# An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information

Xiaoyin Wang[1], Lu Zhang[1*], Tao Xie[2*], John Anvik[3] and Jiasu Sun[1]

[1]Key laboratory of High Confidence Software Technologies, Ministry of Education, Institute of Software, EECS, Peking University, Beijing, 100871, P. R. China, {wangxy06, zhanglu, sjs}@sei.pku.edu.cn

[2]Department of Computer Science, North Carolina State University, Raleigh, NC 27695, USA xie@csc.ncsu.edu

[3]Department of Computer Science, University of Victoria PO Box 3055, STN CSC Victoria, BC, Canada, V8W 3P6 janvik@cs.uvic.ca

## ABSTRACT

An open source project typically maintains an open bug repository so that bug reports from all over the world can be gathered. When a new bug report is submitted to the repository, a person, called a triager, examines whether it is a duplicate of an existing bug report. If it is, the triager marks it as DUPLICATE and the bug report is removed from consideration for further work. In the literature, there are approaches exploiting only natural language information to detect duplicate bug reports. In this paper we present a new approach that further involves execution information. In our approach, when a new bug report arrives, its natural language information and execution information are compared with those of the existing bug reports. Then, a small number of existing bug reports are suggested to the triager as the most similar bug reports to the new bug report. Finally, the triager examines the suggested bug reports to determine whether the new bug report duplicates an existing bug report. We calibrated our approach on a subset of the Eclipse bug repository and evaluated our approach on a subset of the Firefox bug repository. The experimental results show that our approach can detect 67%-93% of duplicate bug reports in the Firefox bug repository, compared to 43%-72% using natural language information alone.

## Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance and Enhancement

## General Terms

Management, Reliability

## Keywords

Duplicate bug report, execution information, information retrieval

## 1. INTRODUCTION

Many open source software projects incorporate open bug repositories[1] during development and maintenance so that both developers and users can report bugs that they have encountered, and call for more useful features or make suggestions for revision. There are at least two important advantages of using such a bug repository. First, the bug repository allows users all around the world to be "testers" of the software, so it can increase the possibility of revealing defects and thus increase the quality of the software [14]. Second, it helps the software evolve according to users' requests, and meet the requirements of more users [1].

However, these advantages come with a cost. Due to a project's reliance on a large number of users acting as testers, this form of testing is asynchronous and loosely organized. Also, the cost of users searching the repository (to determine if their problem has been reported) is higher than the cost of creating a new bug report. As a result, some reported bugs are not new but actually duplicates of existing bugs. To avoid the same bug being addressed by multiple bug fixers, it is necessary for a triager[2] to examine each submitted bug report to determine whether it is a duplicate.

Due to the large number of existing bug reports, it is challenging for the triager to examine all existing bug reports to detect duplication. One solution is that the triager retrieves a small subset of similar bug reports and compares the new bug report with each retrieved bug report to see whether the new bug report is a duplicate. If so, the report is marked as DUPLICATE of the report that it matches. Otherwise, the triager has to assume that there is no duplication [2]. In this paper, we refer to the bug report that the new report duplicates as the "target report", and the set of bug reports retrieved for examination for a given new bug report as the "suggested list". Then, duplicate-bug-report detection can be viewed as the problem of searching for the target report for each new report within the corresponding suggested list.

The quality of the suggested list is essential in the detection of duplicate bug reports. In fact, suggested lists of high quality can reduce both the workload of triagers and the possibility of passing duplicate bug reports to bug fixers. Recently, some research [8] has been conducted to enhance the quality of the suggested list. In general, these approaches adopt information-retrieval techniques to measure the similarity between bug reports using natural language information. Thus, these approaches retrieve only textually similar bug reports for the triager to examine. In particular, the approach proposed by Runeson et al. [18] achieves a recall range of 30%-42% for the Sony-Ericsson Mobile Communications bug repository using suggested-list sizes between 5 and 15. The approach proposed by Hiew [8] achieves a recall range of 36%-50% for the Firefox bug repository using suggested-list sizes between 3 and 7.

Although these approaches already provide some practical help to triagers to detect duplicate bug reports, there is still a need to

* Corresponding authors


*ICSE'08*, May 10–18, 2008, Leipzig, Germany.


[1] We use the colloquial term "bug repository" to refer to the issue tracking system that contains both fault reports and feature requests and "bug report" to refer to contents of bug repositories.

[2] A triager is a person who decides whether a report should be worked on and who should work on it.

improve them due to their low recalls. In this paper, we further consider another important kind of information source: the execution information of bug-revealing runs that cause a bug report to be submitted. Compared to natural language information, execution information has the following advantages. First, execution information can reflect the situation during bug-revealing runs and is not affected by the variety of natural languages. Second, execution information can reflect internal abnormal behavior associated with bug-revealing runs unnoticed by the bug reporter.

We propose an approach using both natural language information and execution information in the detection of duplicate bug reports. Our basic idea is as follows. First, based on information retrieval, we calculate two similarities between the new bug report and each existing bug report using natural language information and execution information respectively. Second, we use some heuristics to determine the suggested list using the preceding similarities. To evaluate our approach, we conducted an experiment, which consists of two parts. First, we calibrated our approach on a subset of the bug repository of the Eclipse[3] project (a popular Java IDE), and also evaluated the calibrated approach on this dataset. Second, we evaluated our approach on a subset of the bug repository of Firefox[4] (a popular open source web browser), and compared the calibrated approach with approaches using only natural language information. Experimental results show that, using suggested-list sizes of 1-10, our approach has a recall range of 67%-93% in the Firefox bug repository, compared to 43%-72% using natural language information alone. This result indicates that our approach achieves a significant improvement in recalls over previous approaches. Furthermore, as the triager can achieve quite a high recall by examining no more than 10 existing bug reports, he or she can be more confident that very few duplicate bugs are missed in the triaging work using our approach than using previous approaches.

This paper makes the following contributions:

- A demonstration of the need to use both natural language information and execution information in detecting duplicate bug reports.
- An approach to detecting duplicate bug reports using both natural language information and execution information.
- An empirical comparison of the effect of using different parameters in our approach: different heuristics using two kinds of similarities and using different natural language sources.
- An experimental evaluation of the proposed approach on a subset of the Firefox bug repository.

The rest of this paper is organized as follows. Section 2 presents two motivating examples to show why the use of both natural language and execution information is necessary. Section 3 presents some background knowledge used in our approach. Section 4 presents our approach. Section 5 reports an experiment of our approach. Section 6 discusses some important issues. Sections 7 and 8 discuss related work and future work, respectively. Section 9 concludes this paper.

# 2. MOTIVATING EXAMPLES

In this section, we present two duplicate-bug pairs from the Firefox bug repository to motivate the need for using both natural language and execution information in duplicate-bug-report detection. In a typical bug report, its natural language part mainly contains two sub-parts: the summary and the detailed description. For brevity, we show only the summary parts in the examples.

## 2.1 Browser-Closing Bug

In the Firefox bug repository, both *Bug-260331* and *Bug-239223* are about the incomplete closing of the browser. *Bug-260331* had been identified as a duplicate of *Bug-239223*. Specifically, their summaries are as follows.

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

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

Both summaries share words like "firefox" and "after closing", but as these words are very common in the Firefox bug repository, it is difficult to use these words to confirm a duplicate relationship between the two bug reports. Furthermore, due to the difference in wording, it is difficult to match some phrases with equivalent meanings like "retain" and "still running", even if synonym lists are used. As a result, it is not easy to decide whether these two reports are duplicates using only natural language processing techniques on the summaries. Including the more detailed descriptions does not necessarily help, because people still use different ways to express the same idea in the detailed description. However, if we involve the execution information in duplicate-bug- report detection, it would be easier to find the common part of the two bugs: their execution traces should share the same abnormal process of quitting Firefox.

This example indicates that using only natural language information may fail to detect some duplicate bug reports due to the variety of natural language usages. In such a case, execution information may be more reliable. However, using only the execution information may also have its own disadvantage. The example below demonstrates this situation.

## 2.2 Document-Contain-No-Data Bug

Another duplicate pair in the Firefox repository is *Bug-219232* and *Bug-244372*, whose summaries are presented below.

***Bug-244372: "Document contains no data" message on continuation page of NY Times article***

***Bug-219232: random "The Document contains no data." Alerts***

In this duplicate pair, the two bugs are both about incorrect loading of web pages. The two natural language descriptions share the common phrase "document contains no data". These words can provide a clear clue that both bug reports are related to an error message "document contains no data". However, *Bug-244372* describes a scenario that the error happens when visiting the web site of NY Times, while *Bug-219232* indicates a random visit. It is highly likely that *Bug-219232* is observed on several web sites that are totally different from the web site of NY Times. Therefore, if we ignore the natural language information but rely on the execution information alone, the part of execution related to loading different files in different pages will be rather different for these two bug reports. These differences may shadow the similar parts of erroneous executions.

This example indicates that using only execution information may fail to detect some duplicate bug reports due to the various ways of observing a bug. In such a case, natural language information may be superior to execution information.

[3] http://www.eclipse.org, accessed on 2007-09-01

[4] http://www.mozilla.com/en-US/firefox, accessed on 2007-09-01

## 2.3 Motivation

In the preceding examples, neither natural language information nor execution information is always superior to the other in all cases. In particular, considering both kinds of information can have the following advantages. First, natural language information acquired from the bug description most likely represents the external buggy behavior observed by the bug reporter, while the corresponding execution information likely records the internal abnormal behavior. Thus, using both kinds of information can make it possible to consider both external and internal behaviors in duplicate-bug-report detection. Second, as descriptions in natural languages often contain uncertainty and imprecision, execution information, which is typically certain and precise, may help reduce the uncertainty and imprecision in existing duplicate-detection approaches. Moreover, as shown by the examples, either natural language information or execution information can be the dominant factor in detecting duplicate bug reports. Thus distinguishing which kind of information is the dominant factor may further facilitate duplicate-bug-report detection.

# 3. BACKGROUND

In our approach, we uniformly deal with both kinds of information sources using information retrieval techniques. Information Retrieval (IR) [7] is a discipline that deals with retrieval of unstructured data, especially textual documents, in response to a query or a topic, which may itself be unstructured or structured.

The vector space model [16] is a widely used technique in traditional information retrieval. Both existing approaches [8][18] to duplicate-bug-report detection adopt the vector space model. In the vector space model, each document or query is represented as an $n$-dimensional vector, where $n$ is the number of unique index terms appearing in all the documents and queries and $w_i$ ($1 \le i \le n$) is the weight of the $i$-th index term in the vector $<w_1, w_2, \ldots, w_n>$ and defined by Formula (1).

$$w_i = tf_i \times idf_i \quad (1)$$

In Formula (1), $tf_i$ refers to the *term frequency* and $idf_i$ refers to the *inverse document frequency*. More precisely, $tf_i$ is the frequency of the $i$-th index term appearing in the document or query, and $idf_i$ is defined by Formula (2).

$$idf_i = log\ (Dsum / Dw_i) \quad (2)$$

In Formula (2), $Dsum$ is the total number of documents, and $Dw_i$ is the number of documents that contains the $i$-th index term.

After transforming documents and queries into vectors, we can calculate the similarity of a pair of documents or queries through a formula defining the similarity of two vectors. Typically, for two vectors $q_1=<w_{11}, w_{21}, \ldots, w_{1n}>$ and $q_2=<w_{21}, w_{22}, \ldots, w_{2n}>$, the similarity of $q_1$ and $q_2$ is defined by Formula (3).

$$Sim = \frac{\sum_{i=1}^{n} w_{1i} w_{2i}}{\sqrt{\sum_{i=1}^{n} w_{1i}^2 \times \sum_{i=1}^{n} w_{2i}^2}} \quad (3)$$

In practice, there are also several other ways [13] of calculating weights and similarities for the vector space model. For simplicity, we present only the formulae used in our approach.

# 4. THE PROPOSED APPROACH

Our approach consists of three steps. First, we calculate the Natural-Language-based Similarities (NL-S) between the new bug report and existing bug reports. Second, we calculate the Execution-information-based Similarities (E-S) between the new bug report and existing bug reports. Finally, we retrieve potential target reports using the two kinds of similarities based on two heuristics. The first heuristic is to combine the NL-S and the E-S into one combined similarity, and use the combined similarity to retrieve potential target reports. The second heuristic is to try to distinguish whether the natural language information or the execution information is the dominant factor in detecting each pair of possible duplicate reports, and use different strategies to deal with different situations.

## 4.1 Calculating NL-S

Our approach adopts a similar technique used in two previous approaches [8][18] to calculate NL-S between two bug reports. First, we extract the text information from the summary and description of the two reports. Second, for the extracted text part of each report, we perform the standard preprocessing in information retrieval, including the stemming work and the removal of the stop words[5]. Third, we obtain one vector for each bug report by applying Formulae (1) and (2). Finally, we use Formula (3) to calculate the similarity for the pair of vectors for the two reports.

There is a slight difference in calculating NL-S between the previous work and our work. As we stated in Section 2, the text part of a bug report mainly contains a summary and a detailed description. Hiew's approach [8] treats the summary and the detailed description equally. Runeson et al. [18] suggested that the summary should be treated as twice as important as the detailed description. That is to say, when calculating the term frequency of an index term, appearing in the summary once will be counted as occurring twice, but appearing in the detailed description once will be counted as once. Ko et al. [10] suggest that using only the summary should be better when dealing with text from bug reports, but they did not provide any experimental evaluation.

Our approach can use natural language in each of the above ways. We experimented with all these approaches, and compare their performance especially when execution information is involved.

## 4.2 Calculating E-S

Similar to the natural language information, our approach also uses the vector space model to calculate E-S. Specifically, we record one execution trace for each reported bug-revealing run, and transform the execution trace into a vector similar to that used for natural language information. In the transformation process, we view each invoked method as a dimension in the vector space model. We use the granularity of methods because many previous approaches [6][15][17] did so. We plan to further investigate the performance of using other levels of granularity in future work. In the specific case of execution information, we further make the following two decisions.

First, when recording an execution trace, we consider only the methods that are invoked during the run without considering how many times each method has been invoked. Intuitively, if a method is invoked during a bug-revealing run, the method may be responsible for the failure, but more invocations of a method do not necessarily imply a higher responsibility.

[5] Stop words are words without enough meaning, such as "the". Stemming is a technique in natural language processing to unify grammatical forms of words. For example, "worked" and "working" are both transformed to "work" after stemming.

Second, the name of a method may contain several words and different methods may share a name due to overloading. We treat the canonical signature of each method as one index term. That is, we treat overloaded methods as different index terms.

Having made these decisions, we transform each execution trace into a vector using Formulae (1) and (2). Note that the term frequency in Formula (1) should be either zero or one, as we treat each method as an index term and we record only the information of whether the method is invoked. Finally, we apply Formula (3) on the vectors to calculate the E-S.

## 4.3 Retrieving Potential Target Bug Reports

### *4.3.1 Basic Heuristic*

After calculating NL-S and E-S, we need to rank the existing bug reports in a list using these two kinds of similarities. The first heuristic is to combine the NL-S and the E-S into one combined similarity, and use the combined similarity to retrieve potential target reports. Thus, we need a tool to combine the NL-S and the E-S. Generally, the combination can be represented as a function in Formula (4).

$$SIM_{combined} = f(SIM_{nlp}, SIM_{exe}) \quad (4)$$

In Formula (4), $SIM_{nlp}$ denotes the NL-S value, $SIM_{exe}$ denotes the E-S value, $SIM_{combined}$ is the combined similarity, and $f$ is the combination function. We consider the most common combination function in our approach: the arithmetic average. Formula (5) formally presents this combination function.

$$SIM_{combined} = \frac{SIM_{nlp} + SIM_{exe}}{2} \quad (5)$$

### *4.3.2 Classification-Based Heuristic*

The preceding heuristic simply treats both kinds of information sources equally for every pair of bug reports. However, as shown in Section 2, in a specific duplicate pair, either natural language information or execution information can be the dominant factor in correctly detecting duplicate bug reports. In such a case, we should naturally rely more on the dominant information source. Thus, we further propose a classification-based heuristic that is based on distinguishing which kind of information source is the dominant factor.

When analyzing some pairs of duplicate bug reports, we found that an extremely high value of NL-S often indicates a case that natural language information dominates, and there is a similar observation for E-S. Thus, when ranking the existing bug reports, those with extremely high NL-S or E-S values should be ranked above other bug reports. Furthermore, we can also define two thresholds on the similarity values to distinguish whether one kind of information source is dominant. Here, we refer to such a threshold as the Credibility Threshold (CT), and use $CT_{NL\text{-}S}$ and $CT_{E\text{-}S}$ to denote credibility thresholds for NL-S and E-S, respectively. For an existing bug report, our classification-based heuristic uses the following strategy to calculate its ranking:

- If its NL-S and E-S are both higher than the corresponding CTs, we put the bug report in Class I, in which bug reports are ranked by their combined similarity using Formula (5).
- If its NL-S is higher than $CT_{NL\text{-}S}$, but its E-S is lower than $CT_{E\text{-}S}$, we put it in Class II, in which bug reports are ranked only by NL-S.
- If its E-S is higher than $CT_{E\text{-}S}$, but its NL-S is lower than $CT_{NL\text{-}S}$, we put it in Class III, in which bug reports are ranked only by E-S.
- Otherwise, we put it in Class IV, in which bug reports are ranked by their combined similarity using Formula (5).

In the final ranking, Class I is ranked higher than Class II, Class II higher than Class III, and Class III higher than Class IV. The reason that we put the class of natural-language-dominant bug reports (i.e., Class II) higher than the class of execution-information-dominant bug reports (i.e., Class III) lies in that $CT_{E\text{-}S}$ cannot always definitely identify execution-information-dominant bug reports. In fact, for two very similar or even identical bug-revealing execution traces, it is still possible that the two execution traces reveal different bugs, especially when the granularity of the execution traces is at the method level.

### *4.3.3 Determining Credibility Thresholds*

For our classification-based heuristic, we need two credibility thresholds. In this sub-section, we present a technique to calculate these thresholds. Our technique is based on the analysis of existing bug reports, among which the duplicate relationships are known. For simplicity, we describe only how to determine $CT_{NL\text{-}S}$. The determination of $CT_{E\text{-}S}$ is similar.

As the aim of $CT_{NL\text{-}S}$ is to rank bug reports with extremely high NL-S values higher in the list, the intuition in determining $CT_{NL\text{-}S}$ is to choose a value (denoted as $v$) such that on average bug reports whose NL-S values are larger than $v$ are more likely to be duplicate bug reports. Our technique is as follows.

Given a set of existing bug reports (denoted as $S$), we use $D$ to denote a subset of $S$ where for each bug report (denoted as $b$) in D, there is at least one duplicate bug report of $b$ in S. For each bug report $b$ in $D$, we calculate the NL-S value between $b$ and each other bug report in $S$. For a given value $v$, we use $dup(b, v)$ to denote the number of duplicate bug reports whose NL-S values with $b$ are larger than $v$; and we use $fp(b, v)$ to denote the number of false-positive bug reports whose NL-S values with $b$ are larger than $v$. Thus, we define the effectiveness of $v$ (denoted as $E(v)$) as in Formula (6):

$$E(v) = \sum_{b \in D} (dup(b, v) - fp(b, v)) \quad (6)$$

Based on Formula (6), we calculate the effectiveness of a series of different values, and choose the value with the largest effectiveness as the threshold.

## 4.4 Presenting Potential Target Bug Reports

When presenting the ranked list of retrieved potential target bug reports to the triager, there are two ways used in previous research: suggesting a list with a predetermined fixed size [18] and suggesting a list with floating sizes [8]. The fixed list size is determined using a predefined number. The floating list sizes are determined using a threshold of similarity. For our basic heuristic, both ways can be adopted. Our classification-based heuristic can use only a fixed suggested-list size, as this heuristic is not based on one combined similarity but a collection of four different sets. In our experiment, we use the fixed list size for both heuristics for the ease of comparison.

# 5. EXPERIMENT

In our experiment, we investigate two research questions. First of all, we want to discover the setting under which our approach can achieve a good performance. Second, we want to see whether our approach can outperform approaches using only natural language information. To evaluate the performance of the different settings

and approaches, we use the recall rate[6] of target reports under a certain suggested-list size as a measure. This technique was suggested by Runeson et al. [18]. Formula (7) shows this measure.

$$recall\ rate = \frac{N_{recalled}}{N_{tatal}} \quad (7)$$

In Formula (7), $N_{recalled}$ refers to the number of duplicate bug reports whose target reports are in the suggested lists, and $N_{total}$ refers to the number of duplicate bug reports used in experiment.

## 5.1 Experimental Setup

In our experiment, we used the bug repositories of two large open source projects: the Eclipse project and the Firefox project. These two projects are from different domains and used by different types of users. Thus, carrying out the experiment on them helps to generalize our conclusions. Also both projects have large bug repositories so as to provide ample data for an evaluation. We selected a subset of each repository to set up an experimental bug set in our study. When setting up the two experimental bug-report sets, we did not select the most recent bug reports, because the resolutions of recent bug reports are more likely to be changed compared to older bug reports. For example, some new bug reports may be incorrectly marked as duplicate and the mistakes have not yet been corrected. Including these new bug reports would affect the precision and fidelity of our experiment.

Typically bug repositories also contain invalid bug reports. A report may be invalid for several reasons, such as not being reproducible or being filed by a spambot. In practice, when encountering an invalid bug report, the triager marks it as invalid and the report receives no further attention, but the report remains in the repository. To avoid interference by invalid bug reports, we discarded such reports when setting up an experimental bug-report set.

Our approach requires both natural language information and execution information for each bug report. In both the Eclipse and Firefox bug repositories, a bug report provides a summary and a detailed description, both of which contain natural language information, but there is no execution information associated with the bug report. Therefore, we needed to create the execution information for each bug report used in our study.

For an experimental bug-report set used in our experiment, we classify the bug reports into three main types: runtime error reports, feature requests, and patch reports. A runtime error report is an erroneous behavior or crash that the reporter encounters when he or she uses an existing feature of the software. A feature request is a request for a non-existing feature of the software. A patch report refers to a bug report (submitted by a highly technical bug reporter) that directly points to a bug in the code with a suggestion of fixes. For different types of reports, we used different techniques to obtain the corresponding execution information.

- For a runtime error report, we started the program and ran the program until the error occurs. As the steps to reproduce a runtime error are provided in the description part of the bug report, we reproduced these bugs according to these steps. Thus the execution information of our reproduction should be very close to the execution information that caused the fault.
- For a feature request, we started the program and ran the program until we reached the point where the new feature (suggested by the reporter) should appear. Like runtime errors, there is also guidance in the description part of a feature request for the bug fixer to reach the point where the new feature is desired. For example, to reproduce a bug calling for automatic sorting of bookmarks, we ran the browser until the list of the bookmarks was shown.
- For a patch report, there is no associated bug-revealing run and a bug reporter will not submit execution information. Furthermore, it is unlikely that two independent bug reporters identify the same buggy code simultaneously. Actually, in the part of bug repository from which we build our experiment set, we did not find any duplicate patch reports. Therefore, we did not use patch reports in the experiment. This decision does not affect the validity of our experiment, as patch reports can be easily distinguished from other reports in a bug repository that involves execution information due to these patch reports' lack of execution information.

## 5.2 Calibration and Evaluation on Eclipse

As there are several parameters in our approach, it is necessary to calibrate our approach experimentally. To do so, we set up a small experimental bug-report set using a subset of bug reports from the Eclipse bug repository. To create the subset, we randomly selected 200 bug reports submitted to the Eclipse repository during June 2004. To make sure that the set of experimental bug-reports contained enough duplicate pairs, we further added the target reports of some duplicate bug reports. After filtering out patches and invalid bug reports, our experimental bug-report set contained 220 bug reports with 44 pairs of duplicate bug reports.

Furthermore, to calculate the credibility thresholds, we randomly selected 200 other bug reports submitted during May 2004, and used the same technique to build a set of 232 bug reports with 42 pairs of duplicates. We applied the technique described in Section 4.3.3 to calculate the credibility thresholds. We determined the values of $CT_{NL\text{-}S}$ as follows: 0.43 for using summary only, 0.55 for using summary and description with equal weights, and 0.53 for using a double-weighted summary. The value of $CT_{E\text{-}S}$ is 0.94.

Using the experimental bug-report repository, we evaluated different combinations of parameters for our approach. For each of the 44 duplicate pairs, we used one bug report as the new bug report and the other 219 bug reports as the existing bug reports. For each different parameter combination, we recorded the recall rate (calculated with Formula (7)) for the 44 bug reports for different suggested list sizes. Note that our evaluation is based only on the 44 bug reports for which we have a target report among the existing reports. The rationale is that if we used a new bug report that is not a duplicate of any existing bug report, the triager will never find the target report after examining the suggested list. In other words, all the evaluated parameter combinations become equally effective for a fixed suggested-list size of unique new bug reports. In our calibration, we considered two parameters. The first parameter was how to use the natural language information from each bug report. As discussed in Section 4.1, we considered three options for this parameter: using the summary only, using both the summary and the detailed description with equal weights, and using both the summary and the detailed description with the summary double-weighted. The second parameter examined was the heuristics for retrieving potential target reports using the two kinds of similarities: the basic heuristic and the classification-based heuristic. Furthermore, we considered a variant of the classification-based heuristic, in which execution-information

[6] We follow Runeson et al's naming [18] but we think that it can also be called accuracy (the percentage of the correct target reports that appeared in suggested lists with the same size).

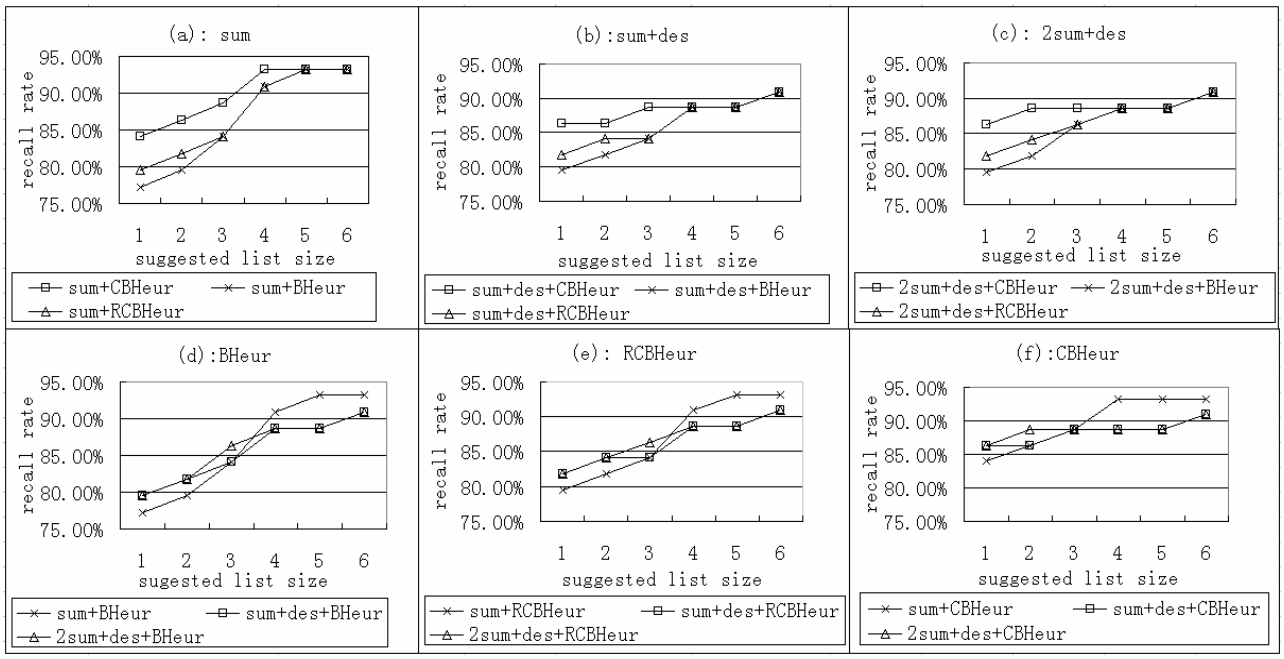


**Figure 1: Recall rates using different parameters in Eclipse**

-dominant bug reports are ranked higher than natural-language-dominant bug reports. For simplicity, we refer to this variant as the reverse classification-based heuristic. The aim of the calibration is to examine which combination of different values for the two parameters performs the best for the bug reports in the experimental bug-report set.

In Figure 1, we use six sub-figures to show the recall rates using the nine different combinations of the parameters for Eclipse. In all the six sub-figures, the x-axis denotes the different suggested-list size used, and the y-axis denotes the recall rate of our approach for the suggested-list size. We use "BHeur" as the abbreviation for the basic heuristic, "CBHeur" for the classification-based heuristic, and "RCBHeur" for the reverse classification-based heuristic. We use "sum" for using only the summary, "sum+ des" for using both the summary and the description with equal weights, and "2sum+des" for using both the summary and the description with the summary double-weighted. The upper three sub-figures show the results of using the three different kinds of natural language information, respectively. In each of these three sub-figures, we compare the recall rates of using different heuristics. The lower three sub-figures show the results of using the three different kinds of heuristics, respectively. In each of the lower three sub-figures, we compare the recall rates of using the three different kinds of natural language information.

From Figure 1, we make the following observations. First, the three upper sub-figures show that if we fix the parameter of how we weight the natural language information, the classification-based heuristic always outperforms the other two heuristics. This observation indicates that the classification-based heuristic is an improvement over the basic heuristic. This observation also confirms that in the classification-based heuristic, natural-language-dominant bug reports should be ranked higher than execution-information-dominant bug reports. It is also interesting to note that for each of the three upper sub-figures, the difference between the three heuristics becomes smaller as the suggested-list size becomes larger. We suspect the reason to be that natural-language-dominant bug reports and execution-information-dominant bug reports in the classification-based heuristic and its variant are often similar to the new bug report according to their combined similarities. Therefore, when the suggested-list size becomes larger, the three heuristics will retrieve roughly the same set of existing bug reports. But the ordering of bug reports retrieved by each heuristic differs. The ordering used by the classification-based heuristic appears to be the best among the three heuristics.

Second, the three lower sub-figures show that if we fix a specific heuristic in bug-report retrieval, neither way of using the natural language information always outperforms the other two. When the suggested-list size is small, using both the summary and the description seems superior to using only the summary. When the suggested-list size becomes larger, using only the summary becomes superior to using both the summary and the description. We suspect the reason to be that the description contains both clues and noise not contained in the summary. The clues can help effectively determine the ordering of retrieved bug reports, but in some cases the noise can make our approach fail to retrieve target bug reports. Of the two ways of weighting both the summary and description, using the double-weighted summary seems to perform slightly better than using the single-weighted summary. This result is in accordance with the finding of Runeson et al. [18].

Although the experimental bug-report set formed from the Eclipse bug repository is quite small and also includes manually inserted duplicate bug reports, we believe that evaluation on this experimental bug-report set provides some initial insights about whether our heuristics using both natural language information and execution information can be an improvement over approaches that use only a single type of information.

Specifically, as the reverse classification-based heuristic is a less effective variant of the classification-based heuristic, we considered only the basic heuristic and the classification-based heuristic for our approach in the evaluation. Similarly, as using double-weighted summary with detailed description always outperforms using both summary and detailed description with equal weights, we did not consider the way of using equally-weighted summary and detailed

description for our approach in the evaluation. Therefore, we had four different combinations of parameters for our approach. As a comparison, we also considered three approaches using only the natural language information (i.e. summary only, equally-weighted summary and description, and double-weighted summary and description) and one approach using only the execution information. As before, we used each of the 44 duplicate bug reports as the new bug report and the other 219 bug reports as the existing bug reports in evaluating each approach. Figure 2 shows the result of the evaluation for suggested-list sizes of 1-6.

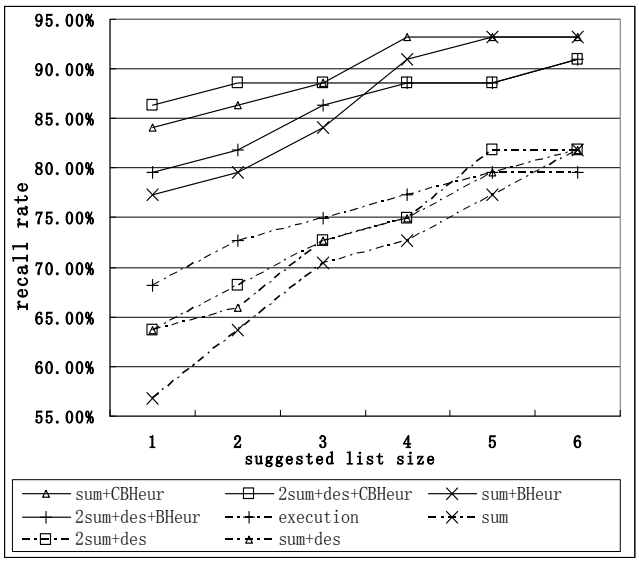


**Figure 2: Recall rates using different similarities in Eclipse**

From Figure 2, we make the following three observations. First, the four parameter combinations for our approach always outperform the other four approaches. In our approach, as the classification-based heuristic always outperforms the basic heuristic, we focus on the two combinations using the classification-based heuristic. When comparing the performance of our approach using both the summary and the execution information with the classification-based heuristic to the best performance of using only the natural language information, there is an increase of 11-20 percentage points for suggested-list sizes of 1-6. The increase of the other combination over the best performance of using only the natural language information is 7-22 percentage points.

Second, when the suggested-list size is small, using only execution information appears to outperform using only natural language information. When the suggested-list size becomes larger, using only natural language information becomes superior to using only execution information. We suspect the reason to be that execution information is more precise than natural language information. The precision in information leads to a better order of the retrieved bug reports. However, as bug reports with quite similar or even identical execution information are not guaranteed to be duplicate bug reports, using only execution information may become more likely to retrieve irrelevant bug reports than using only natural language information when suggested lists are large.

Finally, among the three approaches using only natural language information, the one using the double-weighted summary with the detailed description appears to achieve the best performance, and the approach that uses only the summary appears to perform the worst. This observation is different from the situation of using natural language information together with execution information, where using only the summary becomes superior when the suggested-list size becomes larger. We suspect the reason to be that, without the other kind of information sources, only the summary cannot provide enough clues to achieve a high-quality set of retrieved bug reports or a high-quality ordering of the retrieved bug reports. As using the double-weighted summary with the detailed description performs slightly better than using the equal-weighted summary and description, this observation once again confirms the finding of Runeson et al. [18].

## 5.3 Evaluation on Firefox

We used the Eclipse bug repository mainly to calibrate the parameters of our approach. To further evaluate our approach, we carried out another experiment using data from the Firefox bug repository. When establishing the experimental bug-report set, we considered the two following issues. First, Runeson et al. [18] suggested that searching the bug reports that are submitted 50 days before the new bug report is the most effective for searching target reports. Thus we created our experimental report set using bug reports submitted in three consecutive months, and treated the bug reports submitted in the first 50 days as existing bug reports and the remaining bug reports as new bug reports. Second, there is usually an intensive bug-fixing period after a major release [2] . In this period, the triager usually has the largest triaging workload. Therefore, we downloaded all resolved bug reports between Jan. 1st 2004 and Apr. 1st 2004, which is around the release of version 0.8 on Feb. 6th 2004. In open source projects, some bug reports, especially those submitted shortly before the new release, remain unsolved in the new release, and users of the new release may rediscover them, and submit a duplicate bug report. Therefore, we included bug reports both before and after the release of version 0.8 in our experimental data set. Note that those bug reports treated as new ones were submitted after the release of version 0.8.

In total, we collected 1749 bug reports. After filtering out the patch reports and invalid bug reports, we reproduced the execution traces of the remaining bug reports and established an experimental bug-report set containing 1492 bug reports. Among the 1492 bug reports, there are 744 bug reports submitted in the first 50 days. We treated them as the existing bug reports. We treated the other 748 bug reports as the set of new bug reports.

Similar to our evaluation of Eclipse in Section 5.2, we calculated the credibility thresholds using the 744 existing bug reports. We can do this calculation because the duplicate relationships among existing bug reports can be acquired before triaging new bug reports. The values of $CT_{NL\text{-}S}$ are as follows: 0.39 for using the summary only, 0.57 for using the summary and description with equal weights, and 0.55 for using the double-weighted summary. The value of $CT_{E\text{-}S}$ is 0.95. Note that these values are similar to those acquired in our evaluation on Eclipse.

We then added the bug reports from the set of new bug reports to the existing bug-report set in the same order as they were submitted to the Firefox bug repository. Each time we added a bug report, we performed duplicate-bug-report detection using the same approaches compared in our evaluation on Eclipse. Similar to our evaluation on Eclipse, for different suggested-list sizes, we recorded the performances of the eight approaches on only those new bug reports each of which has a duplicate counterpart in the set of existing bug reports, as all the eight approaches become equally effective for unique new bug reports.

When adding bug reports chronologically to our experimental bug-report set, we faced the *inverse duplicate problem*. In a bug repository, a bug report will sometimes be marked as the duplicate of a

future bug report. This problem has been observed in previous research [8][18]. We used the same solution used in previous research. We designated the earliest bug report in a group of duplicate bug reports as the target report. We also marked the duplicate bug reports whose target report was not in our experimental bug-report set as "unique" instead of "duplicate". Under this strategy, there were totally 77 duplicate bug-report pairs in our evaluation.

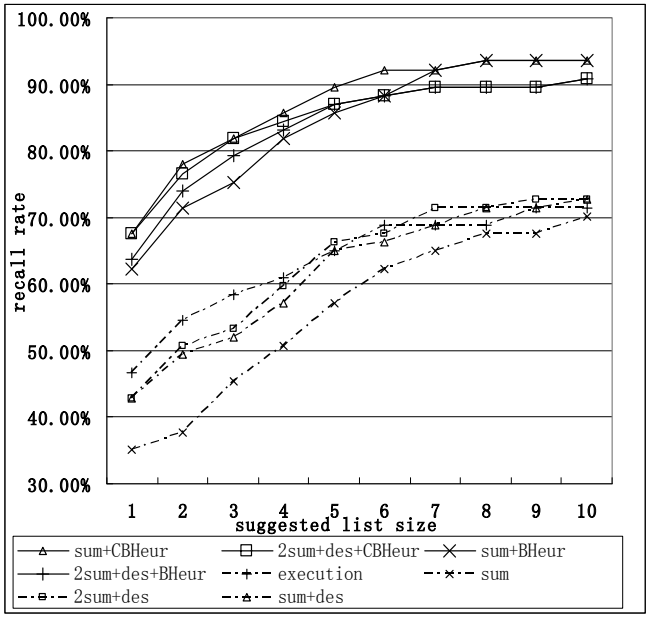


**Figure 3: Recall rates using different similarities in Firefox**

The results of the evaluation on Firefox are shown in Figure 3. We make similar observations from this figure as observed in Figure 2. The results can confirm almost all the findings in our evaluation on Eclipse. First, the four combinations for our approach outperform the other four approaches. Unlike the results on Eclipse, among the four combinations of our approach, the one using the summary and the execution information with the classification-based heuristic outperforms the other combinations. This combination achieves recall rates of 67%-93% for suggested-list sizes of 1-10. However, there is a similar trend in both Figures 2 and 3: when the suggested-list size is small, our approach that uses only the summary has no advantage over our approach that uses the double-weighted summary with the detailed description. The advantage becomes significant only when the suggested-list size becomes larger. Second, for approaches using only the natural language information and only the execution information, using only execution information performs better with small suggested-list sizes, and using only natural language information performs better with larger suggested-list sizes. Finally, among the three approaches using only natural language information, the one using the double-weighted summary with detailed description outperforms the other two.

In summary, the best result of our approach has an increase of 18-26 percentage points in recall rates over the best result of using only natural language information for suggested-list sizes of 1-10.

## 5.4 Threats to Validity

As there was no execution information associated with bug reports in both Eclipse and the Firefox bug repositories, we used reproduced execution traces for our experiment. Thus, these reproduced execution traces are a threat to the construct validity, as the execution traces that we reproduced may be somewhat different from those actually recorded by the bug reporters. Such differences may be due to our misunderstanding of the bug reports or due to external influence beyond our control. To reduce this threat, we carefully examined the summary and the detailed description of each bug report before reproducing the execution traces for it.

In our experiment, we used subsets of bug reports in the original bug repositories. This factor may be a threat to the external validity, as the experimental results may be applicable only to the selected bug reports but not to the entire bug repositories. To reduce this threat, we tried to use as many bug reports as possible in our experiment.

Another threat to the external validity is the bug repositories used in our experiment. It is possible that some particular characteristics of a bug repository lead to our experimental results. To reduce this threat, we used two different real-world bug repositories.

# 6. DISCUSSION

## 6.1 Costs of Using Execution Information

Our experimental results show that using both the execution information and the natural language information can be more effective than using the natural language information alone. However, there are costs to achieve this effectiveness.

The first cost is that the bug repository has to store the execution information in addition to the natural language information for each bug report. Thus, the repository would increase its storage requirement for bug reports. For example, one typical execution trace for an Eclipse bug report in our experiment contains around 30,000 methods. If we record the full signature of each method in an execution trace file, one execution trace file may require more than 1MB to store. However, if the bug repository maintains a table of the signatures of all the methods, an execution trace can be represented as a series of indices to the table or a series of Boolean values, each indicating whether the trace contains a method. Thus, we can keep the storage requirement for execution information within an acceptable level. Furthermore, as this way of execution trace representation is similar to the vector representation, it also facilitates the transformation of execution traces to vectors.

The second cost of using execution information is the cost of calculating the E-S. As an execution trace typically contains much more index terms than its corresponding textual part, calculating the E-S is more time-consuming than calculating the NL-S. This cost may become a burden for the triager to retrieve potential target bug reports. However, as stated in Section 5.3, we can match the new bug report with only recently submitted bug reports using a time frame [18]. Furthermore, there are some methods that frequently appear in many execution traces. Due to the nature of the vector space model, the inverse document frequency (*idf*) of such a frequently appearing index term is small. So in practice we can ignore some methods with very small *idf* in each execution trace, and thus decrease the number of index terms in each execution trace. As a result, we can decrease the costs of both storing execution information and calculating E-S through representing each execution trace with fewer methods. Indeed, this simplification may induce imprecision and requires further investigation.

The third cost of using execution information is the burden posed on bug reporters. To record the execution information, bug reporters need to run an instrumented version of the software and submit execution information. This instrumentation may lengthen the time of testing the system. However, we do not think this lengthening will be a burden for bug reporters, because the overhead of executing instrumented code is usually a small portion of executing the original code. Recently, Clause and Orso [3] suggested an effective ap-

proach to automatically reproduce bug-revealing runs and record execution information associated to bug reports submitted by remote users. This technique further relieves the burden on bug reporters. Another problem is that bug reporters may refuse to submit execution information because it contains private data. As the execution information that our approach requires includes only a list of executed methods but does not include data parts such as the values of variables, we believe that the privacy issue would not be a concern in our approach.

## 6.2 Coping with Existing Bug Repositories

Our approach requires both execution information and natural language information in bug repositories. It is easy for a new open source project to create such a bug repository at beginning. However, if a project already maintains a bug repository that does not contain execution information, the project cannot directly take advantage of our approach. To facilitate such a project to migrate to a bug repository supporting the inclusion of execution information, we suggest the following strategy. First, each bug report in the existing repository can be transformed into a bug report in the new bug repository with the execution information part marked as "missing". Second, when a new non-patch bug report is submitted, the bug repository requests both natural language information and execution information. Third, when the triager detects whether a new bug report is a duplicate bug report, we calculate the E-S between the new bug report and an existing bug report differently according to whether the existing bug report contains execution information. If the existing bug report contains execution information, we use our proposed approach to calculate the similarity; otherwise, we assume that the similarity is the average of the similarities between the new bug report and all the existing bug reports that contain execution information. Specifically, let $R$ be the set of existing bug reports containing execution information, and $i$ be a new bug report, we use Formula (8) to calculate the average E-S.

$$AvgSim_{exe}(i) = \frac{1}{n}\sum_{x \in R} Sim_{exe}(i, x) \qquad (8)$$

In Formula (8), $n$ is the number of bug reports in $R$. We use this similarity uniformly as the E-S between $i$ and any existing bug report with "missing" execution information. Using this formula, the E-S between a new bug report and any existing bug report with "missing" execution information will be the same. Therefore the ranking of these bugs will be decided only by the NL-S.

## 6.3 Evaluation Criteria for Duplicate-Bug-Report Detection

The ultimate criterion to evaluate an approach for duplicate-bug-report detection should be how much workload is saved for the involved developers. However, the saved workload may be difficult to measure in practice. In our experiment, we adopt the evaluation criterion used by Runeson et al. [18]. In this criterion, an approach is assessed through measuring how many duplicate bug reports the approach can detect under a given suggested-list size. This criterion is applicable under the following conditions: (1) the compared approaches use the same suggested-list size; and (2) the triager checks all the suggested bug reports with equal effort.

Given a set of existing bug reports and a new bug report, if two approaches require different suggested-list sizes, these approaches sometimes are incomparable to each other. For example, for a duplicate new bug report, Approach 1 retrieves $n$ bug reports not containing the target bug report, and Approach 2 retrieves $m$ bug reports containing the target bug report. If $n$ is less than $m$, it is unclear which approach is superior. On one hand, Approach 1 fails to detect the duplication but Approach 2 can. This observation indicates that Approach 2 is better than Approach 1. On the other hand, Approach 1 requires less effort for the triager to examine the retrieved bug reports. In fact, this situation reflects the original dilemma that more effort from the triager to detect duplicate bug reports can make developers waste less effort in bug fixing. Without setting up an accurate relationship between the two kinds of effort, we cannot have a satisfactory evaluation of the preceding situation.

Moreover, some bug-report pairs are easy for the triager to determine whether one duplicates the other; but other bug-report pairs may be difficult. Thus, the number of bug reports that the triager examines may not accurately reflect the effort that a triager invests. Even worse, different triagers may feel differently about how hard it is to examine the same bug report. Therefore, the current evaluation criterion used by us and others is only a coarse simplification. More effort is needed to set up a more accurate criterion.

# 7. RELATED WORK

As previously mentioned, there have been some approaches to duplicate-bug-report detection reported in the literature. Runeson et al. [18] proposed an approach based on information retrieval. This approach uses only the natural language information of bug reports without considering the use of execution information. Hiew [8] also proposed an approach to duplicate-bug-report detection using the natural language information alone. Hiew's approach is based on incremental clustering, which is quite similar to information retrieval. The main difference is that Hiew's approach further considers the detected duplicate bug-report pairs/groups as clusters. Thus, when calculating similarities between a new report and existing bug reports, each detected cluster is considered as a whole rather than as several individual existing bug reports. That is to say, for each detected cluster, this approach will calculate one similarity between the new report and the detected cluster instead of calculating several similarities between the new report and all the reports in the cluster. Compared to these two approaches, our approach further considers execution information, and uses effective heuristics to combine the two kinds of information.

Besides the effort on duplicate-bug-report detection, there has also been some effort made on bug-report mining. Anvik et al. [1], Cubranic and Murphy [4] and Lucca et al. [12] all proposed semi-automatic techniques to categorize bug reports. Based on categories of bug reports, their approaches help assign bug reports to suitable developers. All the three approaches rely on only natural language information. Podgurski et al. [15] also proposed an approach to bug-report categorization, and later improved their approach using two tree-based techniques in the clustering process [6]. However, their approaches target at prioritizing bug reports by their severity and frequency, not assigning bug reports to developers. Another main difference is that their approaches [6][15] use only execution information but not natural language information. Ko et al. [10] analyzed the linguistic characteristics of bug-report summaries, and proposed a technique to differentiate failure reports and feature calls. They also emphasized the need of duplicate-bug-report detection, but did not propose a solution. In general, although either natural language information or execution information has been used in bug-report mining, no previous approach to bug-report mining has used both kinds of information.

There have been several statistical studies of existing bug repositories. Anvik et al. [2] reported a statistical study on open bug repositories with some interesting results such as the proportion of different resolutions and the number of bug reports that a single reporter submitted. Sandusky et al. [19] studied the relationships between

bug reports and reported some statistical results on duplicate bugs in open bug repositories. However, neither work proposed any approaches to duplicate-bug-report detection.

There are also approaches on analyzing execution information of bug-revealing runs for debugging [9][11][17]. However, these approaches mainly focus on debugging using execution information, not on how to detect duplicate bug reports.

Some research has been conducted on automatic remote execution-information collection. Elbaum and Diep [5] suggested a strategy to collect remote execution information over Internet; Liblit et al [11] proposed a technique to sampling remote runs for bug isolation. Clause and Orso [3] suggested a technique on automatic reproduction and execution-information recording of remote bug-revealing runs. These techniques, which improve the collection of remote execution information, can facilitate the use of our approach, which requires execution information, but none of these previous techniques target the problem of duplicate bug-report detection.

# 8. FUTURE WORK

There are several ways to improve or extend our current research. First, there are the previously mentioned threats to validity of our experiment. Our future plan includes conducting experiment on larger sets and deploying the tool on some open source projects to see if triagers benefit from our approach, so that we can further address these threats.

Second, as our research indicates that the combination of natural language and execution information provides a more precise representation of bug reports, the combination may be useful in areas beyond duplicate-bug-report detection. For example, automated bug assignment and bug prioritization may also benefit from the combination.

Third, in Section 6.1, we discussed some simplifications of our approach to cope with some practical situations. In future work, we plan to further investigate these simplifications.

Finally, as current evaluation techniques for duplicate-bug-report detection are not precise enough, we plan to further investigate this issue and develop more-suitable evaluation techniques.

# 9. CONCLUSION

In this paper, we present a novel approach to assist triagers in detecting duplicate bug reports. Unlike existing approaches, our approach further considers execution information. Furthermore, our approach employs two heuristics to combine the two kinds of information. We also calibrated and evaluated our approach on bug reports from the Eclipse and Firefox repositories. The experimental results show that, compared with the best performance of approaches using only natural language information, our calibrated approach (with the classified-based heuristic and using only the summary) leads to an increase of 11-20 percentage points and an increase of 18-26 percentage points in recall rates on the two experimental bug-report sets respectively.

## Acknowledgments

We are grateful to Prof. Hong Mei, for his guidance and support in both the research and paper writing process. The authors from Peking University are sponsored by the National 973 Key Basic Research and Development Program No. 2002 CB312003, the State 863 High-Tech Program No. 2006AA01Z 156 and the National Science Foundation of China No.60403015. Tao Xie's work is supported in part by NSF grant CNS-0720641 and Army Research Office grant W911NF-07-1-0431.

# 10. REFERENCES

[1] Anvik, J., Hiew, L., and Murphy, G. Who Should Fix This Bug? *In Proc. ICSE.*, 2006, 371-380.

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

[3] Clause, J. and Orso, A. A Technique for Enabling and Supporting Debugging of Field Failures. *In Proc. ICSE*, 2007, 261-270

[4] Cubranic, D. and Murphy, G. Automatic Bug Triage Using Text Classification. *In Proc. SEKE*, 2004, 92-97.

[5] Elbaum S. and M. Diep. Profiling Deployed Software: Assessing Strategies and Testing Opportunities. *IEEE TSE*, 31, 4: p312–327, 2005.

[6] Francis, P., Leon, D., and Minch, M. Tree-Based Methods for classifying Software Failures. *In Proc. ISSRE*, 2004, 451-462.

[7] Greengrass, E. Information Retrieval: A Survey, University of Maryland, Baltimore County, 2000

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

[9] Hildebrandt, R. and Zeller, A. Simplifying failure-inducing input. *In Proc. ISSTA*, 2000, 135-145.

[10] Ko, A., Myers, B., and Chau, D.H. A Linguistic Analysis of How People Describe Software Problems in Bug Reports. *In Proc. of IEEE Conf. on Visual Language and Human-Centric Computing (VL/HCC)*, 2006, 127-134.

[11] Liblit B., Aiken A. and Zheng A. Bug Isolation via Re-mote Program Sampling. *In Proc. PLDI*, 2003, 15-26.

[12] Lucca, D., Penta, D., Granada, S., An Approach to Classify Software Maintenance Requests. *In Proc. ICSM,* 2002, 93-102.

[13] Manning, D., Schutze, H. Foundations of Statistical Natural Language Processing. Cambridge, USA, MIT Press 1999.

[14] Mockus, A., Fielding, R., and Herbsleb, J. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM TOSEM*, 11, 3: p309-346, 2002

[15] Podgurski, A., Leon, D., and Francis, P. Automated Support for Classifying Software Failure Reports. *In Proc. ICSE*, 2003, 465-475.

[16] Raghavan, V., Wong, M. A critical analysis of vector space model for information retrieval. *Journal of the American Society for Information Science*, 37, 5: p279-287, 1986.

[17] Reiss, S., and Renieris, M. Encoding Program Executions. *In Proc. ICSE*, 2001, 221-230.

[18] Runeson, P., Alexanderson, M., Nyholm, O. Detection of Duplicate Defect Reports Using Natural Language Processing. *In Proc. ICSE,* 2007, 499-510.

[19] Sandusky, J., Gasser, L., and Ripoche, G. Bug Report Networks: Varieties, Strategies, and Impacts in an OSS Development Community, *In Proc. MSR,* 2004, 80-84.