not only to recommend actual duplicates, but also to provide a hint for the testers on which bug reports in the dataset are most relevant based on their current input description—aiding them in being able to reformulate it. Since Bugle always returns a ranked list of the top six bug reports with the highest similarity score, we did not measure any true or false negatives. Therefore the model evaluation had a recall of 100%, making accuracy the relevant metric for the evaluating the model.

The results of duplicate detection for the three large language models on these open source datasets are quite low. This is because the large language models are only used for measuring textual similarity, but a large amount of the bug reports marked as duplicates in these open source datasets do not include semantically coherent textual descriptions at all. Sometimes they do, but the problems mentioned above related to pre-processing make the extraction of the purely textual information difficult and introduce noise into the embeddings. In order to improve the accuracy of the large language models further on these particular datasets, further effort needs to be expended on pre-processing to remove noise in the underlying data. This has been done previously and discussed extensively by Xie et al. [15] and Haering et al. [7] but was not explored further in this study due to time constraints. The dependency on this aspect of bug report data quality becomes clear as the models detect up to 35.26% of duplicates correctly on the Eclipse dataset, but only 13.69% on the Mozilla Firefox dataset. The gap in accuracy between the two datasets is accounted for by the difference in preconditions for extracting the textual descriptions between the two datasets, which is stated above. Eclipse was pre-processed so that only the textual description field was taken into account in the model evaluation whereas for the Mozilla Firefox dataset additional information than the purely textual issue description is still embedded after the pre-processing of the dataset—thus reducing the accuracy for the entire dataset. See Table IV for the complete result of the evaluation of all models and datasets.

V. Results

A. RQ1—Tester Workflow

What is the typical workflow of testers when evaluating duplicate bug reports?

The analysis resulted in a thematic map consisting of three main themes, divided into nine high-level sub-themes and four low-level sub-themes. The main themes for the typical workflow of evaluating bug report duplicates are *knowledge*, *duplicate identification* and *duplicate management*.

1) ***Knowledge****:* An overarching theme that became apparent across all interviews was the role played by different facets of the knowledge of the testers in identifying bug report duplicates—a frequent remark made by all of the participants. As shown in Fig. 5, this knowledge manifested itself mainly as *system knowledge*—specific knowledge about the system and its bugs derived from experience working with the system and as *intuition-based knowledge*—gut feeling or intuition guiding the decision-making. Participants also made reference to knowledge based on *memory*—their capacity to remember previous bugs and recognize specific behavior.

TABLE IV
MODEL EVALUATION OF TWO OPEN SOURCE DATASETS

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%

TABLE V
RQ1—DESCRIPTION OF MAIN THEMES.

ID.	Theme	Description
MT1	Knowledge	The different types of knowledge of the testers utilized in order to identify duplicates.
MT2	Duplicate Identification	The practices and techniques used to identify bug report duplicates
MT3	Duplicate Management	The main steps of the workflow when evaluating bug report duplicates, from the creation of a bug report, through linking and closing duplicates after identification.

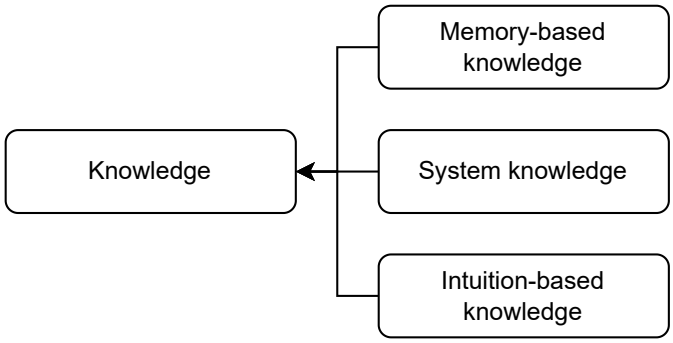

Fig. 5. Sub-Themes Related to Knowledge.

The participants made frequent remarks about the importance of *system knowledge* in their workflow of identifying duplicates. 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 prerequisite context and language necessary to evaluate duplicates in an efficient workflow.

> "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

Several remarks were also made about the knowledge used as part of the workflow as being *intuition-based*. Participants often reported their basis for decision-making regarding duplicates as a gut feeling or intuition—knowledge stemming from prior experience, but not necessarily specific to knowledge about the system-under-test.

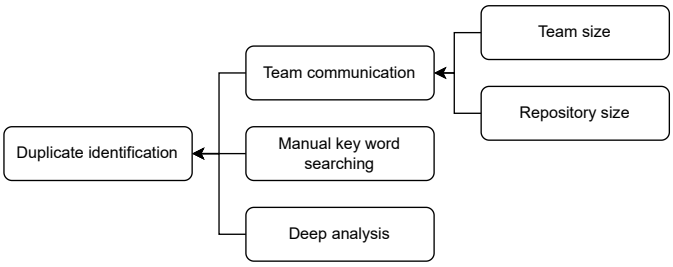

Fig. 6. Sub-Themes Related to Duplicate Identification.

> "My backbone tells me that if I think it's a duplicate, then I close it. I can refer to the other one and say that there is already one." - Participant 1

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

The final type of reference made to knowledge in discussing the typical duplicate workflow is one regarding *memory-based knowledge*. Participants report simply remembering instances of having seen similar bug reports in the issue tracking system previously, and let their memory of current open bug reports guide them in their decision-making when identifying duplicates.

> "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

> "You start writing a bug, and then it is entirely dependent on someone remembering that it exists, I would say." - Participant 2

*2) **Duplicate Identification**:* By *duplicate identification*, we refer to the practices and techniques used to identify bug report duplicates in practice. We identified three high-level themes for duplicate identification—*manual keyword searching*, *team communication* and *deep analysis*, shown in Fig. 6, as well as *team size* and *repository size* being lower level themes affecting the team communication.

One salient insight gained from all interviews was the absence of the use of any specific tool in the company for the purpose of identifying duplicates.

> "There is no tool that I have come across in my career that supports this." - Participant 1

Instead, all participants reported that the most common technique for finding duplicates is performing *manual searches*

for individual keywords and phrases in the issue tracking system used by their organization. This approach potentially saves time for the tester since a quick keyword search may be all that is needed to identify a duplicate, and then the tester doesn't need to write 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 carried out in order to be able to conclude if the bug report is a duplicate or not, such as investigating the source code for a root cause that is manifesting itself throughout the system in different ways. Especially if the manual keyword search results in ambiguity whether the bug report is a duplicate or not, there is a need for further analysis to establish the truth with certainty. Furthermore, 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 analysis before they can be confirmed or rejected as duplicates.

> "It might be the case that, regardless of the situation, one must conduct an analysis to clearly demonstrate that this is a duplicate." - Participant 4

Emphasis was also put on the importance of *team communication* as team members develop insights regarding different parts of the system. Especially as the systems get larger and development requires larger teams, the reliability of the testers' individual knowledge of the system and ability to identify duplicates by themselves is impaired, and emphasis is put on the importance of team communication. However, even in systems involving smaller teams, team communication is a common strategy for duplicate identification.

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

The aspect of *team size* as well as *repository size* were recurring themes—indicating that the process of identifying duplicates via team communication gets more challenging as the sizes of the team and the software repository increase, sometimes necessitating standards to be put in place for how team communication should be done, and sometimes even necessitating standards for how bug reports should be written within the organization to aid in duplicate identification.

> "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

3) ***Duplicate Management****:* By *duplicate management*, we refer to the steps taken both before and after duplicate identification. All participants reported the typical workflow to contain certain measures for management of the identified duplicate bug report. We identified the following three high-level sub-themes for *duplicate management—creating bug reports*, *linking bug reports* and *closing bug reports*, as well as two lower-level sub-themes of *organizational safety criteria* and *evaluation* upon closing, shown in Fig. 7.

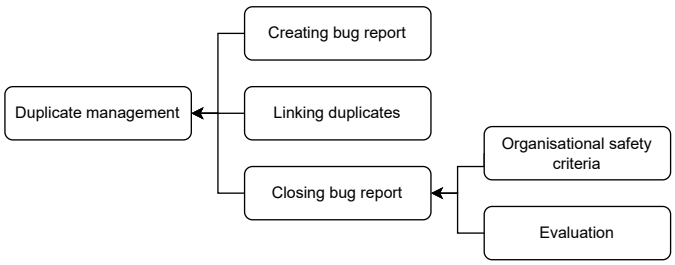

Fig. 7. Sub-Themes Related to Duplicate Management.

Participants reported that how bug reports are created matters for successful duplicate identification. More specifically, how bug reports are formulated when *creating bug reports* should be standardized or, at least, follow a pre-defined structure in order to proactively aid in duplicate identification. Textual data is often ambiguous in nature and an error is likely to be described slightly differently by different people. Since the most useful aspects of a bug report for identifying duplicates are the textual fields [11], [13], [14], the need arises to standardize the way in which these textual inputs are formulated. This need reportedly increases with team size.

> "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 is that confirmed duplicates should always be linked together, and the bug report containing the least amount of information should be closed. The issue tracking systems used in the organizations allow different ways to do this—either by creating an actual link between two reports, or by including the ID of the closed identified duplicate. The *linking of duplicates* is important since a duplicate bug report might contain additional valuable information for solving the bug that would be lost without the linking.

> "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

Closing bug reports is to be encouraged as a way of maintaining the bug report repository and participants reported that it is easier to keep track of open issues when they are few in number. The reason for closing duplicates is one of maintenance. As long as the complementary information contained in the duplicate is linked to the open bug report, closing is always to be preferred since that information is then accessible should the need arise.

> "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 bug report mistakenly closed as a duplicate could potentially result in fatal consequences. In systems where safety is less of a concern, closing bug reports is always to be preferred since reducing the number of open issues reduces the cost and complexity of maintaining the bug report repository.

> "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

> "It depends on how hard one can clean up, I'm used to being able to clean up pretty thoroughly because it doesn't matter that much, people still survive." - Participant 3

There were some differing views among the participants on who should be able to *evaluate and close a duplicate*. Some made the argument that only the initial creator of a bug report should be able to close it. Others argued that it is preferable to only have few open issues, and therefore one should be able to close issues created by others given that one links them together.

> "I generally have the rule that the only one allowed to close such a process is the one who has written it." - Participant 1

> "And sometimes you have to accept that your baby is closed like that." - Participant 3

The typical workflow, as shown in Fig. 8, is derived from the three main themes—*knowledge*, *duplicate identification* and *duplicate management*.

RQ1: What is the typical workflow of testers when evaluating duplicate bug reports?
The typical workflow consists of (i) creating a bug report, (ii) using either manual keyword searches, team communication or deeper analysis to identify if there are any duplicates, (iii) linking duplicate bug reports together, and (iv) closing the bug report. Factors affecting the different parts of the workflow are team and repository size, different forms of knowledge possessed by the testers, and the safety criteria specific to the organization.

B. RQ2—Rate of Disagreement with Tool

How often do testers disagree with the recommendations made by our tool?

In Table VII, we list the results for each of the issue descriptions that the participants evaluated with the help of Bugle as either a duplicate or not. As explained in further detail in Figure 2, we measured the total amount of TPD, FPD, TPA and FPA for each participant. In addition, Table VIII shows the average measures for each of the four issue descriptions that each participant evaluated during the observations. Table VI lists the rate of disagreement between the three different groups who took part in the observations and shows the average accuracy—both for the tool itself based on the input descriptions as well as for the participants in detecting duplicate bug reports using Bugle.

TABLE VI
RATE OF DISAGREEMENT, TOOL ACCURACY AND TESTER ACCURACY OF THE THREE GROUPS OF PARTICIPANTS.

	Group 1	Group 2	Group 3
Rate of disagreement	41.70%	91.67%	58.33%
Tool accuracy	43.75%	93.75% 8	75.00%
Tester accuracy	0.00%	100.00 %	100.00%

RQ2: How often do testers disagree with the recommendations made based on semantic similarity of textual issue descriptions?
The tool correctly identifies the duplicate bug report based on the participants' input descriptions with an average accuracy of 94.44%. However, the participants do not agree with the tool's recommendations in 38.90% of cases. The participants' average accuracy was 75.00%.

C. RQ3—Reasons for Disagreement

What are the reasons for why testers disagree with the duplicate recommendation made by our tool?

The thematic analysis of the data gathered during 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. These themes were *semantic differences in descriptions*, *insufficient information* and *overlooking recommendations*, as described in Table IX.

*1) **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 9, we list the derived themes and corresponding high-level sub-themes.

In certain cases the differences were subtle—e.g. for the first issue description, one participant remarked it mentioning the "flipping of a switch", and not "holding down a switch", which was how the description of the highest ranked recommendation based on that input was phrased. The participant argued that there could be two slightly different behaviors and would have raised a new bug report rather than trusting the highest ranked recommendation to be a duplicate in this case.

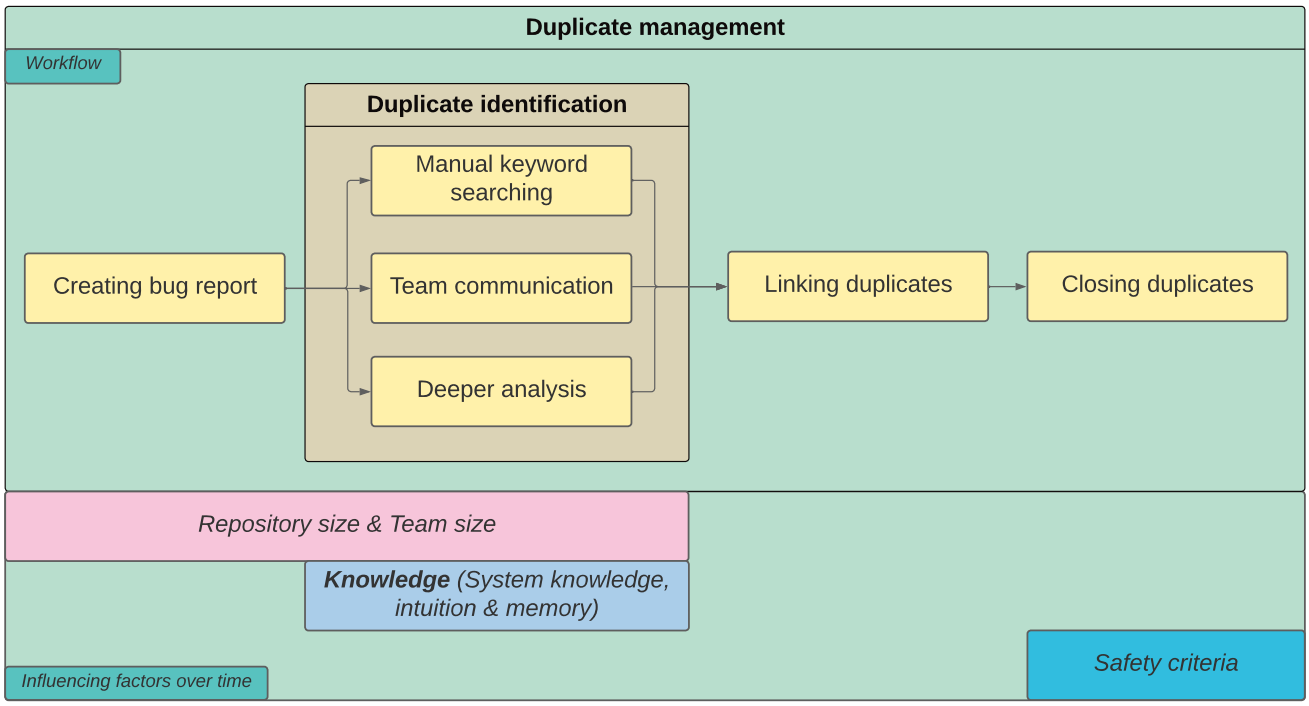

Fig. 8. Typical Workflow when Evaluating Duplicate Bug Reports.

TABLE VII
RESULTS OF HOW THE PARTICIPANTS LABELED ALL ISSUES DURING OBSERVATIONS.

	Disagreement		Agreement	
Issue No.	**True Positive (TPD)**	**False Positive (FPD)**	**True Positive (TPA)**	**False Positive (FPA)**
1	2	-	7	-
2	-	8	-	1
3	2	-	6	1
4	2	-	6	1
Total	**6**	**8**	**19**	**3**
Tool accuracy*	**92.60%**			
Participant accuracy**	**75.00%**			
Rate of disagreement***	**38.90%**			

* Average accuracy of tool recommendations, not including issue no. 2 since there is no duplicate bug for that issue in the bug report repository.

** Average accuracy of participant judgement, including all issues.

*** The rate of disagreement with the participant and the tool, including all issues.

Another issue description detailed a case where a motor ran at top speed, after which it slowed down 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 the motor coming to a complete halt—it came close to a full stop, but then the speed was slightly increased again, which led to a user interface shutting down.

> "Oh, this is very close... but I would say that this is a new issue also" - Participant 9

Most commonly, the disagreements based on semantic differences were concerned with one or several especially important keywords not being present in the description of the recommended bug reports.

> "Some slight similarities but no recommendation where all important keywords were present to be able to confirm duplicate" - Participant 7

> "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

> "The results are talking about the fan and not the motor, that tells me that there are similar issues in the repo, but not anyone referring to the motor." - Participant 9

TABLE VIII
RESULTS OF EACH PARTICIPANT DURING OBSERVATIONS, INCLUDING PARTICIPANT'S TIME TO DECISION, NO. OF REFORMULATIONS AND THE RATE OF CORRECTLY LABELED DUPLICATES FOR PARTICIPANT AND BUGLE

Participant no.	Participant				Bugle	
	Time to evaluate duplicate (min:s)*	No. of queries used**	Accuracy	Rate of disagreement***	Time to detect duplicate (min:s)	No. of queries used****
P1	5:22	3.00	1/4	1/4	4:14	2.70
P2	2:55	2.75	3/4	2/4	1:32	1.70
P3	6:28	7.25	3/4	2/4	0:34	1.00
P4	2:37	5.00	3/4	1/4	1:42	4.70
P5	2:01	4.75	3/4	2/4	0:33	2.30
P6	2:31	3.75	4/4	1/4	0:25	2.00
P7	3:58	4.50	2/4	3/4	1:05	2.00
P8	2:13	3.75	4/4	1/4	0:53	2.70
P9	2:17	3.00	4/4	1/4	0:38	2.01
Average	**3:22**	**4.19**	**75.00%**	**38.90%**	**1:17**	**2.35**

* Average time spent per participant until participant makes final decision

** Average no. of reformulation inputs per participant until final decision.

*** No. of Disagreement/No. of agreement per observation and participant.

**** Average no. of reformulation inputs per participant until tool suggest correct issue description.

TABLE IX
RQ3—DESCRIPTION OF MAIN THEMES FOR REASONS OF DISAGREEMENT.

ID.	Theme	Description
MT4	Semantic Differences	The provided issue description and recommended bug report description reported to have different meanings
MT5	Insufficient Information	The provided issue description or recommended bug report description reported as not containing enough information to be able to make the judgement
MT6	Overlooking Recommendations	Not reading the description of a recommendation at all or not reading it carefully enough

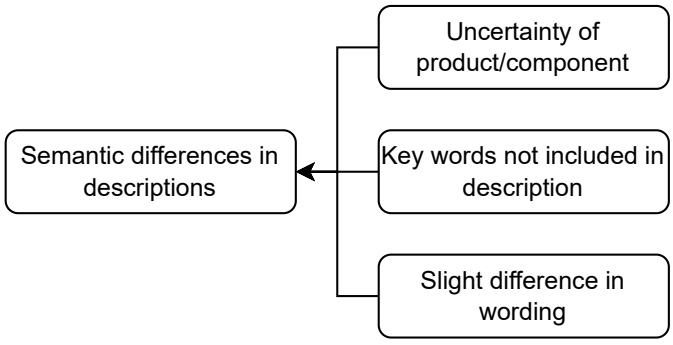

Fig. 9. Sub-Themes Related to Semantic Differences in Descriptions.

*2) **Insufficient Information**:* The second most common reason for disagreement was that the participants reported not having sufficient information at their disposal to be able to determine a recommendation made by the tool to be a duplicate or not, and therefore opting to disagree with the recommendation and to create a new bug report. In Figure 10 we list the derived theme and its corresponding high-level and low-level sub-themes.

This remark was more frequently made by the more experienced participants and the participants who possessed more in-depth system knowledge, as compared to the participants with less experience.

> "It contains too little information to be able to confirm. If it's a duplicate, how would I know?" - Participant 6

Participants with previous experience working with evaluating duplicate bug reports also remarked that they would like more specific pieces of information in order to determine with certainty that a recommendation was in fact a duplicate. Such pieces of information included requirement ID, more detailed requirements, the specific project the bug report belonged to, as well as bug issue status.

> "Would need more information from requirement to determine." - Participant 3

> "Would have been good to see status." - Participant 2

> "It looks like the same one, but I would like to check which project it is to be sure." - Participant 2

*3) **Overlooking Recommendations**:* The third most common reason for disagreement was that participants overlooked a correctly made recommendation by the tool, either by reading through the recommended bug report hastily and missing important details mentioned in the description, or by not reading the recommended bug report at all. In Figure 11 we list the derived theme, with its corresponding high-level and low-level sub-themes.

Upon being asked for the reason for why they disagreed with a by the tool correctly made recommendation, one participant looked at the description again and acknowledged that

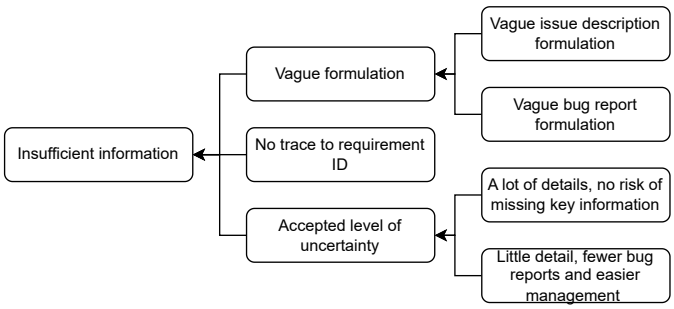

Fig. 10. Sub-Themes Related to Insufficient Information.

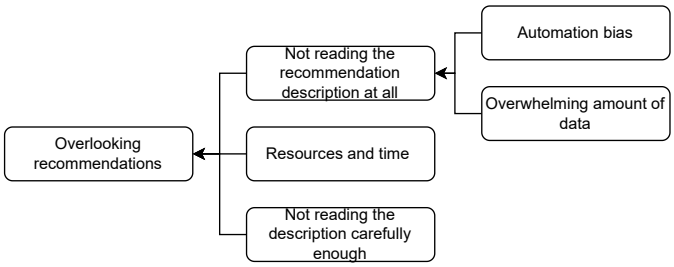

Fig. 11. Sub-Themes Related to Overlooking Recommendations.

it might indeed 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 and difficult to interpret, or because the recommendations were not ranked as high as others among the six most similar ones 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

RQ3: What are the reasons for why testers disagree with duplicate recommendations?

The reasons for disagreement are *semantic differences in descriptions*, *insufficient information* provided by the tool to make recommendations, and *overlooking recommendations* that appeared towards the bottom of the ranked list.

VI. Discussion

A. RQ1

One insight gained from this study is that, while several automated techniques for identifying duplicate bug reports have been described in the literature, none have made their way into any widespread commercial use as a tool to alleviate the workload of bug triaging for software testers. None of the interview participants, some of which had more than two decades of experience in software testing, reported to have heard of such a tool. Instead, the typical activities conducted for duplicate identification, such as keyword searches through datasets of bug reports and communication between key people involved in a project, are performed manually and do not involve any intelligent software tool. Participants did, however, express that such a tool would be of significant benefit to their workflow.

The interview data revealed a trade-off between the choice of either creating a new bug report or spending additional time looking for already existing duplicates. The tolerance for technical debt plays an important part in that tradeoff, which has a strong correlation to the acceptance level of quality assurance regarding safety for the system-under-test. For example, 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 a reported issue prematurely. In contrast, for systems where safety is not of utmost importance, closing an issue prematurely does not have the same risk associated with it.

One interview participant discussed an example of the average cost per bug report in a particular organization amounting to ca. 10000 SEK. If the average time spent looking for potential duplicates can be reduced with the help of a tool, making it possible to perform a more extensive search in a shorter amount of time, not only would this enable testers to conclude that no duplicates exist with a higher level of certainty, but it would also save the organization money and reduce technical debt by making bug triaging more effective.

B. RQ2 & RQ3

*1) **Contradictory findings regarding knowledge**:* One interesting finding from the observational study concerns the relationship between a tester's knowledge and the general ability of that tester to successfully detect duplicates using Bugle. The knowledge of a software tester, expressed as system knowledge as well as intuition and memory, stood out as an overarching theme in evaluating duplicate bug reports. However, the observational data suggests an inverse relationship between tester knowledge and successful duplicate detection.

More experienced testers tended to claim to need more information to determine whether a bug report recommended by the tool 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 with.

More experienced participants also commented on not having access to additional contextual information that they are used to having, thus 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 rather create a new bug report for their provided issue description out of uncertainty based on insufficient information.

We make the conjecture that the reasons for the inverse relationship between tester knowledge and successful duplicate detection have to do with the more knowledgeable participants being able to take additional information from their experience

into account as a basis for their opinion, whereas the less knowledgeable participants can only base their opinion on the limited amount of information at their direct disposal.

The evidence for this conjecture is that the more knowledgeable participants several times used keywords in their input formulations that were not derived from their provided issue description, but rather their own domain knowledge about the dataset. What is interesting is that—although the more knowledgeable participants had more information at their disposal—they were actually less successful than their less experienced peers in the third group in detecting true duplicates with the help of the tool, and had a higher rate of disagreement on average.

Another possible reason could be that the participants with prior experience evaluating duplicate bug reports had an already ingrained method for searching through bug report datasets for duplicates using manual keyword searches, e.g., Ctrl+F in Windows. The choice of input queries used by participants during the observational part of the study suggests that the participants with prior experience evaluating duplicates relied more heavily on concatenating individual keywords for their input than describing the issue in more semantically coherent sentences.

The participants without prior experience of evaluating duplicates seemed to adapt their input formulations as free-flowing descriptions more naturally than their more experienced peers, which helps to explain their superior results with the tool, whose model has been fine-tuned specifically for semantic textual similarity, something we explained to all participants at the start of the observations.

However, these observations are based on a relatively small pool of participants and any hypotheses should be verified in further, larger observational studies.

*2) **Bias**:* Participants made reference to certain types of biases during the observational study as well. In several instances, the similarity score highlighted in either green, yellow or red (to indicate high or low similarity) was commented on as affecting the participants' opinion about a recommendation being a likely duplicate or not.

Sometimes the similarity score seemed to the participants to be too high or low relative to their expectations. In certain cases, this seemed to be a reason for not reading the lower-ranked recommendations' descriptions, leading to participants choosing to create a new bug report for issues whose duplicate the tool had successful retrieved based on their input. However, they had simply failed to notice the duplicate because it had been ranked lower than other recommendations.

Another form of bias commented on concerned the loss of faith in the tool's utility after several tries to find a duplicate had failed to retrieve anything of interest. One participant commented specifically on being subject to such bias after being 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, in fact, dismissed a very likely duplicate due to having underestimated the tool's utility after failed attempts.

*3) **Suitability of LLMs for duplicate detection**:* Another topic for discussion is the question of whether a powerful large language model such as *all-mpnet-base-v2* is actually necessary for the purpose of detecting bug report duplicates—or if it is analog to "killing a fly with a bazooka." Comparing the tool to the current approach used in industry today of manual keyword searches for duplicate detection, some participants with prior experience of evaluating duplicates commented that they did not think that the tool would help them be more effective than their current way of working, while others thought it would.

They also pointed out that the tool would likely be beneficial for less experienced testers as well as for cases in which they would start working on new unfamiliar projects, as the tool would accept free-flowing input descriptions, allowing them to describe issues without the need to be familiar with common keywords of the domain.

Compared with static keyword searches, the power of semantic search lies in the ability of the model to detect not only synonyms, but also context similarity across sentences. This was demonstrated repeatedly in the observations as each issue description based on an actual bug report had been extensively reformulated from the original using multiple synonyms while keeping the underlying semantic content intact, which would not have resulted in successful duplicate detection had the language model not been trained on semantically similar sentence pairs or fine-tuned specifically for the domain of sentence-pair regression tasks.

C. Limitations of automated duplicate bug report detection

The results of our study help to showcase the challenges to overcome in the design of future tools in the domain of duplicate bug detection and helps to highlight the limitations present in current approaches.

One of the biggest current limitations for automated duplicate detection is the dependency on high-quality bug report data. Since this study used general pre-trained language models fine-tuned for semantic textual similarity, possible improvements would include using sentence transformer models specifically trained on bug report data. A hurdle would be the amount and availability of data needed for training such a model, and the model's ability to generalise its performance across the domain from training on that data.

Industry actors are often reluctant of sharing bug report data for security reasons. Large open source bug report repositories such as those used as part of this study are available online[3], and have a history of usage in this domain. However, the quality of bug reports varies widely across projects and organizations—and even within them—and the lack of a standard input language for bug reporting results in a need for extensive data preprocessing to achieve high accuracy.

[3]BugRepo, LogPAI Team, 2018, https://github.com/logpai/bugrepo

Improving organizational standards regarding bug reporting would likely improve model accuracy regardless of whether large language models or models specifically trained on bug report data are used. If more bug reports describe bugs in natural language, a large language model would likely achieve higher accuracy for duplicate detection having been trained on similar natural language data, whether that data is general or domain-specific. This study also showcases how bug descriptions in natural language are one of the most useful items for detecting duplicate bug reports in current non-automated typical workflows evaluating bug reports in industry as well.

The difference in accuracy in this study between the model evaluation on open source datasets and the observation with industry data with a tester evaluating the tool's recommendations also highlights the power of the combination of a large language model and the opinion of testers in successfully detecting duplicate bug reports, in comparison to solely relying on automated duplicate detection. The combination eliminates false recommendations being automatically wrongly marked as duplicates, something observed in 40% of the cases in Hiew's [8] original solution, by having a tester evaluate the recommendations made by the tool.

There is an important design decision related to the similarity threshold at which a bug report is determined to be a duplicate. This decision places limitations on automated duplicate bug report detection. If the threshold is very high, true duplicates with lower similarity scores will not be detected, whereas a lower threshold will result in bug reports being falsely detected as duplicates. There is, thus, a trade-off between either missing true duplicates or falsely marking bug reports as duplicates inherent in the tool design.

We put that threshold at a low level, and Bugle is designed to always offer recommendations to testers, outsourcing the final decision to the testers. This design decision prevents a large number of false duplicates from being marked, even though the threshold is rather low, but automated solutions not involving human testers in the decision making would need additional complexity in duplicate detection to account for the lack of tester knowledge in making that assessment. This additional complexity would mainly relate to contextual system knowledge, but also for the safety criteria demanded by the specific organizations and systems.

VII. Threats to Validity

This section concerns threats to validity of the study, where validity threats from each of the four categories of threats to validity in a case study outlined by Runeson and Höst [12] will be discussed.

Construct Validity concerns the extent to which the findings accurately reflect the intentions of the study. One example of such in this case study is that testers could be too trusting of the recommendations by Bugle, which will lead to participants not utilizing their intuition as intended during the observations. We mitigated this threat by clearly stating to all participants that the tool is merely providing suggestions and that we are interested in the participants' ability to judge the relevance of the recommendations. We further explained to all participants during the observations that potential duplicates could be in any position in the ranked list of recommendations, even with a low similarity score, and that the recommendations are merely based on the input description.

Internal Validity also concerns the extent to which the findings accurately reflect the intentions of the study. In contrast to construct validity, the internal validity concerns the cause-and-effect relationship between an independent and dependent variable. When analyzing whether one factor affects an investigated factor, there could be other unconsidered factors that also affect the result of the investigated factor. In our case study, one such example of threat is that the real-time aspect of the tool will influence the testers in how they formulate the issue, leading to them describing the issue differently than they would have done without the real-time aspect. We mitigated this threat to internal validity by recording and extensive note taking of the observed initial issue description and ask for the participants' explanation of why the recommendations do not match and how they reformulated their issue description in order to get more suitable recommendations.

External Validity concerns the ability to generalise based on the findings. One threat to the external validity of the case study is that the participants of the study were limited to the relatively small number of testers at TestScouts and their perspective and workflow of detecting duplicate bug reports may not be representative of the larger population of testers. Also, the findings relate to the limited datasets used as part of this study may not generalize well to other datasets or bug repositories in general.

Reliability concerns how the data and analysis are dependent on the researchers conducting the study. The result should hypothetically be the same if another researcher conducts the same study. To mitigate this risk, both researchers of this case study were present during all interviews and observations and coded the same data collected independently in order to avoid biased conclusions or questions. Furthermore, we compared our individual findings of codes and themes in our thematic analysis of the collected qualitative data to ensure coherency and agreement.

VIII. Conclusion

The typical workflow of evaluating duplicate bug reports consists of the steps of creating a bug report, identifying duplicates, linking them together and closing them, where the knowledge of the tester is reported to be essential to entire workflow. Bugle was introduced as a state-of-the-art tool to be used in this workflow for helping testers to identify duplicates in real time, and was evaluated in an observational study. The tool showed an accuracy of 92.60%—however, the testers did not always agree with the recommendations made by Bugle, leading to an average accuracy of 75.00% in detecting duplicates. The reasons for the disagreements were varying degrees of semantic differences in bug descriptions, insufficient information, as well as testers overlooking information

provided by Bugle. The testers' disagreement with Bugle's recommendations further prevented false duplicates as being marked as duplicates in 88.89% of the cases.

The quality of bug reports imposes limitations on automated duplicate detection. Since Bugle relies on semantic textual similarity, having bug report textual data available in a semantically coherent format is crucial. The accuracy of duplicate bug report detection should also be evaluated in relation to the number of bug reports falsely marked as duplicates. There is a trade-off when automating the process of detecting duplicate bug reports of maximizing the number of true positives vs. minimizing the number of false positives—a balance that is difficult to strike. By involving the tester in the decision making and letting the tool merely recommend bug reports, Bugle shifts the responsibility of filtering out false recommendations to the tester without decreasing the tool's accuracy.

A. Future Works

This study found that Bugle seems to remove an important barrier to the workflow of evaluating duplicate bug reports—knowledge. The observational data suggest that less experienced testers were in no regard less successful in using Bugle than their more experienced peers. Further research is needed in order to determine if this is a generalizable finding regarding experience level and prerequisite knowledge needed for successful use of Bugle or similar tools.

Further research is also needed in order to determine the impact of different quality bug descriptions for detecting duplicates successfully. In particular, the authors of this paper would be interested in any future work regarding different standards for creating bug reports, as the way in which they are written largely determines the success of large language models in identifying their potential twins.

REFERENCES

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

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

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

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

[5] Bug, definition & meaning, merriam-webster.

[6] Maximilian Flis. 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., 2020.

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

[8] Lyndon Hiew. *Assisted detection of duplicate bug reports*. PhD thesis, University of British Columbia, 2006.

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

[10] Nils Reimers and Iryna Gurevych. 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, 11 2019.

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

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

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

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

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

[16] Thomas Zimmermann, R. Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schröter, and Cathrin Weiss. What makes a good bug report? *IEEE Transactions on Software Engineering*, 36:618–643, 09 2010.

TABLE X
SUB-THEMES RELATED TO KNOWLEDGE.

Main Theme	Sub-theme	Description
MT1	System knowledge	In order to know if a bug report is a duplicate the tester needs to have system knowledge.
MT1	Intuition-based	When identifying a duplicate bug report there is a sense of intuition where a tester "just knows" that the issue is present in the bug report repository.
MT1	Memory-based	Duplicate detection requires the tester to actively remember what is in the bug report repository in order to identify duplicates.

TABLE XI
SUB-THEMES RELATED TO DUPLICATE IDENTIFICATION.

Main Theme	Sub-theme	Description
MT2	Manual keyword searching	A standard operating system command (e.g. Ctrl-F) used to find bug reports within a repository that contain matching words or phrases.
MT2	Team communication	Meetings within the team where discussions are the basis of identifying duplicate bug reports.
MT2	Deep analysis	Experts of the system, could be testers or even system developers who looks into the code in order to identify duplicate bug reports.

IX. APPENDIX

TABLE XII
SUB-THEMES RELATED TO DUPLICATE MANAGEMENT.

Main Theme	Sub-theme	Description
MT3	Creating bug report	Depending of the team and organization size, there are different standards of how to write and create bug reports in order to more easily identify duplicates later on.
MT3	Linking duplicates	There is a consensus across all participants that a bug report closed as a duplicate should be linked for traceability and merging of data.
MT3	Closing bug report	Testers close a bug report once a duplicate has been confirmed.
MT3	Organizational safety criteria	Different criteria for safety determine who should be able to close a bug report as a duplicate.

TABLE XIII
SUB-THEMES RELATED TO SEMANTIC DIFFERENCES IN DESCRIPTIONS.

Main Theme	Sub-theme	Description
MT4	Uncertainty of product/component	There was a mismatch in wording between the provided issue description and the recommended duplicate that had to do with different products or components.
MT4	Key words not included in description	Specific words necessary to determine a duplicate with certainty were not included in the recommended bug report description.
MT4	Slight difference in wording	Very minor differences in phrasing accounting for the reason not to agree with the duplicate recommendation.

TABLE XIV
SUB-THEMES RELATED TO INSUFFICIENT INFORMATION.

Main Theme	Sub-theme	Description
MT5	Vague formulation	Either the provided issue description or the recommended duplicate were reported as being to ambiguous in their description.
MT5	No trace to requirement ID	Testers reported the need to be able to connect the issue description with a requirement ID.
MT5	Accepted level of uncertainty	Testers reporting feeling different levels of certainty with regards to be able to determine the nature of a duplicate.

TABLE XV
SUB-THEMES RELATED TO OVERLOOKING RECOMMENDATIONS.

Main Theme	Sub-theme	Description
MT6	Not reading the recommendation description at all	Automation bias and an overwhelming amount of data was reported to make the tester not read all of the the recommended bug reports.
MT6	Resources and time	Testers reported that they only look at the summary and only if they found that part interesting, they would continue reading its corresponding description field.
MT6	Not reading the description carefully enough	Missing important information in the description resulted in participants failing to identify the duplicates even when duplicates was correctly recommended by the tool.