# Merging Duplicate Bug Reports by Sentence Clustering


## Abstract

Duplicate bug reports are often unfavorable because they tend to take many man hours for being identified as duplicates, marked so and eventually discarded. In this time, no progress occurs on the program in question, and is justifiably an overhead which should be minimized. Considerable research has been carried out to alleviate this problem. Many methods have been proposed for bug report categorization and duplicate bug report detection. However, it is often the case that a duplicate bug report can provide some additional information about a problem which could help in faster resolution of the bug. We propose that duplicate bug reports be merged when possible instead of being discarded, so that maximum information is captured. We propose a clustering-based algorithm to group together similar sentences and create a union of bug reports considered duplicates of each other.


## 1 Introduction

Open-source projects make the source programs that constitute the project available to the general public with the freedom to modify, recompile and redistribute them. Like any other project, open-source projects will have defects in them, which require repair. These defects are called *bugs*, and reports of them are called *bug reports*. Bug reports also very often have requests for new features, instead of reporting any defect. In such projects, discovering these defects and reporting them is not limited to any specific testing team. Anybody interested in using the project, when they discover bugs, can report them.

Anyone who reports bugs from an open-source project can be considered as belonging to the testing team of the project. This implies that bug reports will be large in number. To manage such large numbers of bug reports, open-source projects typically set up *open bug repositories* and *bug tracking systems*. *Bugzilla*[1] is an example of an open bug tracking system. People who use open-source projects and wish to report bugs in them may do so using such a repository and a tracking system.

While the advantages of such a model for reporting bugs are obvious, it also has problems, which brings us to what is called *bug triaging*. This is the process of deciding what to do with an incoming bug report [12]. Part of this process is assigning the bug report to an appropriate developer. While this process is quite tedious in itself, it is exacerbated by *duplicate* bug reports—the elimination of which is another part of triaging. The project does not undergo any improvement in the process of eliminating duplicate reports and it is therefore an overhead which should be minimized to the extent possible.

Duplication of bug reports happens because there are several people from different parts of the world using the same open-source project and often, many of these people discover the same bug and, consequently, report them. For example, from the Mozilla Firefox bug repository [13]:

1. `Bug-239223: (Ghostproc) [Meta] firefox.exe doesn't always exit after closing all windows; session-specific data retained`

2. `Bug-260331: After closing Firefox, the process is still running. Cannot reopen Firefox after that, unless the previous process is killed manually`

[1] http://www.bugzilla.org/

The defect spoken of is the same, but two bug reports have been created for it. One of these bug reports should be discarded as it is redundant to have both of them in the repository. Next, consider these two bug reports, again from the Mozilla Firefox bug repository [13]:

1. `Bug-244372: ‘‘Document contains no data’’ message on continuation page of NY Times article`

2. `Bug-219232: random ‘‘The Document contains no data.’’ Alerts`

These bug reports, although they speak of the document-containing-no-data problem, are not exactly duplicates of each other. One bug report speaks of a specific case (NY Times) while the other report speaks about the problem occuring randomly. In such situations, the triager would probably prefer to *merge* (take a union of) these two reports into a single one, which the concerned developer can then repair.

We want to attempt to address this problem of merging or unifying duplicate or similar bug reports so that no information from any of the reports is lost and at the same time no information is duplicated.

Table 1: Fields in a Bug Report

| **Field** | **Description** |
|---|---|
| Summary | Concise description of the problem |
| Project Name | Which project the report concerns |
| Number | A unique identification number for this report |
| Tester | The name of the tester |
| Date | The date of submission |
| Severity | How important it is that this is fixed |
| Status | Indicates the general health of a bug |
| Type | Whether it is a bug report or a change request |
| Test Object | What software the test was performed on |
| Version | What version of the software was used |
| OS | Which operating system the software was run on |
| Test Specification | Reference to the test case used |
| Description | A description as to what went wrong |
| Appendices | Attachments such as test logs *etc.* |
| Remarks | Comments |

# 2 Background

## 2.1 About a Bug Report

The general anatomy of a bug report is as shown in Table 1 [10].

The *Summary* field provides a very brief description of the problem being reported. It can be thought of as the title of a bug report. The *Project Name* field is usually the name of the product in which the bug being reported was found. The *Severity* field is indicative of the priority of the bug: a higher-severity bug must be worked on and resolved before a lower-severity bug is resolved. Often deadlines are defined for bugs of high severity. The *Status* field indicates what the present status of the bug is. This is described in the next few lines. The *Description* field is where the bug reporter supplies information about when the problem occurred, what was being done at the time the bug occurred and how to reproduce the problem. This field is manually typed by the reporter in natural language and is a key field in resolving the bug. Very often, software projects have guidelines as to what information should be provided in this field. The *Appendices* field consists of any necessary attachments such as any input files necessary in reproducing the bug, any malformed/anomalous output files *etc.*

The *Status* field takes one of several values in the “life cycle” of a bug report. They are as follows [1]. When a new bug report is filed, the status is set to NEW or UNCONFIRMED. After triaging, i.e. when the bug report has been assigned to a developer or if a developer accepts responsibility for it, the status is changed to ASSIGNED by the triager. When a bug report is closed, its status is set to RESOLVED. It

may be further marked as VERIFIED if it is being verified by a quality assurance group or CLOSED if the bug report is closed for good.

A bug report may be resolved in many ways. Bugzilla provides the following *resolution statuses* for a bug. If a source-program-level change was made to address the bug, the status is set to FIXED. If the bug report is found to be a duplicate of an existing report, the status is set to DUPLICATE. In this case, the bug report is makrked with the number of the duplicating bug. If the developer is unable to reproduce the problem, the status is set to WORKSFORME. The resolution status is set to WONTFIX if the bug report describes a problem that will never be fixed, INVALID if the problem described is not a bug and MOVED if the problem described should belong in another bug database.

## 2.2 Related Work

Researchers have not defined exactly what 'duplicate' means, in that there is no single governing rule that classifies two bug reports as duplicates or otherwise. Different methods have been proposed and two reports being duplicates or not depends on the method.

Considerable research has been carried out on categorization of bug reports for bug assignment. Anvik *et al.* [2] have proposed a semiautomatic bug assignment method which uses a supervised machine-learning algorithm. They report having obtained 64% and 57% precision in the Mozilla Firefox and Eclipse repositories, respectively, of to who to assign a bug. Čubranić and Murphy [12] have proposed a naïve Bayesian classifier for semiautomatic bug triaging, and they report that it can predict 30% of correct assignments of bug reports to developers. This is also a machine-learning-based technique. Di Lucca *et al.* [8] have also proposed a semiautomatic bug assignment approach which is based on information retrieval and machine-learning. They have reported that, depending on the classification model used, 71% to 84% of the assignments made were correct. All of these approaches use only natural language information. Podgurski *et al.* [9] have suggested a method of classifying bug reports based on supervised and unsupervised pattern classification and multivariate visualization. They report that their experiments on three programs—gcc, javac and Jikes—showed that their approach is effective. However, their approach is targeted at prioritizing bug reports by severity and frequency, and not for bug assignment. Francis *et al.* [4] have proposed two tree-based methods to refine initial classification of failure reports. Their approach is based on clustering. This work is built upon [9]. These approaches [9, 4] rely on execution information alone.

In addition to bug report categorization, a lot of research has also been carried out on duplicate bug report detection. Runeson *et al.* [10] have proposed a method to detect duplicate bug reports based on natural language information; their approach is based on information retrieval. They carried out their experiments on the bug repository at Sony Ericsson Mobile Communications and they report that 2/3 of duplicates can possibly be found using natural language processing techniques. Hiew [5] has proposed a method for detection of duplicate bug reports based on natural language information and clustering which is similar to information retrieval. Jalbert and Weimer [6] have proposed a classifier to detect duplicate bug reports as they are created. Their approach is based on surface features, textual similarity metrics and graph clustering algorithms. Their classifier filters out 8% of duplicates and allows at least one report per real defect to reach developers.

Wang *et al.* [13] have described an approach to detect duplicate bug reports by combining both natural language information and execution information (execution traces from bug-revealing runs). From their experiments, they were able to report that their method detects 67%-93% of duplicates from the Mozilla Firefox bug repository, as compared to 43%-72% using NLP techniques alone.

Also, there have been statistical studies of existing bug repositories. Anvik *et al.* [1] provide an initial characterization of the Mozilla Firefox and Eclipse bug repositories to understand better the interactions between developers and bug repositories, describe the duplicate bug and bug triage problems and also discuss how they apply machine-learning to help in the automation of these processes. Sandusky *et al.* [11] performed an analysis to identify bug report networks

(groups of bug reports because of duplication, dependency or reference relationships) to help manage problems better. Ko *et al.* [7] performed an analysis on how people describe software problems. They studied several bug report titles and found that they were sufficiently regular to be parsed correctly 89% of the time, paving way for many automated analyses.

### 2.3 An Open Problem

As noted above, there has been considerable amount of research targeted at detecting duplicate bug reports and categorizing bug reports for bug assignment and prioritization. However, to the best of our knowledge, there is little work done in the direction of merging duplicate bug reports. Bettenburg *et al.* [3] have mentioned that it is a good idea to merge duplicate bug reports, but have not presented any strategy for it.

Many times, additional information about a bug can be obtained from duplicate bug reports that aid developers in fixing problems faster [3]. Alan Page, Director of Test Excellence at Microsoft, mentions on his blog the following points about why worrying about duplicate bugs is bad[2]:

1. If there is any sort of negative consequence for a tester entering a duplicate bug, that tester will err on the side of not entering a bug at all if they are worried about entering a duplicate bug.

2. Bug triagers know the system well and can quickly identify if a bug is a duplicate or not, in much lesser time than a tester or a user would take to look through the many existing bug reports before they file one.

3. Often, the information in one bug report doesn't provide enough information to diagnose the problem. Another report on the same issue may lead the developer to the root cause. Most bug database systems have way to mark bugs "related" (or "duplicate") and retain a link between the bugs.

[2]http://blogs.msdn.com/alanpa/archive/2007/08/01/duplicate-bugs.aspx

Oftentimes, detailed bug reports filed by advanced users are discarded as they are found to be duplicates of already-filed detail-lacking bug reports filed by novice users. Not only does this cause frustration to the persons filing the detailed bug reports, but also useful information which could help resolve the bugs is lost [3]. Therefore, we believe merging bug reports will be of considerable help to developers in resolving bugs.

## 3 Challenges

We consider the merging of only the *Description* field. The other fields consist mostly of fixed information not amenable to merging. The *Summary* fields from the bug reports can be simply concatenated with an indication from which bug report they come from and separated by a special character, because they are just a few words in length.

The objective of merging duplicate bug reports is to obtain a "union" of the bug reports with similar sentences grouped together. By "similar", we mean sentences that speak of the same thing. For example, sentences from duplicate bug reports that speak of how to reproduce the problem are considered similar, sentences that describe what was being done when the problem occurred are considered similar. The principal challenge we face is finding out how we can compare sentences for similarity.

Consider the following *Description* fields from bug reports filed on Bugzilla for the Bugzilla product itself:

1. `Bug-125888: It would be nice if *** This bug has been marked as a duplicate of xxx *** included the summary of and a link to bug xxx, and similarily if *** Bug yyy has been marked as a duplicate of this bug. *** included the summary of and a link to bug yyy.`

2. `Bug-96787: When a bug is marked a duplicate, you get the following in the email: >>> *** This bug has been marked as a duplicate of XXXXX *** It would be nice if this also provided a`

```
link to the bug, so you could go and
look at it.
```

The first bug report requests a feature in one particular scenario, whereas the second bug report requests for the same feature in two different scenarios. These two bug reports are worthy candidates for merging in, perhaps, the following way:

1. `Bug-nnnnnn: It would be nice if *** This bug has been marked as a duplicate of xxx *** included the summary of and a link to bug xxx ; When a bug is marked a duplicate, you get the following in the email: >>> *** This bug has been marked as a duplicate of XXXXX *** It would be nice if this also provided a link to the bug, so you could go and look at it ; and similarily if *** Bug yyy has been marked as a duplicate of this bug. *** included the summary of and a link to bug yyy.`

The first portion of the bug report and the second bug report are together because they are similar sentences, followed by the second portion of the first bug report. Such intelligent and intuitive grouping of sentences may be very hard to achieve, but it would certainly be of help if it can be done. Having similar information grouped together in merged bug reports will lend a degree of coherence to them.

# 4 Proposed Research

## 4.1 Proposed Approach

In our approach, we make the assumption that we are presented with a set of bug reports that have been classified as duplicates. We propose to do the merging using a clustering-based algorithm. Specifically, given a set of duplicate bug reports, we want to cluster the similar sentences from the *Description* fields in them together in the output merged bug report.

We can develop constraints or rules that can guess the context of a word correctly. For example, we can have rules that can tell if the word *tree* is used in the computer-science-context, perhaps depending on the words surrounding it.

The first stage involves breaking of the sentences in the paragraph present in a bug report's Description field. This will give us a set of sentences. Thus, we would be creating several sets of sentences for the set of bug reports provided as input. We extract the textual information of the Description field of the bug report. This set will be used in the second stage for standard preprocessing.

The second stage of the approach is used in performing the preprocessing of information available in the description of the bug report. This includes stemming and removal of the stop words from the set of sentences obtained in the first stage. We get a set of words which we would classify as keywords in this paper. The keywords generated from all the reports are grouped together by meaning, *i.e.*, we create synonym lists. The synonym lists are augmented by more words (from a thesaurus). While doing so, we address *polysemy*, in that we create separate synonym lists for different connotations of a word. For instance, *tree* or *chip* will have separate synonym lists for each of their meanings. We make use of the aforementioned constraints for the language under consideration while creating the word lists.

The next stage involves correlating the sentences that appear to reason the same information in their descriptions. We create a matrix that has the sentences along its rows and synonym lists along its columns. sentences and words created in the previous stages. If in a sentence, a word from a synonym list appears, then, an entry is made in the intersection of the corresponding row (sentence) and the column (synonym list). In this manner, we construct the entire matrix and mark the values based on comparison of keyword lists in the sentences. This stage provides us with a form of clustering where we are able to group the relevant descriptions based on the number of columns they relate to in the matrix.

The final stage includes outputting the clusters obtained. This final output will be the merged Description field of all the duplicate bug reports. The other fields are simply concatenated with a indicator that tells which bug report each of those fields are from.

### 4.2 Evaluation Plans

We plan to evaluate our approach in a two-step process. First, we plan to add manually duplicate reports to an existing repository and test our approach against it to check the correctness of our approach. Next, we plan to conduct user-based studies to ascertain the effectiveness of our approach.

We seed "fake" duplicate bug reports into an existing bug repository repository to ensure and provide sufficient test data for the evaluation of this approach. We can provide the tool with several pairs of bug reports. Some pairs will consist of duplicate reports. We can then run the pairs against the tool. Then, we will manually verify the results from the evaluation against the predicted results. This will not only give us a clear idea about the correctness of the tool, but we would also be able to verify the extent to which the tool helps in merging of redundant bug reports.

Secondly, we would want to consider user-based studies for evaluating our approach on "real" duplicate bug reports reported in real-word bug repositories. This would involve taking multiple sets of bug reports (assuming that some contain duplicate bug reports), run the tool against the real world data and compare the results. A triager would be able to comment on the effectiveness of our approach. The involvement of the triager in this approach enables us to generalize the level to which the tool works correctly. Once we have generalized and validated on the result sets from the evaluation, we can exclude the triager from verification. This would mean that the tool has been validated to work efficiently in merging redundant reports without margin for error.

## 5 Summary and Foreseen Contributions

An approach for merging duplicate bug reports has been presented. Previous related work is directed at identification of duplicate bug reports and categorizing bug reports. Our approach is novel in that it addresses the problem of merging many duplicate bug reports into a single, coherent bug report. Being able to merge bug reports enables us to capture more information about a given bug than is available in any one bug report. This, we believe, is a key contribution to the resolution of bugs. A detailed bug report will not be discarded if found to be a duplicate of an already-filed not-so-detailed bug report; instead, it will contribute more information that could clearly point out the direction in which to proceed in debugging. In addition to this, merging reports can also reduce the work load of developers—all the information from all the duplicate reports is grouped and available in one place and there will be no need to look in several reports.

## References


[1] J. Anvik, L. Hiew, and G. Murphy. Coping with Open Bug Repositories. In *Proc. of OOPSLA Workshop on Eclipse Technology eXchange (ETX)*, pages 35–39, 2005.

[2] J. Anvik, L. Hiew, and G. Murphy. Who Should Fix This Bug? In *Proc. ICSE*, pages 317–380, 2006.

[3] N.P. Bettenburg, R. Zimmermann, and T.S. Kim. Duplicate Bug Reports Considered Harmful ... Really? In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, pages 337–345, 2008.

[4] Patrick Francis, David Leon, Melinda Minch, and Andy Podgurski. Tree-Based Methods for Classifying Software Failures. In *ISSRE '04: Proceedings of the 15th International Symposium on Software Reliability Engineering*, pages 451–462, Washington, DC, USA, 2004. IEEE Computer Society.

[5] L. Hiew. Assisted Detection of Duplicate Bug Reports. Master's thesis, University of British Columbia, Canada, 2006.

[6] N. Jalbert and W. 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, 2008.

[7] Andrew J. Ko, Brad A. Myers, and Duen Horng Chau. A Linguistic Analysis of How People Describe Software Problems. In *VLHCC '06: Proceedings of the Visual Languages and Human-Centric Computing*, pages 127–134, Washington, DC, USA, 2006. IEEE Computer Society.

[8] G. A. Di Lucca, M. Di Penta, and S. Gradara. An Approach to Classify Software Maintenance Requests. In *ICSM '02: Proceedings of the International Conference on Software Maintenance (ICSM'02)*, page 93, Washington, DC, USA, 2002. IEEE Computer Society.

[9] Andy Podgurski, David Leon, Patrick Francis, Wes Masri, Melinda Minch, Jiayang Sun, and Bin Wang. Automated Support for Classifying Software Failure Reports. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 465–475, Washington, DC, USA, 2003. IEEE Computer Society.

[10] Per Runeson, Magnus Alexandersson, and Oskar Nyholm. Detection of Duplicate Defect Reports Using Natural Language Processing. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 499–510, Washington, DC, USA, 2007. IEEE Computer Society.

[11] R. J. Sandusky, L. Gasser, and G. Ripoche. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004)*, pages 80–84, 2004.

[12] D. Čubranić and G. Murphy. Automatic Bug Triage Using Text Classification. In *Proc. SEKE*, pages 92–97, 2004.

[13] 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 *Proc. of ICSE '08*, pages 461–470, 2008.