A Replicated Study on Duplicate Detection: Using Apache Lucene to Search Among Android Defects

Borg, Markus; Runeson, Per; Johansson, Jens; Mäntylä, Mika

Published in:
[Host publication title missing]

DOI:
10.1145/2652524.2652556

2014

A replicated study on duplicate detection: Using Apache Lucene to search among Android defects

Jens Johansson, Markus Borg, Per Runeson and Mika Mäntylä

Abstract

Duplicate detection is a fundamental part of issue management. Systems able to predict whether a new defect report will be closed as a duplicate, may decrease costs by limiting rework and collecting related pieces of information. Previous work relies on the textual content of the defect reports, often assuming that better results are obtained if the title is weighted as more important than the description. We conduct a conceptual replication of a well-cited study conducted at Sony Ericsson, using Apache Lucene for searching in the public Android defect repository. In line with the original study, we explore how varying the weighting of the title and the description affects the accuracy. Our work shows the potential of using Lucene as a scalable solution for duplicate detection. Also, we show that Lucene obtains the best results the when the defect report title is weighted three times higher than the description, a bigger difference than has been previously acknowledged.

1 Introduction

Issue management is an important part of large-scale software evolution. Development organizations typically manage thousands of defect reports in issue repositories such as Jira or Bugzilla. In large projects, the continuous inflow of defect reports can be a hard to overview [2]. An important step in the early issue triaging is to determine whether an issue has been reported before. This activity, known as *duplicate detection*, is typically a manual process. Previous studies show that the fraction of duplicate defect reports range from 10% to 30%, both in proprietary and Open Source Software (OSS) development [1, 7, 5]. Moreover, in projects where also end users can submit defect reports, i.e., not only developers and testers, the amount of duplicates may be even higher [12].

In 2007, we implemented automated duplicate detection at Sony Ericsson [7]. We showed that our prototype tool, based on Information Retrieval (IR) techniques, discovered about 40% of the true duplicates in a large project at the company. Furthermore, we argued that the approach had the potential to find about 2/3 of the duplicate issue reports. Also, we estimated that the tool support could save about 20 hours of issue triaging per 1,000 submitted defect reports.

In our previous study, we also discovered a number of IR configurations that have guided subsequent research on duplicate detection. We found that weighting the title twice as important as the description of an defect report yielded improved results, a finding that has been confirmed in several later studies [12, 5, 9]. Moreover, we found that most duplicate defect reports were submitted within three months of the master report, thus providing us an option to filter the duplicate detection accordingly.

We now return to two of the directions pointed out as future work in the original study. First, we argued that the IR approach

used in our prototype tool should be *compared to an off-the-shelf search engine*. While several papers have been published on duplicate detection in recent years [3], such an evaluation has not yet been conducted. Second, we argued that our work should be *replicated on another dataset*. Several researchers have presented work on other datasets (see Section 2), and also confirmed the value of up-weighting the the title; however a deeper study of the title vs. description balance has not been presented. Also, the usefulness of filtering duplicates based on differences in submission dates have not been confirmed in later studies.

In this study, we evaluate using the search engine Apache Lucene[1] for duplicate detection. Lucene is an OSS state-of-the-art search engine library, often integrated in large-scale full-text search solutions in industry. We study a large set (n=20,175) of defect reports collected from the Android development project, i.e., the dataset originates from the domain of mobile devices for telecommunication, in line with our original study.

The following Research Questions (RQ), the same as in the original study, guide our work:

RQ1 How many duplicate defect reports are found by Apache Lucene?

RQ2 What is the relative importance of the title vs. the description?

RQ3 How can the submission date be used to filter the results?

2 Background and related work

When IR is used for duplicate detection, the query consists of the textual content of the newly submitted defect report and the IR system will search the issue repository for potential duplicates and list them accordingly. Several researchers have published studies on duplicate detection since our original study. Wang *et al.* complemented the textual content of defect reports with stack traces [12]. While their evaluation contained fewer than 2,000 defect reports, they report a Rc@10 (defined in Section 3.3) as high as 93% on data collected from the development of Firefox. Sun *et al.* proposed including also categorical features to represent defect reports [9]. Also, they considered both unigrams (single terms in isolation) and bigrams (sequences of two terms) in the textual description, reaching a Rc@10 of 65-70% on the three OSS projects Firefox, OpenOffice, and Mozilla. Sureka *et al.* presented a different approach to text representation, considering defect descriptions on character level [11]. They did not perform any pre-processing, and thus allow duplicate detection across languages. They evaluated their approach on a random sample of 2,270 defect reports from the Eclipse project, and obtained a Rc@10 of 40%.

Other researchers have instead considered duplicate detection a classification problem. A classifier can be trained on historical defect reports to answer the question "is this new defect report a duplicate of an already known issue?". Jalbert and Weimer complemented IR techniques with training a linear regression model using both textual information and categorical features [5]. The regression model then learns a threshold that distinguishes duplicates and unique issues, reaching a Rc@10 of 45%. Sun et al. [10] instead used SVM, a standard classifier, to detect duplicated defect reports. By considering both unigrams and bigrams in the title and description, they achieve a Rc@10 of about 60%.

3 Method

This section presents the IR tool used, the dataset, and the experimental design.

3.1 Apache Lucene for duplicate detection

We use Apache Lucene version 3.6.1, to develop a prototype tool for duplicate detection.

[1] http://lucene.apache.org/

Lucene provides a scalable solution to index and search text, in our case identify textually similar defect reports. For each defect report in our dataset, Lucene creates a *document* which consists of *fields*, that represent the defect report. The documents are stored in an *index*. Finally, we use other defect reports as search queries to the index and Lucene returns a ranked list of the search results, i.e., potential duplicate defect reports.

Lucene matches queries and documents by calculating similarities. Each search result is given a raw score, i.e., a floating-point number ≥ 0.0. Higher numbers indicate better matches. Also, it is possible to set *boost factors* in Lucene, weighting features as more or less important. The default boost factor for all fields is 1.0, but can be any value ≥ 0. General practice is to find the best boost factors by trial and error.

3.2 Dataset and feature selection

We study a set of 20,175 Android defect reports submitted between November 2007 and September 2011, publicly available as part of the MSR Mining Challenge 2012 [8]. We represent a defect report by the textual content of the title and the description, i.e., as two separate bags-of-words. We apply stop word removal and stemming to the text, using built-in functionality in Lucene. Moreover, we also consider the submission date. We planned to include also categorical features such as the severity and component, but pilot runs showed that they did not add any predictive value for the duplicate detection task.

The dataset contains 1,158 defect reports with duplicates. However, a defect report can have several duplicates. Also, we observed that some bug reports have high numbers of duplicates, 160 defect reports have as many as 20 duplicates or more. In the dataset, duplicates are signaled using one-way references to the master report. By translating all duplicate links to two-way relations, and by using the transitive property of equality[2], we obtain in total 16,050 duplicate relations. More than half of the duplicate defect reports were submitted within 20 days, and almost 75% within 200 days.

3.3 Measures

The most commonly reported measure used for evaluating duplicate detection in issue management is the recall when considering the top-n recommendations (Rc@N) [3], i.e., how many of the duplicates are found within the top 1, 2, 3, ... 10 recommendations (as a cut-off). Thus, Rc@N shows the fraction of the true duplicate reports actually presented among the top N recommendations. We restrict the cut-off level to 10 as that is a reasonable number of duplicate candidates for a developer to assess.

We complement this measure with Mean Average Precision (MAP), a popular IR measure that provides a single-figure result reflecting the quality of a ranked list of search results. Within IR research, MAP is known to be a stable measure good at discriminating different approaches [6]. Also, we have previously argued that MAP, a so called secondary measure for IR evaluation, should be used more often in software engineering research [4]

We define Average Precision (AP) until the cut-off N as:

$$AP@N = \frac{\sum_{r=1}^{n}(P(k) \times rel(k))}{number\ of\ relevant\ documents} \quad (1)$$

where $P(k)$ is the precision at cut-off k, and $rel(k)$ is equal to 1 if the search result at rank k is relevant, otherwise 0. The MAP is calculated over all the queries in the dataset. For duplicate detection relevant documents turn into true duplicates.

3.4 Experimental design and validity

We perform two controlled experiments, ExpA and ExpB, to tackle our RQs. For both experiments we use all 20,175 defect reports as search queries in Lucene once, and evaluate the

[2] if a is duplicate of b and b if duplicate of c, then a is duplicate of c

ranked output at the cut-off 10. The dependent variables are in both experiments the Rc@10 and MAP@10.

First, ExpA evaluates the performance of duplicate detection of defect reports among Android defect reports using Lucene (RQ1). To study the relative importance of the title vs. the description (RQ2) we used Lucene boost factors as the independent variables. We explore the effect of different boost factors in a systematic manner, varying the boost factors from 0 to 4.9 in steps of 0.1. For all combinations of boost factors we perform an experimental run with Lucene, in total 2,500 experimental runs.

ExpB addresses RQ3, i.e., whether it is useful to filter the results based on submission dates. We evaluate removing duplicate candidates returned by Lucene, keeping only duplicate candidates within a specific time window, in line with the original study [7]. We fix the boost factors based on the results of ExpA. Then we use the size of the time window as the independent variable, varying between 1 and 1,141 days (the longest time between a duplicate and a master defect report in the dataset). We consider the defect report used as the query as being in the middle of the time window.

We consider the following validity threats as the most relevant for our study: 1) The share of duplicate defects in the Android database is only roughly 5% whereas prior work report that typically 10-30% are duplicates. This atypical share of duplicate defects might make comparison to prior work difficult. 2) The way we establish our gold standard is different. Our dataset does not have fine-granular time stamps, i.e., only dates are available, thus we cannot not determine a master report as in previous work. In our sets of duplicates, typically several defect reports have the same submission date. We are restricted to a set-based evaluation, which is likely to boost precision values but decrease recall. Still, we argue that a high-level comparison to the results in previous work can be made.

Rc@10	**Title boost**	**Desc. boost**	**Title/Desc.**
0.138	1.9	0.8	2.375
0.138	2.6	1.1	2.364
0.138	3.4	1.1	3.091
0.138	3.7	1.2	3.083
0.138	3.1	1.3	2.385
0.138	3.3	1.4	2.357
0.138	3.8	1.6	2.375
0.138	4.0	1.7	2.353
0.138	4.3	1.8	2.389
0.138	4.5	1.9	2.368
		Baseline	
0.134	1.0	1.0	1.0

Table 1: Best relative weighting of title vs. description wrt. Rc@10. The baseline reflects the Rc@10 when using balanced weighting.

4 Results and discussion

In ExpA, the highest Rc@10 we achieve is 0.138, i.e., detecting 2,215 out of 16,050 cases where duplicates are listed among the top ten candidates. Ten different combinations of boost factors yield this result, as presented in Table 1. Thus, we answer RQ1 by: *Lucene finds more than 10% of the true duplicates among the top-10 duplicate candidates.*

The performance of our approach appears significantly lower than what has been reported in previous work. However, while previous work has assumed the simplification that every duplicate report has exactly one master report, typically the first one submitted, we instead consider sets of duplicates. Thus, every duplicated defect report might have several different reports to find, many more than if using the simplifying assumption. The Rc@10 we obtain reflect the more challenging gold standard used in our evaluation.

ExpA confirms the claim by previous research, stating that up-weighting the title over the description provides better results in defect duplicate detection. Although the improvement of overweighing the title is small, 3.0 and 2.8% for Rc@10 and MAP@10, respectively, those improvements are greater than in our original work where improvement was 1.2% [7]. Table 2 lists the ten best combinations of boost factors wrt. MAP@10. We see that the best results are achieved when the title is given three times as high weight as the description, a

MAP@10	Title boost	Desc. boost	Title/Desc.
0.633	4.9	1.6	3.063
0.633	4.6	1.5	3.067
0.633	4.3	1.4	3.071
0.632	3.1	1.0	3.100
0.632	2.5	0.8	3.125
0.632	2.2	0.7	3.143
0.632	4.4	1.4	3.143
0.632	4.7	1.5	3.133
0.632	2.8	0.9	3.111
0.632	2.9	0.9	3.222
		Baseline	
0.616	1.0	1.0	1.0

Table 2: Best relative weighting of title vs. description wrt. MAP@10. The baseline reflects the MAP@10 when using balanced weighting.

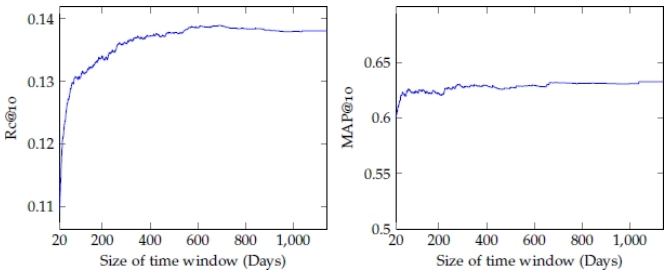

Figure 1: Rc@10 and MAP@10 when considering duplicate candidates within a varying time window.

bigger difference than has been acknowledged in previous work. Our answer to RQ2 is: *the title is more important than the description when detecting duplicates using IR; for Android defect reports it should be given thrice the weight.*

We hypothesize two main reasons for this difference. First, submitters of defect reports appear to put more thought into formulating the title, carefully writing a condensed summary of the issue. Second, less precise information is added as parts of the description. Stack traces, snippets from log files, and steps to reproduce the issue might all be useful for resolving the defect report, but for duplicate detection based on IR it appears to introduce noise. On the other hand, it is still beneficial to use the textual content in the description, but it should not be given the same weight.

ExpB uses output from ExpA to set promising boost factors. We use two weight configurations to up-weight the title in relation to the description: 2.5 (ConfA) and 3.1 (ConfB). Figure 1 shows the effect of using different time windows. The left subfigure shows the effect on Rc@10 for ConfA, the right subfigure displays how MAP@10 is affected for ConfB. Our results do not show any improvements of practical significance for filtering duplicates based on the submission time. Instead, it appears that considering the full dataset as potential duplicates gives the best results. Our results do not confirm the results from the original study, that only considering an interval of 60 days is beneficial [7]. We answer RQ3 by *duplicate detection among Android defect reports does not benefit from filtering results based on the submission date.*

As reported in Section 3.2, more than 50% of the duplicate defect reports in the Android dataset were submitted within 20 days, and almost 75% within 200 days. Thus we expected filtering to have a positive effect on the precision of our ranked lists. This was not at all the case; instead it appeared consistently better to skip filtering, in contrast to the finding in the original study. The filter appears to not remove any meaningful noise in our case, but instead remove meaningful candidates. This could either be because differences in the dataset, or in the IR tool used. No matter what, our recommendation is that *if the textual similarity is high, consider the defect report a potential duplicate even if it was submitted at a distant point in time.*

5 Summary and future work

In this paper, we make three important findings in duplicate defect detection. *First* we *confirm* the prior findings that weighting the title more important than the description can significantly improve duplicate defect detection. Since this confirmation comes from a different dataset and with the use of another tool, we think this result might be generalizable. Yet, future studies are needed before we can make definitive conclusions. *Second*, we present *conflicting* evidence on the benefits of

filtering defect reports based on the submission date. This suggests that the benefits of time based filtering are dependent on the context. *Third*, we open a *new avenue for future work* by considering all duplicate relations whereas the prior work has only considered that every defect has a single master report. The use of all duplicate relations led to much poorer Rc@10 when compared with prior work. However, since our approach is more realistic we suggest that future works follow our more complex scenario.

Acknowledgement

This work was funded by the Industrial Excellence Center EASE – Embedded Applications Software Engineering.

References

[1] J. Anvik, L. Hiew, and G. Murphy. Coping with an open bug repository. In *Proc. of the OOPSLA workshop on Eclipse technology eXchange*, pages 35–39, 2005.

[2] N. Bettenburg, R. Premraj, T. Zimmermann, and K. Sunghun. Duplicate bug reports considered harmful... really? In *Proc. of the Int'l Conf. on Software Maintenance*, pages 337–345, 2008.

[3] M. Borg and P. Runeson. Changes, evolution and bugs - recommendation systems for issue management. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *Recommendation Systems in Software Engineering*. Springer, 2014.

[4] M. Borg, P. Runeson, and L. Brodén. Evaluation of traceability recovery in context: a taxonomy for information retrieval tools. In *Proc. of the Int'l Conf. on Evaluation & Assessment in Software Engineering*, pages 111–120, 2012.

[5] N. Jalbert and W. Weimer. Automated duplicate detection for bug tracking systems. In *Proc. of the Int'l Conf. on Dependable Systems and Networks*, pages 52–61, 2008.

[6] C. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[7] P. Runeson, M. Alexandersson, and O. Nyholm. Detection of duplicate defect reports using natural language processing. In *Proc. of the Int'l Conf. on Software Engineering*, pages 499–510, 2007.

[8] E. Shihab, Y. Kamei, and P. Bhattacharya. Mining challenge 2012: The Android platform. In *Proc. of the Working Conf. on Mining Software Repositories*, pages 112–115, 2012.

[9] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proc. of the Int'l Conf. on Automated Software Engineering*, pages 253–262, 2011.

[10] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. In *Proc. of the Int'l Conf. on Software Engineering*, pages 45–54, 2010.

[11] A. Sureka and P. Jalote. Detecting duplicate bug report using character n-gram-based features. In *Proc. of the Asia Pacific Software Engineering Conf.*, pages 366–374, 2010.

[12] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. An approach to detecting duplicate bug reports using natural language and execution information. In *Proc. of the Int'l Conf. on Software Engineering*, pages 461–470, 2008.