Information and Software Technology 000 (2016) 1–13

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Automated triaging of very large bug repositories

Sean Banerjee[a,*], Zahid Syed[b], Jordan Helmick[c], Mark Culp[d], Kenneth Ryan[d], Bojan Cukic[e]

[a] Clarkson University, Potsdam, NY, USA
[b] University of Michigan - Flint, Flint, MI, USA
[c] MedExpress, Morgantown, WV, USA
[d] West Virgina University, Morgantown, WV, USA
[e] University of North Carolina - Charlotte, Charlotte, NC, USA

A R T I C L E I N F O

Article history:
Received 28 February 2016
Revised 18 September 2016
Accepted 20 September 2016
Available online xxx

Keywords:
Automated triaging
Bug tracking
Big data analytics
Software problem repositories

A B S T R A C T

Context: Bug tracking systems play an important role in software maintenance. They allow both developers and users to submit problem reports on observed failures. However, by allowing anyone to submit problem reports, it is likely that more than one reporter will report on the same issue. Research in open source repositories has focused on two broad areas: determining the original report associated with each known duplicate, and assigning a developer to fix a particular problem.

Objective: Limited research has been done in developing a fully automated triager, one that can first ascertain if a problem report is original or duplicate, and then provide a list of 20 potential matches for a duplicate report. We address this limitation by developing an automated triaging system that can be used to assist human triagers in bug tracking systems.

Method: Our automated triaging system automatically assigns a label of original or duplicate to each incoming problem report, and provides a list of 20 suggestions for reports classified as duplicate. The system uses 24 document similarity measures and associated summary statistics, along with a suite of document property and user metrics. We perform our research on a lifetime of problem reports from the Eclipse, Firefox and Open Office repositories. Results: Our system can be used as a filtration aide, with high original recall exceeding 95% and low duplicate recall, or as a triaging guide, with balanced recall of approximately 70% for both originals and duplicates. Furthermore, the system reduces the workload on the triager by over 90%.

Conclusions: Our work represents the first full scale effort at automatically triaging problem reports in open source repositories. By utilizing multiple similarity measures, we reduce the potential of false matches caused by the diversity of human language.

1. Introduction

Tracking bug reports is one of the best practices in the maintenance of computer software. Bug reports are submitted by both developers and users of the system. Allowing users to submit bug reports provides developers with continuous feedback about the operational behavior of their product. However, submitted reports vary in quality, and it is often difficult to understand what is being reported. Bug reports are initially classified through a process called triaging. An analyst, commonly called a triager, must determine whether a newly submitted report describes a previously unreported issue or a previously reported issue. We refer to the former as an original report, while the latter is called a duplicate report.

Manual triaging of bug reports is both challenging and time consuming. The very nature of the English language entails that two people can use vastly different language to describe the same issue. For example, bug reports #2478 and #2644 in Mozilla describe the same issue, yet share little similarity in language. An inexperienced triager or a machine could easily mistake the reports as being distinct from each other. Similarly, an individual can use nearly identical language to describe two completely different issues. For example, reports #123376 and #123734 in Eclipse describe different issues, and yet share nearly every word in common. An inexperienced triager or a machine could easily be fooled into thinking the reports describe the same issue, when in fact the subtle difference is the build date. Anvik et al. [10] reported

[*] Corresponding author.
E-mail addresses: sbanerje@clarkson.edu (S. Banerjee), zahsyed@umflint.edu (Z. Syed), jordan.helmick@medexpress.com (J. Helmick), mvculp@mail.wvu.edu (M. Culp), kjryan@mail.wvu.edu (K. Ryan), bcukic@uncc.edu (B. Cukic).

http://dx.doi.org/10.1016/j.infsof.2016.09.006
0950-5849/© 2016 Published by Elsevier B.V.

in 2005 that Mozilla employees triage over 300 bug reports per day. Our own experience indicates that 10 years later, Mozilla still receives over 350 reports per day. Other repositories, such as RedHat receives over 330 reports per day, Novell receives over 150 reports per day and Eclipse receives over 80 reports per day. Today, repositories from RedHat and Mozilla contain over 1.2 million and 1.1 million bug reports respectively, the Novell repository exceeds 950,000 reports and the Eclipse repository exceeds 480,000 reports. Thus, it should not be difficult to appreciate the human effort required in triaging bug reports for large open source projects and understand the need for automation.

While many papers claim contributions to "duplicate bug report detection", we note that the focus is on finding a "match" that already exists in the repository for a report that has already been assumed to be a duplicate *a priori*. In reality, a triager or an automated system will have no knowledge of whether an incoming report is original or duplicate. The resulting tools suggest the link between this duplicate bug report and the report that originally described the problem. Typical research results offer a list of 20 best matches for a known duplicate report. Having the correct match in the list of 20 candidates is considered a successful outcome, as the triage team now analyzes a short list of potentially matching reports rather than the contents of the entire repository. Related research therefore answers the question: "Given that the report is a duplicate, what are its most likely matches?" In practice, *a priori* knowledge that the new report is, indeed, a duplicate is a strong assumption. In this work we address this assumption by building a tool that determines whether the incoming report is an original or a duplicate. In our work, the status of all incoming bug reports is initially set to unknown. The reports are subsequently classified as original or duplicate based on the analysis of their features and comparison with the previously submitted reports. If determined to likely be a duplicate, a set of 20 potential matches are generated and provided to the user.

It can be argued that the "duplicate bug report detection" can be formulated as detecting original vs. duplicate reports. A top 20 list of best matches can be generated for every bug report in the repository, and presented to the triager. A triager would then manually inspect the bug report and the associated 20 matches to decide if the report is original or duplicate. This approach would increase the workload on the triager by several orders, as the triager now has to read not just the original report, but through 20 additional reports to determine if the suggestions are by random chance or true duplication. The correct approach to automated triaging is to provide the triager with reports that are "hard" to classify automatically. Reports that can be classified as original should be sent to the developers to fix. Our approach follows this paradigm, we provide triagers with reports that our system classifies as duplicate and ask for a manual inspection, thereby reducing their workload by over 90%.

The techniques applied in related work has also solely focused on similarity metrics combined with report metrics, while ignoring the human user. In [13], we demonstrated that the experience of the user greatly affects the type of report that they submit. In our approach the user demographics are summarized and used as a feature in the classification process. Moreover, the techniques proposed in prior research for finding duplicate bug reports have been evaluated on relatively small subsets of bug repositories as shown in Table 1. This can be misleading because the performance of such techniques will likely degrade as the number of reports in the repository increases [26,28,29]. Therefore, the long term accuracy of automated bug report management in large scale open source projects will remain unclear until the empirical analysis is based on large and diverse samples of bug reports. We address this by utilizing a lifetime of reports from 3 diverse projects - Eclipse, Firefox and Open Office. It can be argued that time and cost precludes researchers from analyzing complete repositories. As we demonstrated in [11], distributed techniques can be applied that relies on utilizing unused computing resources found in most academic institutions.

Table 1
Scalability challenge in duplicate problem report classification.

Dataset	Range	Recall	Reports	% of Dataset
Eclipse	Jan 2008 – Dec 2008	68% [29]	2013	6%
Eclipse	Start – Dec 2009	46% [30]	2270	6%
Eclipse	Jan 2008 – Dec 2008	78% [28]	3080	8%
Eclipse	Start – Dec 2007	71% [28]	27,495	100%
Firefox	Apr 2004 – Jun 2004	93% [31]	77	0.44%
Firefox	Apr 2002 – Jul 2007	53% [29]	3307	12%
Firefox	Apr 2002 – Jul 2007	70% [29]	3307	12%
Firefox	Start – Jun 2010	53% [27]	19,480	50%
Firefox	Start – Mar 2012	68% [12]	25,045	50%
Firefox	Start – Sept 2005	50% [21]	8070	85%
Mozilla	Jan 2010 – Dec 2010	68% [28]	6925	5%
Mozilla	Feb 2005 – Oct 2005	51% [23]	8225	7%

It can also be argued that Bugzilla, or other issue tracking systems, provide a rudimentary framework for assessing whether a report is novel or not [1]. However, the Bugzilla search tool uses a Boolean search on the title of the report, which consists of less than 10 words on average. Thus, unless the reporter is using very similar terminology as an existing report, an effective match will not be made. As a result, the submitter may inadvertently submit a duplicate report which has to be read by a human triager to ascertain its true status. An automated tool which can flag reports as original or duplicate can greatly assist the triager.

Our prior research in the domain has demonstrated the effect of time on classifying duplicate problem reports [27], and in [12] we demonstrated that subsequence matching techniques can address the loss of context challenges faced in frequency based approaches. Our work in [14] shows how a multi-label classification approach can alleviate the challenges faced in frequency and sequence based approaches. We also introduced a set of features that were used to determine the most effect similarity measure for each problem report. The true nature of problem repositories was explored in [13], where we showed the quality of reports submitted in relation to the experience of the submitter. Finally, in [11] we presented a detailed cost analysis associated with mining large scale problem repositories.

In this paper, we present a system that can classify each incoming bug report as either an original or a duplicate by extracting a novel set of features associated with the reporter, the report and the extent of similarity to prior reports. Reports that are classified as duplicate are then tagged with a list of 20 potential matches. These features are designed to provide multiple viewpoints into the content of reports, and offer an extensive yet diverse set of features allowing the automatic system to differentiate between original and duplicate reports. We further propose several learning approaches to classification, each geared towards a particular practical consideration.

The major contributions of this paper include:

1. A detailed description of a novel automated framework for classifying bug reports as originals or duplicates.
2. A novel set of features associated with the similarity scores, report and reporter that can be used to automatically classify bug reports.
3. A demonstration, through empirical study, that this approach reaches high accuracy when applied to the entire Eclipse, Firefox and Open Office repositories.