The Impact of Cross-Distribution Bug Duplicates, Empirical Study on Debian and Ubuntu

Vincent Boisselle, Bram Adams
MCIS, Polytechnique Montréal, Québec, Canada

Abstract—**Although open source distributions like Debian and Ubuntu are closely related, sometimes a bug reported in the Debian bug repository is reported independently in the Ubuntu repository as well, without the Ubuntu users nor developers being aware. Such cases of undetected cross-distribution bug duplicates can cause developers and users to lose precious time working on a fix that already exists or to work individually instead of collaborating to find a fix faster. We perform a case study on Ubuntu and Debian bug repositories to measure the amount of cross-distribution bug duplicates and estimate the amount of time lost. By adapting an existing within-project duplicate detection approach (achieving a similar recall of 60%), we find 821 cross-duplicates. The early detection of such duplicates could reduce the time lost by users waiting for a fix by a median of 38 days. Furthermore, we estimate that developers from the different distributions lose a median of 47 days in which they could have collaborated together, had they been aware of duplicates. These results show the need to detect and monitor cross-distribution duplicates.**

I. Introduction

Early January 2011, a developer reported bug #697498 in the Ubuntu bug repository, after which the *triager* confirmed the validity of the bug, checked for duplicate bug reports in the Ubuntu bug repository, then dispatched the bug to the assignee best suited to fix the bug. However, nobody noticed that the bug was already reported 3 months earlier in the mother distribution Debian (bug #599582). Unknowingly, developers in both distributions kept on working in parallel on a patch. Eventually, the Debian developers figured out a patch at the end of July 2011 (fixed, but not released). At this point, developers in Ubuntu were still stuck with the bug, until 9 months later, when they finally found the fix at the end of April 2012. In this situation, the Ubuntu users who were victim of the bug lost 9 months of useless time waiting for a fix because the people involved were not aware of the bug fix presence in Debian. Developers in both distributions also lost a potential collaboration time of 8 months, where they worked in isolation on a fix for the same issue instead of being able to join forces.

The above is a typical example of time loss occurring when dealing with bug duplicates across bug repositories of open source systems, yet finding duplicates across an organization's border is hard. Indeed, whereas 44% of Ubuntu bug reports have been linked to a within-project (i.e., Ubuntu) duplicate, an analysis of the Debian and Ubuntu bug repositories shows that less than 2% of the Ubuntu bug reports have been linked manually to at least one Debian bug. This gap could either be explained by the fact that cross-organization duplicates are rare or that people are actually good at finding within-project duplicates, but not detecting duplicates across an organization's borders.

Indeed, to help triagers find duplicates inside their organization's bug repository [3], various researchers have proposed duplicate bug report detection tools [2], [13], [19], [17], [14], [18], [12], [20]. These approaches, upon submission of a new bug report, provide a recommendation about similar bug reports based on the subject, description or even characteristics of the affected code. Triagers then need to check the top recommendations, potentially recording identified duplicates inside the bug report. These techniques are starting to become ready for real-life usage, for example Ubuntu's bug repository has such a technique built-in for new bug report submissions. However, the above techniques have not been designed for finding duplicates across repositories of different organizations. Organizations typically use different bug repository technologies, which have some overlapping data fields, but also unique data fields on either side.

This paper aims to empirically study the performance of an existing bug duplicate detection approach on cross-distribution duplicates, as well as to estimate the time lost due to missed cross-distribution duplicates. For this, we first build a cross-project duplicate detection model based on an existing information retrieval (IR) approach, then we apply it on two large, popular open source distributions (Debian and Ubuntu) to answer the following RQs:

RQ1: What is the performance of an IR-based duplicate detection technique across two distributions? *Our approach detects duplicates across distributions with a maximum recall of 60% using a threshold of 60% (with precision of 5%), while we get an optimal precision and recall of 43% when using a threshold of 88%.*

RQ2: How much time is lost by users and developers during the fixing process? *While fixing a bug typically takes a median time of 29 days, developers lose a median of 47 days of potential collaboration and users lose 38 days waiting for fixes already made in the other distribution.*

II. Background and Related Work

Ray et al. perform a case study on cross-system porting (i.e. applying patches to other products) in the open source BSD ecosystem [16], in which they found that porting occupies a significant portion of the BSD family evolution and involves a significant portion of the active committers. Part of the patches ported to other BSD products consist of bug fixes, implying that similar bugs are reported across distributions. Crowston et al. study the social interactions of developers fixing bugs in OSS projects and found a lack of coordinations between developers, where, as a consequence, work load is not equally distributed among developers and some of them work on the same patch without collaborating and exchanging productive information that could empower the bug fixing process[8]. Lack of coordination between developers and other stakeholders is a typical OSS bug fixing challenge that also holds for OSS distributions, which share multiple packages requiring coordination across organizations, and hence introduce significant maintenance cost caused by the integration activities. Considering OSS distributions, Adams et al. identified seven major integration activities such as the Local Patch, which consists of patching an upstream source code locally. Doing local patching with a lack of collaboration with the related upstream organization can cause major issues, for example there is a well known case of OpenSSL security breach[1]. Since bug repositories are the principal tools used to manage patching activities, optimizing the bug fixing process can reduce the cost of integration and maintenance activities, and decrease time loss by stakeholders working with cross-distribution projects.

Multiple studies show an interest for bug repositories to attack the problem of software maintenance. Anvik et al. were among the first to study open source bug repositories and to report on the major challenges faced by triagers and developers of these projects. These challenges consist of dealing with a high volume of bug reports, part of which are invalid or duplicates. To ease the work of triagers, many researchers propose automated techniques based on information retrieval to recommend a top list of potential bug duplicates within bug repositories [12], [18], [14], [13]. All those works share the same basic technique to recommend a top N bug duplicates list by extracting textual features (e.g, title and description), then compare those features using techniques such as TF-IDF or BM25F [9], which yields a similarity value ranging from zero (totally not similar) to one (exactly the same). Sun et al. propose a multiple features approach, which combines textual features and numerical features comparing two reports (e.g., delta of bug priority numerical values), then get better results than traditional textual approaches obtaining a maximal recall of 71% [18].

Other researchers observe that most of the duplicates are reported within a certain time scope around the first duplicate publish date, thus they focus the recommendation of bug duplicates on a specific time period, which can provide a gain up to 24% on the recall compared on a basic TF-IDF approach encompassing the whole bug repository [11], [14]. Most of these previous studies analyze open source bug repositories based on the Bugzilla platform.

To the best of our knowledge, no research so far has dealt with cross-project duplicates, or analyzed how those techniques could impact the time wasted by end users and software maintainers.

[1]http://lwn.net/Articles/282038/

III. Methodology

A. Study Setup

Our study takes place on the Ubuntu and Debian Linux distribution bug repositories. A Linux distribution packages an operating system kernel (Linux) with thousands of open source end user applications in order to distribute an integrated work environment to its users. The reason why we perform our study on cross-project duplicate detection on Linux distributions in general, and Ubuntu and Debian specifically, is because distributions can be derived from other distributions. In this case, Ubuntu inherits the tools and packages of the Debian distribution, then adds and customizes additional packages on its own. As such, Ubuntu depends heavily on Debian, such that being aware of duplicate bugs in Debian is an important goal. Furthermore, according to the unofficial distribution popularity ranking of distrowatch.com, Ubuntu and Debian are the second and third most popular distributions at the time of writing (behind Mint, which is derived from Ubuntu but performs substantially less customization on Ubuntu than Ubuntu does on Debian).

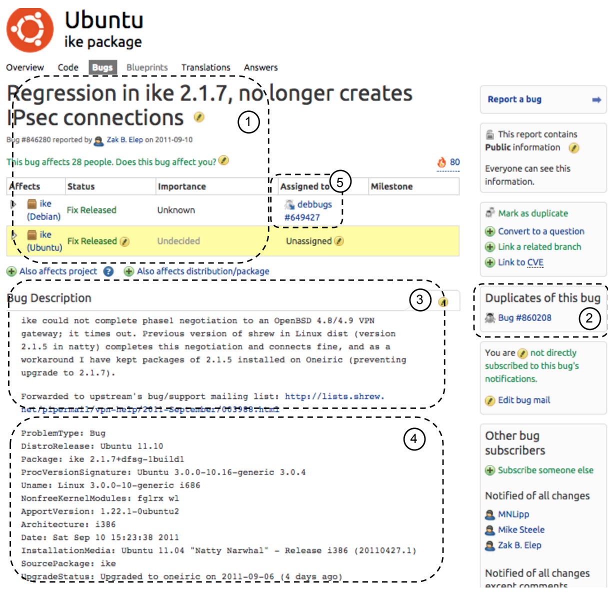
The Ubuntu community uses the Launchpad web platform to create bug reports and keep track of them. The Debian community uses a more basic approach using a mail server. The launchpad platform provides trackers to track bugs from another provider such as a related distribution like Debian or a package provider. Such trackers need to be enabled manually, which means that a human first needs to become aware of a duplicate bug report before being able to use a tracker.

We mined the Debian bug repository from 2009 to 2013 and the Ubuntu bug repository from 2011 to 2013. We select a wider dataset for Debian because Debian bug reports tend to be older (i.e., were reported earlier) than their Ubuntu duplicate. We have selected datasets that are recent enough to develop an approach applicable on actual bug reports and old enough to have most of the bugs fixed.

B. Identifying Common Bug Report Fields

In general, bug reports are characterized by a set of different fields that capture different kinds of information about a bug. Each bug duplicate detector tool uses a subset of these fields to compare a new bug report to existing ones to know whether the reports are similar or not. Hence, as a first step, we need to find a set of representative fields that are common across both distributions under study.

The most common bug report fields are the description, title, date, and the author's name. Sun et al. [18] added a variety of more specialized fields to build a bug report similarity model, including product, component, version, priority, and type of bug report. These new fields can enhance a detector's similarity model, however they require the fields to be available in both bug repositories, which represents a challenge in the case of cross-project duplicate detection. Figure 1 and 2 show bug report examples for Ubuntu and Debian respectively.



Regression in ike 2.1.7, no longer creates IPsec connections

Bug #846280 reported by Zak B. Elep on 2011-09-10

This bug affects 28 people. Does this bug affect you?

Affects	Status	Importance	Assigned to	Milestone
ike (Debian)	Fix Released	Unknown	debbugs #649427	
ike (Ubuntu)	Fix Released	Undecided	Unassigned	

Bug Description

```
ike could not complete phase1 negotiation to an OpenBSD 4.8/4.9 VPN
gateway; it times out. Previous version of shrew in Linux dist (version
2.1.5 in natty) completes this negotiation and connects fine, and as a
workaround I have kept packages of 2.1.5 installed on Oneiric (preventing
upgrade to 2.1.7).

Forwarded to upstream's bug/support mailing list: http://lists.shrew.
net/pipermail/vpn-help/2011-September/003988.html

ProblemType: Bug
DistroRelease: Ubuntu 11.10
Package: ike 2.1.7+dfsg-1build1
ProcVersionSignature: Ubuntu 3.0.0-10.16-generic 3.0.4
Uname: Linux 3.0.0-10-generic i686
NonfreeKernelModules: fglrx wl
ApportVersion: 1.22.1-0ubuntu2
Architecture: i386
Date: Sat Sep 10 15:23:38 2011
InstallationMedia: Ubuntu 11.04 "Natty Narwhal" - Release i386 (20110427.1)
SourcePackage: ike
UpgradeStatus: Upgraded to oneiric on 2011-09-06 (4 days ago)
```

This report contains **Public** information

Everyone can see this information.

Duplicates of this bug

Bug #860208

You are not directly subscribed to this bug's notifications.

Other bug subscribers

Notified of all changes

MNLipp

Mike Steele

Zak B. Elep

Notified of all changes except comments

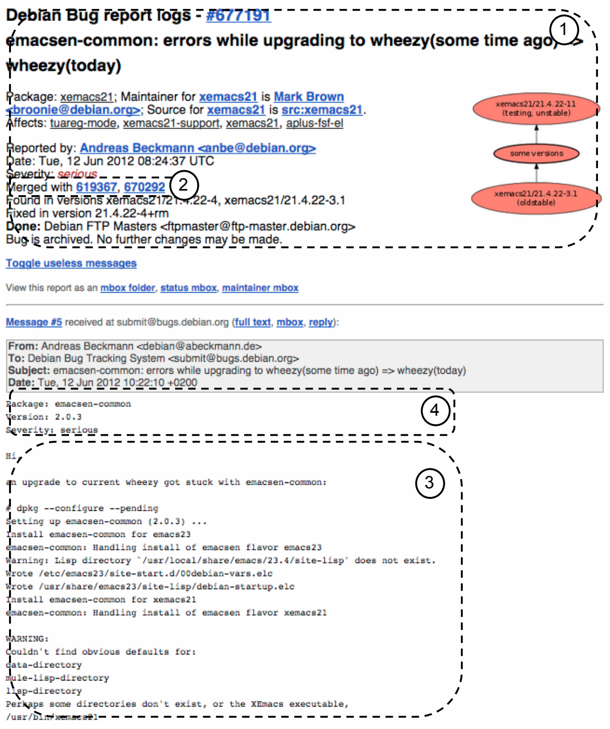
Figure 1: Ubuntu bug report #846280

On the Ubuntu bug report (Figure 1), we have access to a variety of fields in zone #1, where we find the title, bug number, author name, creation date, affected packages, status, and importance. Zone #2, contains links to related duplicates within the same bug repository, while zone #3 is the general description field of the bug. Zone #4 contains multiple fields (some already in zone #1) that provide more precise information about the bug, and zone #5 contains links to different trackers listening to a bug reported elsewhere (Debian in this example). Fields in zone #4 are delimited by the ":" character, where the left part is the field's name and the right part its value, e.g., in Figure 1 "Architecture" is a field's name and "i386" its value.

The Debian bug report (Figure 2) has the same kind of information as Ubuntu in the delimited zones #1, #3, and #4, while zone #2 contains links to bug duplicates inside the Debian repository. The Debian bug report contains less information than Ubuntu, especially in zone #3. This represents one of the essential problems of cross-project duplicate detection. To find the fields that are common to both repositories, we counted the number of occurrences of non-empty fields that are present in the analyzed bug reports of each repository, then manually mapped the most frequent ones to each other as shown in Table I. To avoid losing too much information when parsing report text into pre-selected fields, we leave out other less frequent sub-fields in the "misc" field.

C. *Bug Report Similarity Model*

To build a cross-distribution duplicate detection approach, we first need to extract the bug report data and build a model to calculate the level of similarity between bug reports. Later on, such a model can then be used to determine whether a new bug report in one distribution is similar enough to a bug report of another distribution. This section describes step by step the process to generate the model, in which we first


1
2
4
3
xemacs21/21.4.22-11 (testing, unstable)
some versions
xemacs21/21.4.22-3.1 (oldstable)

Debian Bug report logs - #677191

emacsen-common: errors while upgrading to wheezy(some time ago) => wheezy(today)

Package: xemacs21; Maintainer for xemacs21 is Mark Brown <broonie@debian.org>; Source for xemacs21 is src:xemacs21.
Affects: tuareg-mode, xemacs21-support, xemacs21, aplus-fsf-el

Reported by: Andreas Beckmann <anbe@debian.org>
Date: Tue, 12 Jun 2012 08:24:37 UTC
Severity: *serious*
Merged with 619367, 670292
Found in versions xemacs21/21.4.22-4, xemacs21/21.4.22-3.1
Fixed in version 21.4.22-4+rm
Done: Debian FTP Masters <ftpmaster@ftp-master.debian.org>
Bug is archived. No further changes may be made.

Toggle useless messages

View this report as an mbox folder, status mbox, maintainer mbox

Message #5 received at submit@bugs.debian.org (full text, mbox, reply):

From: Andreas Beckmann <debian@abeckmann.de>
To: Debian Bug Tracking System <submit@bugs.debian.org>
Subject: emacsen-common: errors while upgrading to wheezy(some time ago) => wheezy(today)
Date: Tue, 12 Jun 2012 10:22:10 +0200

```
Package: emacsen-common
Version: 2.0.3
Severity: serious

Hi,

an upgrade to current wheezy got stuck with emacsen-common:

# dpkg --configure --pending
Setting up emacsen-common (2.0.3) ...
Install emacsen-common for emacs23
emacsen-common: Handling install of emacsen flavor emacs23
Warning: Lisp directory `/usr/local/share/emacs/23.4/site-lisp' does not exist.
Wrote /etc/emacs23/site-start.d/00debian-vars.elc
Wrote /usr/share/emacs23/site-lisp/debian-startup.elc
Install emacsen-common for xemacs21
emacsen-common: Handling install of emacsen flavor xemacs21

WARNING:
Couldn't find obvious defaults for:
data-directory
mule-lisp-directory
lisp-directory
Perhaps some directories don't exist, or the XEmacs executable,
/usr/bin/xemacs21
```

Figure 2: Debian bug report #677191

Table I: Selected bug report fields common to both repositories

Field	Description	Example Value
Description	Bug general description	"This is an..."
Misc	Fields in zone #4 not common to both distributions	"InstallationMedia: Ubu..."
Title	A short bug summary	"It crashes!"
Date	Bug creation date	2013-06-05
Package	Problematic package	emacs
Version	Package version	1.0
Architecture	Processor architecture	amd64
Severity	Bug severity level	important

extract bug report fields for comparison, then generate an oracle, and finally build the bug report similarity model. The whole process is shown in Figure 3.

1) *Attribute Extraction:*

a) Repository Updater and Ubuntu/Debian Parsers: The repository updater continuously fetches new bug reports from both Debian and Ubuntu distributions. Because the bug reports of both repositories do not have the same format, we need to parse them using a specialized parser for each distribution to parse the information provided by the bug report into multiple fields. For the Ubuntu distribution, we use the Launchpad API[2], while for Debian the only way to extract fields is using regular expressions on the bug report emails. The parser also retrieves information about already reported bug duplicates. In Debian, duplicates are most of the time from the same distribution. However, in Ubuntu one can track duplicates

[2]Launchpad API: https://launchpad.net/+apidoc/

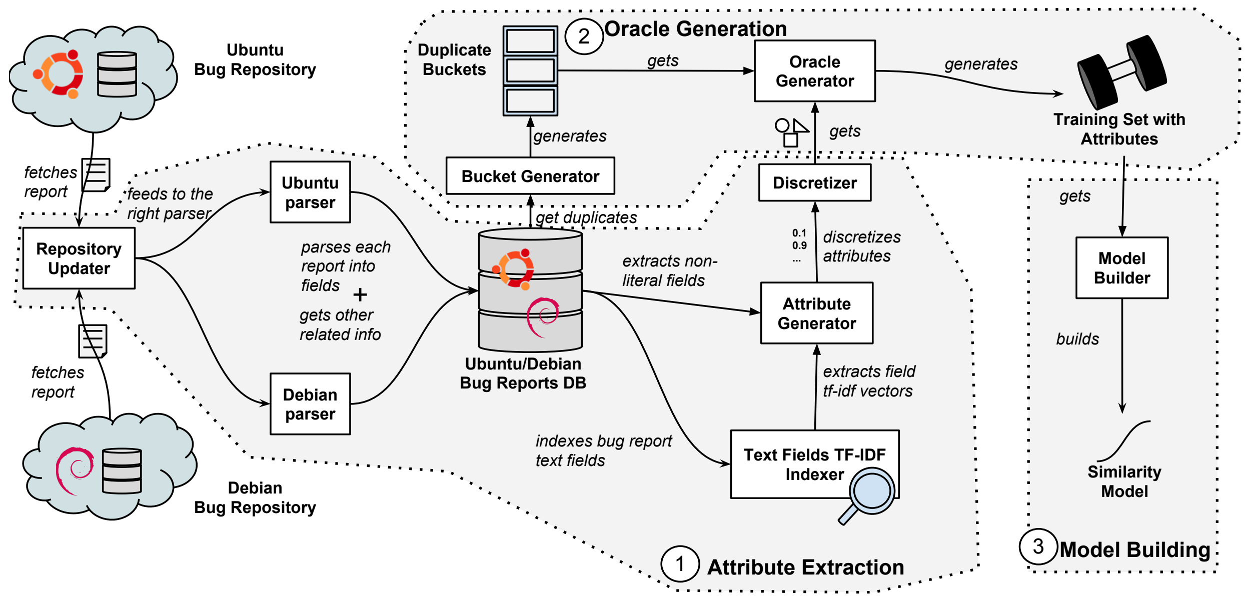


Figure 3: Approach to generate bug report similarity model.

to other distributions, provided developers became aware of these. To distinguish Ubuntu bug report identifiers (id) from the Debian ones, we add a small letter to the bug id, i.e., the Debian bug #677191 is now #d677191 and the Ubuntu bug #846820 is now #u846820. Once a bug report is parsed into its fields, we push it to the Ubuntu/Debian database.

b) TF-IDF Indexers: Using extracted fields, we adapt a well-established within-project duplicate detection approach for cross-distribution detection. Most of the studies so far use text similarity to compare textual fields such as the description and the title of a bug [19], [13], [14]. To do so, they use the popular Term Frequency-Inverse Document Frequency (TF/IDF) weighting technique [9], which *"is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus"* [15].

For each textual field, we create a corpus containing all the values for the field stored in the database, after applying tokenization, stemming, and stop words removal. We then apply the TF/IDF, which generates one term weighting vector for each textual field of a bug report. We can then measure the traditional similarity between field vectors of different bug reports by calculating the cosine distance between both vectors: a value of 0 means that the fields have nothing in common and 1 means that the fields are the same. We developed the indexers using the text feature extraction tools from the Scikit-Learn Python package[3].

c) Attributes, Attributes Generator, and Discretizer: For a given pair of Debian and Ubuntu bug reports, the Attribute Generator compares each report's fields, then generates one attribute with the outcome of each comparison. For example, there is one generated attribute for the title comparison, date comparison, etc. For most of the fields, this comparison is straight-forward. For textual fields, we use the cosine distance between the corresponding TF/IDF vectors. For the date field, we compare two dates by getting the absolute value of their time delta, e.g., 2012-04-06 - 2012-04-04 = time delta of 2 days.

Eventually, each pair of reports yields one row of comparison metrics in a CSV file. Because our bug report similarity model works better with discretized data (see subsection F), we also need a Discretizer that takes numerical attributes and converts them into categorical data. For this, we use Weka's[4] equal-frequency algorithm. This algorithm automatically clusters numeric data into K groups containing a 1/K proportion of the data. We discretize our data in five groups (K = 5) labeled from "L1" to "L5", which makes the data easy to understand for the machine learning algorithm that we use.

2) Oracle Generation:

d) Buckets and Bucket Generator: Sun et al. [18] use the bucket technique with which it is possible to generate an oracle to train the similarity model. This technique, as shown in Table II, consists of finding all the bug reports in the repository that are already reported as duplicates and putting those in the same group ("bucket"). In each bucket, we use one responsible master bug report as identifier of the bucket. The master bug reports in Ubuntu track duplicates from all distributions (see zone #2 and #5 on Figure 1). Such reports can be identified by looking whether they have trackers or reported bugs in the "Duplicate of this" field. Ubuntu bugs that are duplicates of another bug report do not possess links to the other duplicates, they only link back to their master report. In Debian, all

[3]http://scikit-learn.org/stable/

[4]http://www.cs.waikato.ac.nz/ml/weka/

Table II: Bug duplicate buckets example, based on Sun et al. [18] technique

Bucket Masters	u846280	u789706
Duplicates	u860208, d649427 d619367, d670292	d619367, d619367 u917601, u976781

Table III: Example oracle for a part of bucket #u846280 in Table II. The green rows contain pairs of duplicates, whereas the white rows contain random pairs. There are missing pairs for #d619367 and #d670292.

id 1	id 2	desc	misc	title	date	pack.	vers.	arch.	sev.	dup
u846280	u860208	L4	L3	L3	L2	L2	L2	L2	L2	1
u846280	u789706	L2	L1	L2	L3	L1	L1	L1	L1	0
u860208	d649427	L4	L3	L3	L2	L2	L2	L2	L2	1
u860208	d619367	L1	L1	L2	L3	L1	L1	L1	L1	0
d649427	d619367	L4	L3	L3	L2	L2	L2	L2	L2	1
d649427	d619367	L2	L2	L2	L2	L1	L1	L1	L1	0
...	...	...	...	...	...	...	...	...	...	...

duplicate bug reports possess links to their duplicates, hence we do not have to select a master for them.

For example, the Ubuntu bug report in Figure 1 and the Debian bug report on Figure 2 are cross-distribution duplicates. The Debian report #649427 has 2 duplicates (#619367 and #670292) as shown in zone #2 and the master Ubuntu report has a duplicate in the same distribution (#860208) and one in Debian (#649427). To build the cross-distribution bucket, we first look for the duplicates in the Ubuntu master report, we create a new bucket with its id, then put the other bug duplicates in it. We see on the Debian report that there are two other duplicates listed in zone #2, hence we also put them in the same bucket, where we get the final bucket as depicted in Figure 1 for #u846280.

e) Oracle and Oracle Generator: Using buckets, we can build an oracle for our bug report similarity model. We use the same algorithm as Sun et al. [18], where we iterate through each bucket that is not used in our test set (see section IV) to obtain the generated attributes for pairs of duplicates in the bucket (see Attribute Generator). For each duplicate, we generate two such pairs, i.e., a first one containing another duplicate in the same bucket, and a second one containing a randomly picked bug report id from outside the bucket. By doing this, we balance our training set 50/50 with duplicates and non-duplicates, which is a common practice in machine learning.

Table III shows an example oracle that we obtain by running the algorithm only on bucket #u846280 of Table II (in practice, we consider almost all buckets). In the table, each report of the bucket is compared to another duplicate (green rows) and a randomly picked bug in the database (white rows). When a bug report is a duplicate, we set the “dup” attribute to 1, otherwise to 0. As discussed before, the Attribute Generator generates discrete attributes from “L1” to “L5” on an ordinal scale (low to high).

3) Model Building:

f) Similarity Model and Model Builder: We use logistic regression to build a bug report similarity model based on the oracle [10]. This machine learning technique is used to predict a binary value (i.e., duplicate or not) from independent variables (i.e., bug report attributes), as represented by the following formula:

$$\text{Similarity} = \frac{1}{1+\exp^{(\beta_0+\sum_{i=1}^{8}\beta_i a_i)}} \quad (1)$$

Here β_0 is a constant and the β_i are the weights learnt for the attributes a_i using Weka’s Simple Logistic Regression algorithm. The logistic regression model yields a probability value ranging from 0 to 1, and hence does not automatically decide whether a bug report is a duplicate or not. To decide whether a bug report is a duplicate, we use the methodology of the next subsection.

D. Bug Duplicate Recommender

To recommend bug duplicates (Figure 4), we use the Attribute Extraction and Similarity Model of Figure 3. A triager submits a new bug report query to the system to retrieve a top rank list of recommended reports that are the most prone to be duplicates. The Attribute Extraction takes care of everything related to the processing of the submitted bug report from parsing to the generation of discretized attributes on each bug report available in the Debian/Ubuntu database. For example, if someone submits a report query while there are 300,000 bugs in the Debian/Ubuntu, there will be 300,000 sets of attributes generated for a new bug report.

The Recommender then generates a top rank list of possible duplicates by first calculating the similarity level of each bug report in the repository with the new report. For this, each generated set of attributes (one at a time) is given as input to the similarity model, then the outcome of the model (a probability of similarity) is ranked in descending order to obtain the top rank list of duplicate bug candidates. We set a limit of 15 bug reports and use a selective threshold to refine the results, based on prior works (average limits from 1 to 20 bug reports are common [13], [14], [19]. The triager then manually analyzes the top list until a relevant duplicate is found and the new bug is recorded as duplicate, or until the triager decides that the new report is not a duplicate and she discards the top rank list.

IV. Results

Using the data and approach presented in the previous section, we now address the research questions of the introduction. For each RQ, we provide a motivation, approach and our findings.

A. RQ1: What is the performance of an IR-based duplicate detection technique across two distributions?

Motivation:

Before we can study the impact for users and developers of not being aware of cross-duplication duplicates, we first need to identify such duplicates. In addition to an existing set of recorded duplicates (used as oracle for the similarity

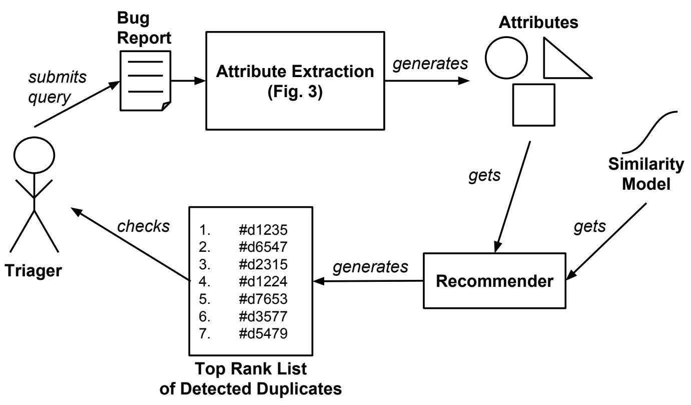


Figure 4: Bug Duplicate Recommender

model), we want to use the duplicate detection technique presented in the previous section to find additional (un-recorded) duplicates. However, this requires that the approach does not generate too many false alarms (i.e., has high precision) and that it finds a large enough number of all un-recorded duplicates (i.e., has high recall). For this reason, this RQ evaluates the precision/recall of the detection approach for retrieving Debian bug report duplicates of Ubuntu bug reports using multiple thresholds, after which we pick a threshold that we believe is the best to provide relevant new duplicates for RQ2.

Approach:

Although our approach is able to find duplicates from both distributions, we focus our evaluation on the perspective of the Ubuntu users, where we submit queries related to Ubuntu bug reports and we retrieve their Debian duplicates. We do this because we have access to more information about the cross-distribution duplicates with Ubuntu Launchpad than with the Debian mail server. However, the approach could equally work the other way around, starting with a new Debian bug report and trying to find possible Ubuntu duplicates.

Hence, we submit existing Ubuntu bug reports as a query to the Bug Duplicate Recommender, retrieve the list of Debian bug reports suggested as duplicates, then compare these results with an oracle of known cross-distribution duplicates. The outcomes of this comparison are the precision and recall of the approach, given by the following formulas:

$$\text{Precision} = \frac{\text{\# Correctly identified duplicates}}{\text{\# Recommended duplicates}} \tag{2}$$

$$\text{Recall} = \frac{\text{\# Correctly identified duplicates}}{\text{\# All duplicates known by the oracle}} \tag{3}$$

Our technique to evaluate precision/recall in the cross-distribution context is different from the one used in prior works on within-project duplicates detection [18], [17], [14]. The latter could train and evaluate models based on reports that have been tagged as duplicate of one unique master bug report in the same repository. Hence, their evaluation could measure how well a model is able to predict the master bug report of a bug report. In our case, Ubuntu reports can track multiple Debian bug reports, i.e., there is no concept of a Debian master bug report for a Ubuntu report. For this reason, we followed a different evaluation approach.

We will calculate precision and recall on two different oracles. The first oracle consists of a random sample of 335 Ubuntu bug reports from the oracle of recorded cross-distribution duplicates used in Section III. 335 corresponds to the sample size necessary for a population of 2,484 Ubuntu bug reports tracking a Debian bug to have a confidence level of 95% and confidence interval of 5% [6].

Since these 335 reports only contain duplicates marked manually in the Ubuntu repository by humans, some duplicates might be missing, which could artificially reduce the precision and recall of the detection approach. Our second oracle consists of the same bug reports as the first one, but this time we manually went through the top 15 recommended duplicates by the detection approach to mark any reports that indeed are duplicates but were not recorded as such in the repository. We only look at the top 15 ranking to stay consistent with the maximal number of duplicates that our approach is allowed to identify.

Table IV provides an example of a bug report list that we get using our approach, where we submit a query to retrieve duplicates for bug report #u1. Usually, the triager should only consider reports above or equal to a fixed threshold (here 75%) that is represented by the triple line in the table, and at most 15 bug reports. Let's assume that the Ubuntu community has already identified the duplicates #d1, #d2, and #d3, we then define a new manual oracle by analyzing all the top 15s bug reports identified by our approach, where we manually identify new duplicates #d4 and #d6 that were not recorded in the bug repository and hence were unknown by the community.

For the first oracle, our approach would retrieve all duplicates with a perfect recall of 100%, however we have a lower precision of 60%, since our approach identified five bug reports to be duplicates when only three reports have been tagged by developers as duplicates (3/5). The evaluation with the manual oracle has different results, since our approach has a lower recall of 80% because the bug #d6 (highlighted in red) is below the threshold (not recommended), however we get a better precision of 80% because #d4 is now recognized as a real duplicate.

We make sure that there are no bug reports in the evaluation test set that are present in the training set. For both oracles, we use the detection approach for a range of logistic regression thresholds from 0.60 to 0.99 in steps of 0.01. Then, we plot the average precision/recall for each oracle across the 335 sampled bug reports, in order to pick a threshold with the best trade-off between precision and recall.

Results:

We manually found 16% (57) new cross-distribution duplicates over the already reported ones using our approach: For 35 submitted reports (10% of the bugs with already reported duplicates), we manually found 57 new

Table IV: Top 15 ranking for #u1 bug query. The last column ("sim.") corresponds to the similarity level (ranging from 0 to 1), and the m/a tags identify duplicates in the **m**anual oracle and the **a**lready reported duplicates oracle

bug id	desc	misc	title	date	pack.	vers.	...	sim.
(m/a) d1	L4	L3	L3	L1	L1	L1	...	0.95
(m/a) d2	L4	L2	L4	L3	L1	L1	...	0.9
(m/a) d3	L3	L1	L3	L1	L1	L1	...	0.85
(m) d4	L3	L1	L3	L1	L1	L1	...	0.8
d5	L3	L1	L3	L1	L1	L1	...	0.8
(m) d6	L3	L1	L3	L1	L1	L1	...	0.75
d7	L3	L1	L3	L1	L1	L1	...	0.75
...	...	...	...	...	...	...	...	...
d15	L4	L2	L3	L2	L1	L1	...	0.55

Debian duplicates over the 347 that were already reported. This shows that the recorded cross-distribution duplicates are not complete, either because people were not aware of them or forgot to mark them. In other words, there is room for tools to automatically detect such duplicates.

The attributes with the largest weight in the similarity model (in descending order) are: date, title, misc, and description. The logistic Similarity Model gives most of the weight to: date="L1", title="L5", misc="L5", and description="L5". This means that recently reported bugs that are very similar in title, miscellaneous fields and descriptions are favoured over other reports in the list of recommended duplicates. On the other hand, the weights of the package, architecture, and severity attributes are marginal. Especially the low importance of package name is surprising, since one would expect the package name to help narrow down the set of reports to compare a new bug report to (as the reporter likely knows which application shows the error) to and hence lead to more precise recommendations.

At the minimum threshold, we get maximum recall of 58/60% and a minimum precision of 4/5% for the two oracles, while at the maximum threshold, we get a minimum recall of 3% and a maximum precision of 99% for both oracles: Figure 5 shows the evaluation of precision and recall for the original and manual oracles. As expected, the precision increases with higher thresholds, while the recall drops (less duplicates are found, since it is harder to have a value higher than the threshold).

Overall, the results for both oracles have only small differences. At the intersection of precision/recall curves (threshold of 88%), we get a precision and recall of 43% for both oracles. This point of balance represented the best pick if one has no preferences between precision and recall (i.e., false alarms and completeness). For RQ2, we prefer more precise results than recall, hence we opt for a threshold of 97%, which yields a higher precision of 91% compared to a lower recall of 14%.

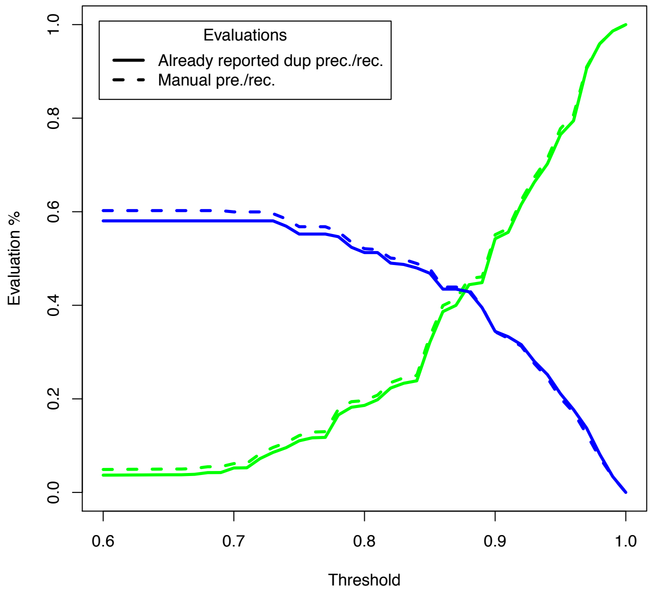


Figure 5: Precision and recall of our approach across multiple thresholds. The green line represents precision and the blue line represents recall.

Our approach detects duplicates across distributions with a maximum recall of 60% using a threshold of 60% (with a precision of 5%), while we get an optimal precision and recall of 43% when using a threshold of 88%.

B. RQ2: How much time is lost by users and developers during the fixing process?

Motivation: to estimate how much time is actually lost by users and developers because they are not aware of cross-project duplicates, we use the detection approach on Ubuntu bug reports to detect new Debian duplicates. We estimate how much time is actually lost by the users in Ubuntu when waiting for a fix that is already available upstream in Debian. We also estimate how much collaboration time is lost when developers are not aware that a bug has been reported in both distributions. In such cases, it would be helpful to merge information from the different bug reports to get new valuable information to fix a bug [5]. Note that we are only able to "estimate" the time loss, by providing an upper limit, since developers have other occupations than working on the bug reports considered in this study, and the data in the repositories might not be complete.

Approach: For RQ2, because we want to analyse the underlying cost of not using our approach, we use another oracle made of bug reports without reported Debian duplicates to evaluate user time lost and collaboration time lost. We randomly pick a sample of 14,500 Ubuntu bug reports (in which 1,844 bugs have within-project duplicates) from our data set of 237,787 Ubuntu bug reports that do not have tracked Debian bugs yet, then we use for each of them the Bug Duplicate Recommender with a threshold of 97% (cf. RQ1) to get a list of suggested Debian bug reports. We are limited to 14,500 bug reports because the use of our detection tool has some overhead (see the cost of using our approach in section V).

Figure 6 shows an example of **collaboration time lost** for developers of both distributions. Let's say someone in Ubuntu reports the bug #u1 on 2011-05-02 and is not aware of the 2 related Debian bug reports that were reported before. At this point, developers start losing collaboration time because the bug #u1's active period overlaps with the one of bugs #d1 and #d2. As shown in this example, we measure collaboration time lost by calculating the time period where developers on both distributions are working on a fix in parallel without knowing each other. To do so, we calculate the maximal number of days where the newly found duplicates' active period overlap with the original report, in this example there are a total of 49 collaborative days lost. The active period is delimited by a bug report's creation date and its closure date. Of course, the obtained collaboration time loss is a theoretical amount of time, since developers do not work full-time on one specific bug [8]. Hence, it should rather be interpreted as a measure of missed opportunities. Also, to simplify things, we do not take in consideration the time where a bug can be fixed and then reopened.

Figure 6 also shows an example of the **time lost by users**, where the Debian bug reports #d1 and #d2 are duplicates of the Ubuntu bug report #u1, and the triager is not aware of these two Debian duplicates. In that case, the Ubuntu users would have lost 14 days on waiting for a fix already available in Debian for bug report #d2. We do not consider the report #d1 because its fix date is later than the one of #d2, we always consider the earliest fix date.

Hence, we measure the time lost by the users waiting for a fix by calculating the time difference in days between the Ubuntu bug closure date and the earliest Debian duplicate fix date. In the case where the Ubuntu bug report is fixed before any of the Debian duplicates, we consider the time loss to be 0 because the users do not wait extra time due to being unaware of cross-distribution duplicates. Similar to RQ1, this paper does not consider the case where, on the opposite side, the Debian users lose time while waiting for a fix already done in Ubuntu. We perform our analysis of user time loss on our sample of 14,500 Ubuntu bug reports that do not have Debian trackers, since for reports with tracker people are already aware of the duplicate, hence there is no user time loss.

To put the amount of time loss (both for collaboration and users) into perspective, we compare it to the typical time used for fixing Ubuntu reports without cross-project duplicate, calculated as the time difference between the creation date and closure date of all 28,840 bug reports that are not linked in any way to a tracker (including masters and their duplicates). Although this time difference does not represent the real work load done by the developers, it gives an idea on how significant the user and collaboration time loss are in the bug fixing process.

Results:

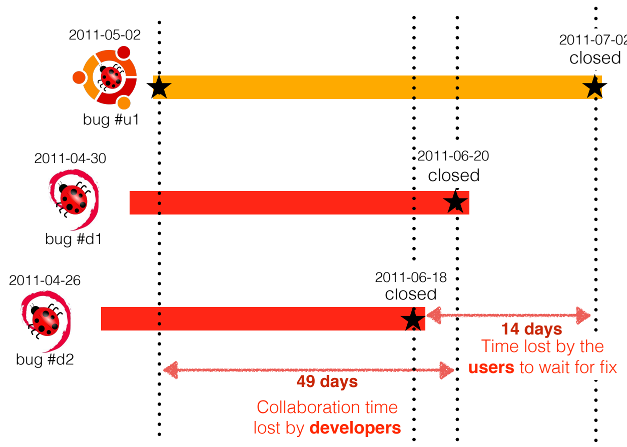


Figure 6: Time lost by Ubuntu users waiting for a fix.

Our approach retrieved new Debian duplicates for 4% (821) of the submitted Ubuntu bug reports: Out of 14,500 bug reports, our approach with a threshold of 97% recommended 821 Debian duplicates for 591 Ubuntu bug reports. Note that of these duplicates, 91±5% should be expected to be correct (cf. precision in RQ1), while the recall is expected to be low. In other words, more duplicates exist in the data set than identified with a 97% threshold. Here, we prefer to identify less, but more correct recommendations.

Figure 7 shows the data in the form of beanplots. Each bean (either left or right) shows a shape depicting the distribution of a group's data across a range of values and shows a line delimiting the data's median.

It takes a median of 29 days to fix a bug in Ubuntu: As depicted by the both right hand side beanplots on Figure 7, the time to fix a bug in Ubuntu is almost fairly distributed within a range from a few days to even a year and more, albeit more bugs tend to be fixed within a few days (see the small peek at the bottom of the bean).

Users waiting for a fix lost a median of 38 days: As depicted by the left hand side of the left beanplot on Figure 7, a significant part of the time lost by developers is distributed around 30 days and 700 days. A user in Ubuntu can either wait a few days or more than a year for an already existing fix in Debian, which could depend on how long is the bug fixing process in Ubuntu is compared to Debian (without any collaboration across distributions). The median of user time loss is 9 days higher than the one of bug fixing time, which means that, whereas Ubuntu bugs typically are fixed within a month, bugs that are not noticed to be cross-distribution take more than a month extra time on top of the typical time to fix a bug in Ubuntu!

Developers lost a median collaboration period of 47 days: As depicted by the left hand side of the right beanplot on Figure 7, a significant part of the time lost by developers is distributed around 30 days and 500 days, which is similar to

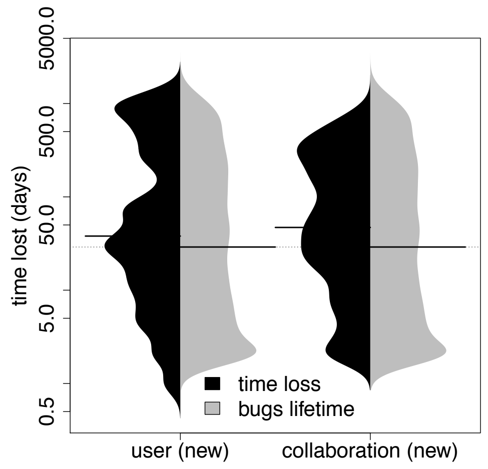


Figure 7: Collaboration and user time lost. The left and right bean plots depict the time loss that we uncovered by detecting new duplicates. Both cases of time loss are compared to the typical bug fixing lifetime of all Ubuntu bug reports in the repository having no known duplicates.

what we observed for user loss time. The median collaboration time loss is higher than the one to fix a bug, meaning that being able to spot and exploit collaboration opportunities for cross-distribution bugs could substantially reduce the time to fix those bugs. Of course, this would also automatically reduce the time for users waiting for a bug fix.

While fixing a bug takes a median time of 29 days, developers lose a median of 47 days of potential collaboration and users lose 38 days on waiting for fixes already made in the other distribution when considering new duplicates detected by our approach.

V. Discussion

A. Real Cases where Time Is Lost Across Distributions

We analyzed results of RQ2 to identify cases where time is lost. Ubuntu bug #1113370 represents a case where time is lost by waiting users and for collaboration. On February 2nd 2013, the same PostgreSQL bug is reported on Debian (#699604) and Ubuntu. Even thought the author was the same for both reports, he did not start a tracker for the upstream Debian bug nor inform other people about the existence of the bug across distributions, which means that developers lost time by working independently on a fix. The patch consisted of an update to package "roundcude". Both bug reports were closed the same year, i.e., on March 16th for Debian and on April 1st for Ubuntu. A total of 16 days were lost by waiting Ubuntu users and 42 days were lost on collaboration, albeit we do not know if the author of both bug reports was working discretely with people from both distributions. We only have access to information available from the bug repository in this case.

There are cases where Ubuntu users could lose over half a year waiting because maintainers in Ubuntu were fixing a given bug later than maintainers in Debian. For example, maintainers involved in Ubuntu bug #723940 for "xlog" take 213 more days than the maintainers in Debian (bug #614847) to update the package. In such a case, a better collaboration across distributions might have reduced the package updating process for Ubuntu.

B. Collaboration Benefits from the Detection of Bug Duplicates

Our results show that there is an average of 47 days that is not used by the developers from Ubuntu and Debian to collaborate. Although this is of course an approximation, and collaboration does not expedite the fixing of every bug (some Debian bug fixes could require additional validation by Ubuntu developers), we can argue that the collaboration at least could consist of obtaining more information about a bug (e.g., more stack traces), which could help fix a bug quicker [4].

C. Duplicates that our Approach can Find

To evaluate our approach, we submitted queries that consist of Ubuntu bug reports and only identified the Debian bug duplicates, however we can retrieve other kinds of duplicates using our approach. For example, it is possible to rather query a Debian bug report and get the Ubuntu duplicates, and it is also possible to retrieve a mix of duplicates from both repositories. Our approach is independent from Ubuntu/Debian, hence it is also scalable to other distributions and even other kinds of open source projects, as long as we are able to parse the bug reports (i.e., build new parsers, see Figure 3), identify common bug report fields, and push those to the common database.

D. Cost of Using Our Approach

To convince bug repository triagers to use our approach, it has to provide results in a reasonable delay. Our Bug Duplicate Recommender takes on average 130 seconds to provide recommendations for a new bug report, where it has to deal with a database of 375,931 bug reports in our case study, which is less than the average 20 minutes spent by triagers to find duplicates [7]. Hence, the approach's run-time is marginal considering the fact that our approach can reduce the triaging time, lost time, and lost collaboration time. Even if the Bug Duplicate Recommender takes a reasonable time to recommend duplicates for a single bug report, it requires a significant amount of time (over 500 hours) to generate duplication recommendations for thousands of bug reports in our case study (see RQ2 in IV).

E. Finding the Optimal Threshold

By looking at Figure 5 and Figure 6 we can see that there is no perfect threshold for detecting cross-distribution duplicates, hence we need to make a trade-off between precision, recall, and lost time by users. The choice of the threshold depends on the triager's preference, since one could prefer a shorter and more relevant list of duplicates (higher precision, cf. RQ2), or the full list of suggested bug reports to be sure that most of the duplicates are detected. The latter case would make the users lose less time while waiting for a bug fix, but it requires more effort from the triager. In other words, the triager has to trade-off between how much time he/she is willing to spend and how much time he/she can make the end users wait. A compromise could be to use a threshold of 88%, where the recall and precision are at the same value of 43%.

VI. Threats to the validity

A. External Threats

We only consider the Ubuntu and Debian Linux cross-distributions in our study, which may not represent a generic case for the hundreds of other cross-distributions relations in the Linux ecosystem. Our study concerns two of the most popular distributions and considers over 350,000 bug reports. Debian is involved in multiple major Linux distributions while Ubuntu is one of the most popular derived distribution's of Debian (similar to Mint). The duplicate detection approach should be applied to those other distributions, as well as to non-distribution projects.

B. Construction Threats

We manually build an oracle to evaluate the precision/accuracy of our duplicate detection technique. The oracle in RQ1 is relatively small (335 reports), since we had to manually analyze all 15 recommendations of each report to find duplicates that are not recorded as such in the Ubuntu repository. Furthermore, we are not expert triagers of Debian and Ubuntu bug repositories, hence we could make errors in the manual classification of the duplicates.

We did not use the same technique as prior works to evaluate precision and recall of the recommendation approach, since the concept of master bug report used by prior work does not exist for a Ubuntu report (see approach of RQ1).

VII. Conclusion and Future Work

In this study, we adapt an existing approach for recommending within-project duplicate bugs to recommend bug duplications across distributions (Ubuntu and Debian), which can achieve a recall up to 60%, comparable to the performance of existing duplicate detection inside one project. Our approach offers a threshold adjustment that provides flexibility to the bug repository triager to decide between detecting more duplicates or having more precise results. A threshold of 88% represents a balanced choice between precision and recall of 43%. We found that the application of our approach could help triagers to find relevant duplicates, and reduce by a median of 38 days the time lost by the Ubuntu users waiting for a fix already available in Debian. We also found that developers from both distributions lost a median of 47 days of potential collaboration, where our approach has the potential to help them save this precious time. Based on these promising results, we suggest triagers working in cross-distribution software projects to apply the approach to reduce the time that is actually lost by users and developers.

References

[1] Bram Adams, Ryan Kavanagh, Ahmed E. Hassan, and Daniel M. German. An empirical study of integration activities in distributions of open source software. *Empirical Software Engineering*, 2015. to appear.

[2] Anahita Alipour, Abram Hindle, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, pages 183–192. IEEE Press, 2013.

[3] John Anvik, Lyndon Hiew, and Gail C Murphy. Coping with an open bug repository. In *Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, pages 35–39. ACM, 2005.

[4] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. What makes a good bug report? volume 19, pages 1–30, 2008.

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

[6] Sarah Boslaugh and Dr. Paul A. Watters. *Statistics in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, first edition, 2008.

[7] Yguarata Cerqueira Cavalcanti, Eduardo Santana de Almeida, Carlos Eduardo Albuquerque da Cunha, Daniel Lucredio, and Silvio Romero de Lemos Meira. An initial study on the bug report duplication problem. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 264–267. IEEE, 2010.

[8] Kevin Crowston and Barbara Scozzi. Bug fixing practices within free/libre open source software development teams. 2008.

[9] Ronen Feldman and James Sanger. *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press, 2007.

[10] David A Freedman. *Statistical models: theory and practice*. cambridge university press, 2009.

[11] Hongying Gu, Long Zhao, and Chang Shu. Analysis of duplicate issue reports for issue tracking system. In *Data Mining and Intelligent Information Technology Applications (ICMiA), 2011 3rd International Conference on*, pages 86–91. IEEE, 2011.

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

[13] Nicholas Jalbert and Westley Weimer. Automated duplicate detection for bug tracking systems. In *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pages 52–61. IEEE, 2008.

[14] Tomi Prifti, Sean Banerjee, and Bojan Cukic. Detecting bug duplicate reports through local references. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, page 8. ACM, 2011.

[15] Anand Rajaraman and Jeffrey David Ullman. *Mining of massive datasets*. Cambridge University Press, 2011.

[16] Baishakhi Ray and Miryung Kim. A case study of cross-system porting in forked projects. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 53. ACM, 2012.

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

[18] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 253–262. IEEE Computer Society, 2011.

[19] 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. ACM, 2010.

[20] Cheng-Zen Yang, Hung-Hsueh Du, Sin-Sian Wu, and Xiang Chen. Duplication detection for software bug reports based on bm25 term weighting. In *Technologies and Applications of Artificial Intelligence (TAAI), 2012 Conference on*, pages 33–38. IEEE, 2012.