Automatic Bug Assignment Using Information Extraction Methods

Ramin Shokripour* Zarinah M. Kasirun* Sima Zamani* John Anvik†

*Faculty of Computer Science & Information Technology
University of Malaya
Kuala Lumpur, MALAYSIA
Shokripour@siswa.um.edu.my,
zarinahmk@um.edu.my,
sima.zamani@siswa.um.edu.my

†Department of Computer science
Central Washington University
Ellensburg, Washington, United States of America
janvik@cwu.edu

***Abstract*—The number of reported bugs in large open source projects is high and triaging these bugs is an important issue in software maintenance. As a step in the bug triaging process, assigning a new bug to the most appropriate developer to fix it, is not only a time-consuming and tedious task. The triager, the person who considers a bug and assigns it to a developer, also needs to be aware of developer activities at different parts of the project. It is clear that only a few developers have this ability to carry out this step of bug triaging. The main goal of this paper is to suggest a new approach to the process of performing automatic bug assignment. The information needed to select the best developers to fix a new bug report is extracted from the version control repository of the project. Unlike all the previous suggested approaches which used Machine Learning and Information Retrieval methods, this research employs the Information Extraction (IE) methods to extract the information from the software repositories. The proposed approach does not use the information of the bug repository to make decisions about bugs in order to obtain better results on projects which do not have many fixed bugs. The aim of this research is to recommend the actual fixers of the bugs. Using this approach, we achieved 62%, 43% and 41% accuracies on Eclipse, Mozilla and Gnome projects, respectively.**

***Index Terms*—Bug Assignment; Information Extraction; Named Entity Recognition; File Activity Histories**

I. Introduction

Debugging, as one of the significant steps of the Open Source Software (OSS) development process, has a considerable effect on the quality of the projects [1]. The speed of the debugging and easy access for users to bug reports are two important factors to consider in order to satisfy the users of the OSS. Hence, open source projects usually use a Bug Tracking System (BTS) to improve the debugging process. At any time and from any where, users and developers can use the BTS to report the problems which occurred on the project. In addition, anyone can read about the bugs reported by others and even add comments on the reports. The reported bugs and other data about them, which are sent to the project through the BTS, are stored in a Bug Repository (BR). Hence, there is a lot of useful information in this repository which can be used to improve the process of OSS development.

BTS has improved the first step of the debugging process, but the next steps (such as bug triaging) have a significant influence on the quality of debugging. For example, making decisions about the validation of bugs and assigning bugs to an appropriate developer to fix them, take a lot of time [2] and reduces the speed of the bug fixing process.

In addition, the high number of the reported bugs in large open source projects (such as Eclipse and Mozilla) increases the time of the bug triaging process and the cost of the projects. Hence, it spurs research on improving different aspects of the bug triaging process (such as automatically identifying the duplicate bugs [3], [4] and determining the quality of bugs [5]).

Selecting the most appropriate developer to fix a new bug report is one of the important stages in the bug triaging process which has a significant effect in decreasing the time taken for the bug fixing process [2] and the cost of the projects [6]. In traditional bug triage systems, a developer who is dominant in all parts of the project as well as the activities of the developers, plays the triager role in the project. The triager reads a new bug report, makes a decision about the bug, and then selects the most appropriate developer to fix it. In huge projects, with attention to the large number of developers and implemented changes in different files of the project, selecting an appropriate developer to fix a new bug report through the traditional bug triage system is very time-consuming and also imposes additional cost on the project [6]. For example, Eclipse has 239 active developers on January 2011 and 282 modified files on the Eclipse Platform project. Hence, many researches have been done to make the traditional bug assignment automatic [7], [6], [8], [9], [10].

Information about the activities which have been done on different parts of the project helps to make a more accurate decision about a new bug report. The information on activities performed on the project is recorded in various repositories of the project (such as bug repository and version control repository). Most of these repositories are written in textual format but in this research, we use the Natural Language Processing (NLP) and Information Extraction (IE) methods to extract information from these software repositories. The rest of this paper is organized as follows: Section Two describes the needed background for underestanding the proposed approach in this paper; Section Three contains the related works to our research and Section Four explains the suggested approach. The evaluation of each approache is described in Section Five and the conclution is presented in Section Six.

II. Background

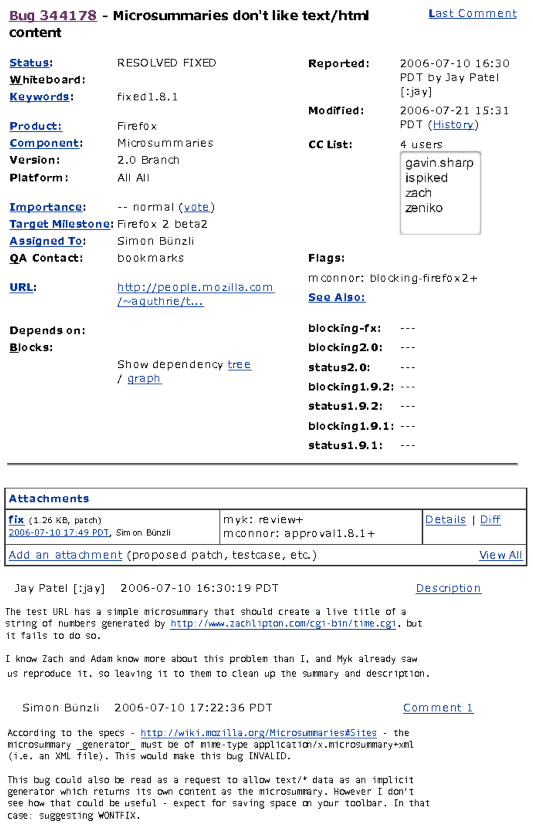
As mentioned in the introduction section, open source projects usually use bug tracking systems to accelerate and simplify the bug reporting process. The reported bugs are stored in the bug repository. There are various bug repositories which are used in open source development projects, such as Bugzilla, JIRA and GNATS. Despite some differences among these repositories, the fundamental structure of all of them is similar. To implemente our proposed approach, Bugzilla is selected as a case study. Therefore, the bug report structure which is described in this section is based on the structure of bugs in a Bugzilla repository.

A bug report contains various fields which consist of different information about the bug report and the problem or issue described in the bug report. The first category of these fields contains the predefined fields that are filled as a bug report is created. Some fields in this category are not susceptible to change during the bug life-cycle (such as ID, Reported and Product) and some others may be changed (such as Status, Assigned to, and Modified). The second category in the bug report contains textual fields (for example, Summary and Description) that describe the problem or issue. These fields have a significant role in resolving the reported problem. The last category includes the additional information about the problem, such as the files attached to the bug report that can help the fixer to understand the problem more accurately and quickly. Moreover, the other users and developers can add comments to the bug report. Fig 1 shows an instance of a bug report which has been reported to the Mozilla project.

The project developers who have permission to change the source code are able to directly check the changes in main source code and allow the other users and developers to access the last version of the source code. A commit refers to the latest submited changes of the source code and includes various information about these changes (such as developer ID, date and message). This information is recorded in the log of the Version Control System (VCS) [11]. Fig 2 shows an instance of commits which have been stored in the CVS log of Eclipse.

III. Related work

Previous approaches for improving bug report assignment fall into two main categories. The approaches in the first category use Machine Learning (ML) algorithms to recommend appropriate developers to fix a new bug report. The approaches in the second category employ a combination of Information Retrieval (IR) and ML methods.



Bug 344178 - Microsummaries don't like text/html content   Last Comment

Status:	RESOLVED FIXED	Reported:	2006-07-10 16:30 PDT by Jay Patel [:jay]
Whiteboard:			
Keywords:	fixed1.8.1	Modified:	2006-07-21 15:31 PDT (History)
Product:	Firefox	CC List:	4 users gavin.sharp ispiked zach zeniko
Component:	Microsummaries		
Version:	2.0 Branch		
Platform:	All All		
Importance:	-- normal (vote)		
Target Milestone:	Firefox 2 beta2		
Assigned To:	Simon Bünzli		
QA Contact:	bookmarks	Flags:	
			mconnor: blocking-firefox2+
URL:	http://people.mozilla.com/~aguthrie/t...	See Also:	
Depends on:		blocking-fx:	---
Blocks:		blocking2.0:	---
	Show dependency tree / graph	status2.0:	---
		blocking1.9.2:	---
		status1.9.2:	---
		blocking1.9.1:	---
		status1.9.1:	---

Attachments		
fix (1.26 KB, patch) 2006-07-10 17:49 PDT, Simon Bünzli	myk: review+ mconnor: approval1.8.1+	Details | Diff
Add an attachment (proposed patch, testcase, etc.)		View All

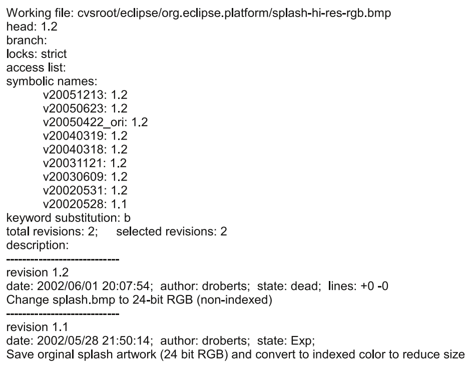
Jay Patel [:jay]   2006-07-10 16:30:19 PDT   Description

```
The test URL has a simple microsummary that should create a live title of a
string of numbers generated by http://www.zachlipton.com/cgi-bin/time.cgi, but
it fails to do so.

I know Zach and Adam know more about this problem than I, and Myk already saw
us reproduce it, so leaving it to them to clean up the summary and description.
```

Simon Bünzli   2006-07-10 17:22:36 PDT   Comment 1

```
According to the specs - http://wiki.mozilla.org/Microsummaries#Sites - the
microsummary _generator_ must be of mime-type application/x.microsummary+xml
(i.e. an XML file). This would make this bug INVALID.

This bug could also be read as a request to allow text/* data as an implicit
generator which returns its own content as the microsummary. However I don't
see how that could be useful - expect for saving space on your toolbar. In that
case: suggesting WONTFIX.
```

Fig. 1. An Instance of a reported bug in the Mozilla Project



```
Working file: cvsroot/eclipse/org.eclipse.platform/splash-hi-res-rgb.bmp
head: 1.2
branch:
locks: strict
access list:
symbolic names:
        v20051213: 1.2
        v20050623: 1.2
        v20050422_ori: 1.2
        v20040319: 1.2
        v20040318: 1.2
        v20031121: 1.2
        v20030609: 1.2
        v20020531: 1.2
        v20020528: 1.1
keyword substitution: b
total revisions: 2;     selected revisions: 2
description:
----------------------------
revision 1.2
date: 2002/06/01 20:07:54;  author: droberts;  state: dead;  lines: +0 -0
Change splash.bmp to 24-bit RGB (non-indexed)
----------------------------
revision 1.1
date: 2002/05/28 21:50:14;  author: droberts;  state: Exp;
Save orginal splash artwork (24 bit RGB) and convert to indexed color to reduce size
```

Fig. 2. An Instance of CVS log of the Eclipse project

Cubranic and Murphy [7] approached the automatic bug assignment system as a text classification problem. They used the Bayesian learning method to classify the training set and to find the similarities among these classes in order to create a new bug report. Anvik et al. [6], [12] extended the approach of Cubranic and Murphy and evaluated the performance of various ML algorithms on bug assignment problems.The results showed that Support Vector Machine (SVM) algorithm has the best performance on automatic bug assignment. Hence, They used the SVM algorithm as a text classifier. Their approach recommended a list of developers who may be able to fix the bug.

Shokripour et al. [13] used the SVM algorithm to determine the location of a new bug in the source code. The most appropriate developer to fix the bug is selected based on the activities of the developers on the determined location of the bug in the source code. Nasim et al. [14] used Alphabet Frequency Matrix (AFM) to improve the classifying of change requests based on the developers of project. They apply machine learning on classes which are achieved from AFM classification for recommending the most appropriate developers for fixing a new bug.

Bhattacharya et al. [15] used the combination of machine learning tools and probabilistic graph-based model to predicting the most appropriate developer for fixing a new bug. they investigated the results of the various classifier algorithms on different projects. Their results showed that based on the quality of the bug reports of each project, each classifier can be useful for classifying the textual data of a specific project and selecting a classifier for all projects is not possible.

Kagdi et al.[16] recommended an approach based on this rule that each developer who has most contributed changes on a file in past is most appropriate developer for fixing a new bug of that file. Thay measured three paprameters of commit contribution, the most recent activity date, and the number of active workdays for selecting the best developer for fixing a new bug of a specific file.

Canfora and Cerulo [9] used the existing information contained in the comments on the CVS and the description of the bug reports to select the most appropriate developer to fix a bug. In their approach, indexing of the developers is done by using the description of the stored bugs in the bug repository as the documents of an information retrieval system.

Ahsan et al. [17] suggested an approach based on both IR methods and the SVM algorithm. In this approach, they extract the keywords of the fixed bugs and create a Term Document Matrix (TDM). They use Term Frequency and Inverse Document Frequency (TF-IDF) for indexing. In addition, they exploit the features selection method and the Latent Semantic Indexing (LSI) method to reduce the dimension of the TDM matrix.

Baysal et al. [8] applied preference elicitation techniques to select the best developer to fix a new bug. They presented a framework for automatic bug assignment. This framework determines the levels of the expertise of the developers which they gained from previously fixing bugs. They use the vector space model to extract information from the summary, description and comments on the bugs. The TF-IDF weighting scheme is used to assign the higher weights to the most important words. Their innovative step to increase the recall of this proposed framework is using the preference elicitation methods to determine the preference of the developers. They suggest adding a new field to the bug report for determining the preference of developers about a bug after fixing it. Finally, the framework uses the information on expertise levels and the developer's preferences to assign a new bug to the most appropriate developer.

Unlike all the proposed approaches to automatic bug assignment that used the summary and description of previously fixed bug reports to extract information and make a decision about the new bug reports, Matter et al. [10] determined the expertise of the developers based on the vocabularies which have been used in source code. The extracted vocabularies from source code are compared with extracted vocabularies from a new bug report and then the appropriate developers are determined. Their approach uses information retrieval methods to weigh and determine the relationships between the extracted vocabularies.

Kagdi et al. [18], [19] used an IR-based concept location technique to recommend the most appropriate developers to fix a new change request. They used this technique to establish the relationship between the reported bug and the source code of the project. After determining the site of the bug in the source code of the project by using the concept location technique, the most appropriate developers to fix the bug are recommended based on the location of the bug in the source code.

Servant and Jones [20] recommended a automatic bug assignment system that recommends the fixer for determined fault by failing test case based on the lines of code that should to change for fixing the fault. They used the fault localization methods and history mining methods for predicting the lines code that should to change for fixing the fault and selecting the most appropriate developer who bug should to be assigned her.

Tamrawi et al. [21] presented a new approach based on fuzzy set-based modeling of bug-fixing expertise of developers. They used the extracted terms from previous bug reports which have been fixed by each developer of project for calculating the membership function of developer in fuzzy set. For a new bug report, its terms are extracted and corresponding fuzzy sets are unioned. Potential fixers will be recommended based on their membership scores in the unioned fuzzy set.

IV. The proposed method

Most of the suggested approaches to automatic bug assignment used machine learning and information retrieval techniques to recommend one or more appropriate developers to fix a new bug. Moreover, a majority of these approaches used the previously fixed bugs to extract desired information to select the fixer for a new bug. While recognizing the fact that enough instances are required to train the machine learning algorithm to learn, the number of the bugs which are selected as a training set has a significant role in the accuracy of the approaches. Hence, the accuracy of the methods which used ML methods depends on the number of the previously fixed bugs in a project. Therefore, these approaches did not give good results on projects which have a few number of fixed bugs. For example, Anvik et al. in [12] mentioned that the GCC project was too small and that was one of the reasons which caused the poor results of their suggested approach to the GCC project.

The goal of this research is to use NLP and IE methods to extract suitable information in order to make decisions about

TABLE I
SAMPLE OF COMMON PHRASES BETWEEN THE BUG REPORTS AND COMMITS

Bug ID	Common phrase	Commit File	Revision
143073	Occurrences in File	org.eclipse.jdt.ui/ui/org/eclipse /jdt/internal/ui/actions/ OccurrencesSearchMenu-Action.java	1.2
137091	Occurrences shortcut	org.eclipse.jdt.ui/ui/org/eclipse /jdt/ internal/ui/javaeditor/ BasicJavaEditorActionCon-tributor.java	1.32

the new bug reports. In addition, the proposed method in this paper uses the existing information contained in the messages of the commits. Our approach does not depend on the number of the previously fixed bugs and can be used on small and new projects.

A study of the bug reports and commits showed that there are many similar phrases in the bug reports and messages which were written for each commit of the project. Some of these phrases can be used to link a bug report to a specific commit. We use this fact to recommend a developer to fix a bug.

There are too many similarities between the existing phrases in the bug reports and the messages of the commits. We use the term 'phrase composition' to refer to the combination of the phrase plus the phrases which are concatenated to each other by a preposition. For example, the phrase "preference page code snippet" plus the concatenated phrase "for options related to type arguments" gives "preference page code snippet for options related to type arguments" which is referred to, in this paper, as a 'phrase composition'. We use the phrase composition, to extract the relationships between bug reports and commits. Therefore, the entities that we should extract in this research are all existing phrase compositions in the bug reports and messages of the commits. The phrase compositions help to determine the relationships accurately. For example, the first and second rows of TableI show two different phrase compositions in which the first words are similar but the phrases belong to two different relationships between bugs and commits.

In this research, the appropriate developers to fix a new bug are selected based on the location of the bug. In the first step, the location of the bug (the file which should be changed to fix the bug) is determined, and in the second step the developer is recommended based on the activities which have been done on the suggested file as the location of bug.

A. Using Automatic Bug Localization

When a developer checks the changes in the version control repository, the useful information in commit (for example, developer ID, the changed file(s), date of commit and message of commit) is stored in the Log of the version control repository. In this research, we use the messages of commits to determine the location of new bugs. The process of determining the location of a new bug involves the following steps:

- Move all commits data from VCS log to the database.
- Determine the phrase compositions which have been used in comments of the commits.
- Extract the phrase compositions which have been used in Summary and Description fields of the bug report.
- Determine the similarity between the phrase compositions of the commits and the bug report.
- Recommend the files of the commits which have the most similar phrase compositions with bug report as location of the bug.

Some of these steps are described in the following pages.

1) Extract phrase compositions from comments: In this step, we need a method that has the ability of identifying the phrase compositions from the text documents. Information Extraction technique provides this facility in Named Entity Recognition (NER). If the phrase compositions are considered as the entities of the documents, we can extract them by using NER methods. These methods use Natural Language Processing technique (NLP) to decompose the text to its elements and then use the results of the NLP to extract the desired information. There are some frameworks that provide tools for NLP and IE methods (such as UIMA[1], GATE [2], OpenNLP[3] and INTEX/UNITEX/NooJ[4]). In this research, we use Gate framework which provides various tools for NLP and IE methods.

To extract the desired entity from a text, the text should first be decomposed. In this research, the ANNIE Sentence Splitter is used to split the sentences of the text and then the sentences are further split into tokens, such as numbers and punctuation, by using Annie English Tokeniser. In the third step, the tokens are labeled with tags of the part-of-speech. In this research, ANNIE POS Tagger is used to assign the part-of-speech tags to the elements of the sentences. These steps are known as preprocessing steps [22] which prepare the text to extract information by IE methods. The goal of this research is to extract the phrase compositions as the entities of the text. Hence, we use NER methods to extract these entities from messages of commits and bug reports. In this step, we use ANNIE NE Transducer which is one of the rule-based tools of NER. This tool uses Java Annotation Patterns Engine (JAPE) language to define the templates based on which the entities should be extracted. The desired entities in this research are the phrase compositions and the phrases which are located between two quotations. We name these phrases "quotation-phrase". For example, in the Eclipse case study, "link" appeared in the messages of 301 commits, but "link with editor" only appeared in five commits. Thus, when "with editor" is added to "link", it changes this word to a special phrase composition which helps us to determine the related commits to the bug into which this composition was used. The aim of this part of the

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

[2] http://gate.ac.uk/

[3] http://incubator.apache.org/opennlp/index.html

[4] http://intex.univ-fcomte.fr/

research is to determine the location of the bug in the project. In other words, we want to determine the files that should be changed to fix the bug. The file in which changes were done by commit is recorded in the log. Hence, the extracted phrase compositions from the message of the commit can be considered as phrase compositions of commit's file. Indeed, we determine the similarity between the phrase compositions of the bugs and files of the commits. Furthermore, we extract the phrases which are located between two quotations (quotation-phrase). This is because developers and users usually put the phrase between two quotations to emphasise on a phrase which has special importance. Therefore, in this step, we extract all phrase compositions and quotation-phrases for all files of the project.

2) Link a new bug report to appropriate file: After extracting the phrase compositions and quotation-phrases from messages, the same process should be done on a new bug report for which we want to recommend the most appropriate developer to fix. The extracted phrase compositions from commits and the new bug report are compared and the amount of similarities between each file of the project and the bug was determined. The files that have more similar phrase compositions with the new bug report are recommended as the more probable locations of the bug. The quotation-phrases have a significant role in determining the similarities between files and bugs. For example, using the quotation-phrases improved the accuracy to 17% on the Mozilla project. Hence, unlike each common phrase composition between file and bug which is counted as one relationship, each common quotation-phrase is counted as two relationships between bug and file. The investigation of recommending different number of files for a new bug report (as predicted locations of bug) showed that there are three cases for recommended files:

1) The recommended file is the actual file of the bug.
2) The recommended file is indirectly related to the actual file of the bug.
3) The recommended file does not have any relationship with the actual file of the bug.

The relationship between the first and third cases with desired the results of predicting the location of the bug is obvious. Investigation of the recommended files for the bug reports showed that many of the recommended files have common commits with the actual files of the bugs. It means that in the past, a developer worked on the recommended file and the actual file of the bug at the same time. Therefore, the recommended file and the actual file of the bug have a relationship with each other. The minimum relationship between these files which have a common commit is that these files are located in a same path of the project and some of them are located in a same package of the project. For example, in recommending three files for each bug of the Eclipse project, 56% of the recommended files had common commits with actual files of the bugs and 43% of these files were located in a same path of the actual files of the bugs. This relationship helps to recommend the most appropriate developer to fix a bug. For example, in 40% of recommended files which have common commits with actual files of the bugs, the developers of the recommended files are actual fixers of bugs.

B. Developer Recommendation

In this step, we recommend a developer based on the recommended files (predicted locations of bug) for the bug report. The most appropriate developer to fix a bug is selected according to the activity histories of developers on files of the project [13]. In our research, the activities of the developers on projects are divided into two categories: (1) development activities, and (2) bug-fixing activities. Development activities are all activities which have been done to develop the project (development-based), and bug-fixing activity histories only contain the bug-fixing activities (bug-fixing-based). In this research, we want to select a fixer for the bugs, so bug-fixing activities have a significant influence over decision-making. Hence, the first category contains all the activities of the developers (including development and bug-fixing activities) and the second category only contains the bug-fixing activities. We extracted the development and bug-fixing activity histories and used these histories to recommend the fixer to fix the new bugs.

The second important factor in creating the activity histories is the criterion of selecting the activities of the history. This criterion can be a specific number of the activities or the activities of a specific period time [12]. There is the probability that in different period times, developers change their activities. For example, a developer changes the component on which he/she has worked until now or does not continue his/her contribution to the project. Selecting a suitable number of activities which contains enough information to make decisions about the activities of the developers is a complex process and needs many trials and errors to find the best number of activities. Therefore, our criterion for selecting the developer activities is the time interval. Investigation of different time intervals on different projects showed that the interval of eight months is the most appropriate interval to create activity histories for each bug report. Therefore, to create activity histories for each new bug we use the activities which were done in the last eight months from the bug report date. Fig 3 shows the relationships of different intervals and achieved accuracies on the Eclipse project.

V. Evaluation

To evaluate the performance of our approach we use the standard measure of recall. Recall measures how many of the developers who may be appropriate to resolve the problem are actually recommended.

We evaluated our suggested approach on three favorite open source projects: Eclipse, Mozilla and Gnome. We used the CVS commits which had been checked in the version control repositories of the Eclipse and Mozilla projects before December 2006 and January 2007 respectively and we also used SVN commits of the Gnome project which had been checked in before October 2008. We did not use all existing commits

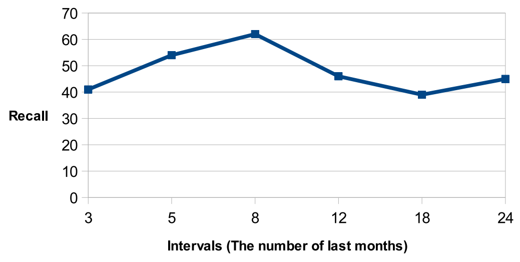


Fig. 3. The achieved accuracies from testing different intervals to create activity histories

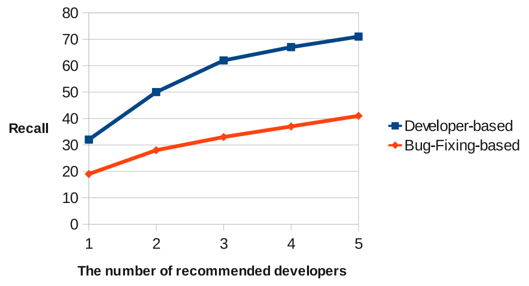


Fig. 4. Achieved accuracies on Eclipse project on recommending different number of developers for Development-based and Bug-Fixing-based cases

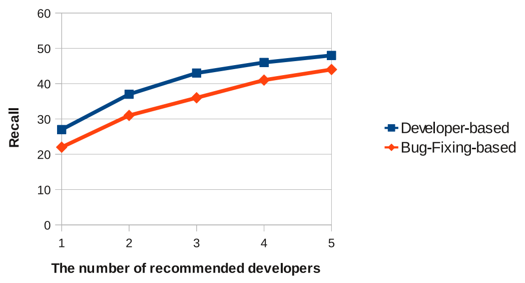


Fig. 5. Achieved accuracies on Mozilla project on recommending different number of developers for Development-based and Bug-Fixing-based cases

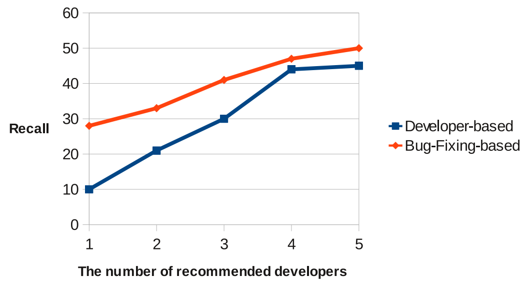


Fig. 6. Achieved accuracies on Gnome project on recommending different number of developers for Development-based and Bug-Fixing-based cases

in repositories because we should analyze the commits of each file of the project from the time of creation of the file. Hence, to prevent the problem of having too huge a data set, we did not use all existing commits in the repositories. We used 35140, 9917 and 119176 commit logs to evaluate our approach on Eclipse, Mozilla and Gnome projects, respectively.

Some of the extracted phrase compositions from messages of the commits are not useful data and they lessen the accuracy of results. For example, ”mozilla” is a word which is repeated in many commits and bugs of the Mozilla project. Hence, this word does not help to link a new bug to files of the project. In order to improve the results, these phrases and phrase compositions should be removed from the extracted data.

As mentioned in the previous sections, we used the phrase compositions which had been used in the messages of commits to determine the location of the bugs and select the developer to fix them. In this research, the number of recommended files is equal to the number of recommended developers because the most active developer of the file is the recommended developer for the bug. We evaluated recall for recommending different numbers of developers to fix a new bug report on the Eclipse, Mozilla and Gnome projects (Fig. 4, Fig. 5 and Fig. 6).

The evaluation of using development and bug-fixing activity histories on the Eclipse, Mozilla and Gnome projects showed that using development history on the Eclipse and Mozilla projects leads to better results than using bug-fixing activity histories (Fig 4 and Fig 5), and on the Gnome project using bug-fixing activity histories is more appropriate (Fig 6).

VI. Conclusion

In this paper, we present a new approach to automatic bug assignment based on Information Extraction methods. Unlike most previous suggested methods, the presented approach in this paper only uses the messages of commits to recommend the most appropriate developer to fix a new bug. Therefore, it does not have the limitation of the previous approaches which need a minimum number of fixed bugs to train the ML methods. Our approach uses rule-based entity extraction method to identify the phrases which help to determine the location of the bugs in a project and then, according to the recommended location for the bug, the best developers to fix the bug are suggested. The main aim of this research is to recommend the most appropriate developer to fix a new bug (actual fixer) and according to this goal, we reached a recall level of 62%, 43% and 41% on the Eclipse, Mozilla and Gnome projects respectively when we recommended three developers for each bug.

References

[1] K. Crowston, J. Howison, and H. Annabi, “Information systems success in free and open source software development: theory and measures,” *Software Process: Improvement and Practice*, vol. 11, no. 2, pp. 123–148, 2006. [Online]. Available: http://dx.doi.org/10.1002/spip.259

[2] G. Jeong, S. Kim, and T. Zimmermann, “Improving bug triage with bug tossing graphs,” in *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ser. ESEC/FSE ’09. New York, NY, USA: ACM, 2009, pp. 111–120. [Online]. Available: http://doi.acm.org/10.1145/1595696.1595715

[3] P. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in *ICSE ’07: Proceedings of the 29th international conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 499–510.

[4] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, “An approach to detecting duplicate bug reports using natural language and execution information,” in *ICSE ’08: Proceedings of the 30th international conference on Software engineering*. New York, NY, USA: ACM, 2008, pp. 461–470.

[5] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, “What makes a good bug report?” *IEEE Trans. Softw. Eng.*, vol. 36, pp. 618–643, September 2010. [Online]. Available: http://dx.doi.org/10.1109/TSE.2010.63

[6] J. Anvik and G. C. Murphy, “Reducing the effort of bug report triage: Recommenders for development-oriented decisions,” *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 10:1–10:35, Aug. 2011. [Online]. Available: http://doi.acm.org/10.1145/2000791.2000794

[7] D. Cubranic and G. C. Murphy, “Automatic bug triage using text categorization.” in *SEKE’04*, 2004, pp. 92–97.

[8] O. Baysal, M. Godfrey, and R. Cohen, “A bug you like: A framework for automated assignment of bugs,” in *Program Comprehension, 2009. ICPC ’09. IEEE 17th International Conference on*, May 2009, pp. 297 –298.

[9] G. Canfora and L. Cerulo, “How software repositories can help in resolving a new change request,” in *In Workshop on Empirical Studies in Reverse Engineering*, 2005.

[10] D. Matter, A. Kuhn, and O. Nierstrasz, “Assigning bug reports using a vocabulary-based expertise model of developers,” in *Mining Software Repositories, 2009. MSR ’09. 6th IEEE International Working Conference on*, May 2009, pp. 131 –140.

[11] M. Fischer, M. Pinzger, and H. Gall, “Populating a release history database from version control and bug tracking systems,” in *Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on*, 2003, pp. 23 – 32.

[12] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?” in *ICSE ’06: Proceedings of the 28th international conference on Software engineering*. New York, NY, USA: ACM, 2006, pp. 361–370.

[13] R. Shokripour, M. Khansari, and Z. M. Kasirun, “Automatic bug assignment using history of packages,” in *ICSIE 2011: Proceedings of the 2011 International Conference on Software and Information Engineering*. Kuala Lumpur, Malaysia: ASME, June 2011, accepted.

[14] S. Nasim, S. Razzaq, and J. Ferzund, “Automated change request triage using alpha frequency matrix,” in *Proc. Frontiers of Information Technology (FIT)*, 2011, pp. 298–302.

[15] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, “Automated, highly-accurate, bug assignment using machine learning and tossing graphs,” *J. Syst. Softw.*, vol. 85, no. 10, pp. 2275–2292, Oct. 2012. [Online]. Available: http://dx.doi.org/10.1016/j.jss.2012.04.053

[16] H. Kagdi, M. Hammad, and J. Maletic, “Who can help me with this source code change?” in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*, 28 2008-oct. 4 2008, pp. 157 –166.

[17] S. Ahsan, J. Ferzund, and F. Wotawa, “Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine,” in *Software Engineering Advances, 2009. ICSEA ’09. Fourth International Conference on*, 2009, pp. 216 –221.

[18] H. Kagdi and D. Poshyvanyk, “Who can help me with this change request?” in *Program Comprehension, 2009. ICPC ’09. IEEE 17th International Conference on*, may 2009, pp. 273 –277.

[19] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, “Assigning change requests to software developers,” *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[20] F. Servant and J. A. Jones, “Whosefault: automatic developer-to-fault assignment through fault localization,” in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012. Piscataway, NJ, USA: IEEE Press, 2012, pp. 36–46. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337228

[21] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, “Fuzzy set-based automatic bug triaging: Nier track,” in *Proc. 33rd Int Software Engineering (ICSE) Conf*, 2011, pp. 884–887.

[22] S. Sarawagi, “Information extraction,” *Found. Trends databases*, vol. 1, pp. 261–377, March 2008. [Online]. Available: http://portal.acm.org/citation.cfm?id=1498844.1498845