BOAT: An Experimental Platform for Researchers to Comparatively and Reproducibly Evaluate Bug Localization Techniques

Xinyu Wang[1], David Lo[2], Xin Xia[1], Xingen Wang[1], Pavneet Singh Kochhar[2], Yuan Tian[2]
Xiaohu Yang[1], Shanping Li[1], Jianling Sun[1], and Bo Zhou[1]
[1]College of Computer Science and Technology, Zhejiang University, China
[2]School of Information Systems, Singapore Management University, Singapore
{wangxinyu,xxkidd,newroot,yangxh,shan,sunjl,bzhou}@zju.edu.cn,
{davidlo,kochharps.2012,yuan.tian.2012}@smu.edu.sg

ABSTRACT

Bug localization refers to the process of identifying source code files that contain defects from descriptions of these defects which are typically contained in bug reports. There have been many bug localization techniques proposed in the literature. However, often it is hard to compare these techniques since different evaluation datasets are used. At times the datasets are not made publicly available and thus it is difficult to reproduce reported results. Furthermore, some techniques are only evaluated on small datasets and thus it is not clear whether the results are generalizable. Thus, there is a need for a platform that allows various techniques to be compared with one another on a common pool containing a large number of bug reports with known defective source code files. In this paper, we address this need by proposing our <u>B</u>ug l<u>O</u>calization experimental pl<u>AT</u>form (*BOAT*). BOAT is an extensible web application that contains thousands of bug reports with known defective source code files. Researchers can create accounts in BOAT, upload executables of their bug localization techniques, and see how these techniques perform in comparison with techniques uploaded by other researchers, with respect to some standard evaluation measures. BOAT is already preloaded with several bug localization techniques and thus researchers can directly compare their newly proposed techniques against these existing techniques. BOAT has been made available online since October 2013, and researchers could access the platform at: `http://www.vlis.zju.edu.cn/blp`.

Categories and Subject Descriptors

D.2.5 [**Software Engineering**]: Testing and Debugging

ICSE Companion'14, May 31 – June 7, 2014, Hyderabad, India
Copyright 2014 ACM 978-1-4503-2768-8/14/05...$15.00
http://dx.doi.org/10.1145/2591062.2591066

General Terms

Experimentation

Keywords

Bug Localization, Benchmark, Experimental Platform, BOAT

1. INTRODUCTION

Bugs are inevitable during software development and maintenance. To improve the reliability of software systems, many software projects allow end users to report bugs that they encounter. For a large software system, the project team can receive hundreds of bug reports every day [1]. Once a bug report is received, a developer often needs to spend much effort to locate the defective program code and fix it. To reduce this effort, a number of techniques have been proposed to automatically return program files that are likely to contain defects given a bug report [8, 3, 6].

Despite advances in bug localization studies, there are a few challenges that potentially hamper the development of this field:

1. Many bug localization techniques are tested on different datasets and thus it is hard to compare their effectiveness.
2. To make matter worse, at times the datasets used are not publicly released, e.g., [8], and thus it is hard to replicate the results of the study.
3. Furthermore, a number of techniques have only been tested on small number of bug reports (less than 50), e.g., [5]. Thus it is not clear if the results reported for these techniques are generalizable across a wide variety of projects and bug reports.

Based on these observations, there is a need for a platform that researchers can use to compare various techniques, replicate the results of various techniques, and reduce the threats to external validity. In this work, we provide such platform named <u>B</u>ug l<u>O</u>calization experimental pl<u>AT</u>form (*BOAT*). This web-based platform allows researchers to upload their bug localization techniques, check the effectiveness of their techniques on a large bug report dataset, and compare the effectiveness of their techniques with other techniques. Currently, *BOAT* contains thousands of bug reports and the implementations of several popular bug localization approaches.

In the remainder of this paper, we first describe the details of our platform in Section 2. We describe the datasets contained and techniques implemented in the current version of *BOAT* in Section 3. Next, we describe related work in Section 4. We conclude and describe future work in Section 5.

2. *BOAT* PLATFORM

Figure 1 presents the architecture of *BOAT*. *BOAT* contains 3 components: data collection component, local debugging component, and remote execution component. The data collection component analyzes software projects to find bug reports, the code before the corresponding bugs are fixed, and the corresponding buggy files that get fixed. The local debugging component supports researchers during the development of *BOAT*-compatible executables of their bug localization techniques in their own local environments. Its goal is to reduce the debugging effort when researchers eventually upload their executable files in *BOAT*. The remote execution component runs the uploaded executables of bug localization techniques against the dataset that are extracted by the data collection component and returns the results to developers. In the following paragraphs, we describe the details of these 3 components.

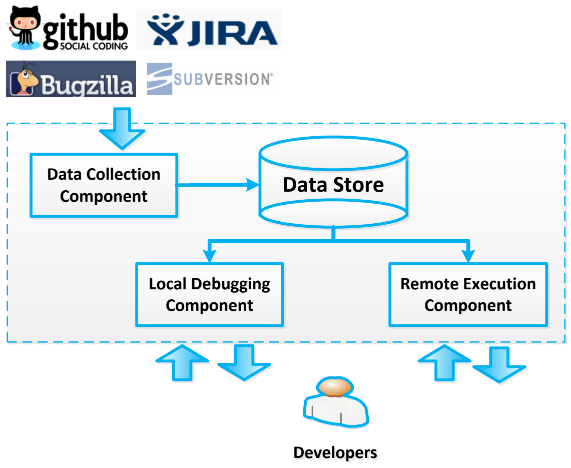

github SOCIAL CODING
JIRA
Bugzilla
SUBVERSION
Data Collection Component
Data Store
Local Debugging Component
Remote Execution Component
Developers

Figure 1: The Architecture of *BOAT*.

The data collection component processes software projects and for each project it analyzes its bug tracking system (e.g., JIRA, Bugzilla, etc.) and version control system (e.g., git, SVN, etc.). *BOAT* extracts commit logs from the version control system. For each log, it analyzes whether the corresponding commit fixes a bug report. BOAT performs regular expression checks to identify whether a bug report identifier exists in the log. If there is an identifier, *BOAT* would get the details of the bug (i.e., a bug report) with that identifier from the bug tracking system. *BOAT* would group commits that fix a defect reported in the same bug report. From these pieces of information, for each bug report whose identifier exists in the commit logs, *BOAT* recovers the code before the fix, and the files that are changed or deleted to fix the bug. These extracted data is stored in a database to be used by the other components of *BOAT*. Note that there is no need to process a huge number of projects at once. *BOAT*'s administrators can incrementally add more bug reports and their corresponding buggy files. Thus, over time *BOAT* would contain more and more reports.

```
1: public static void main(String[] args) {
2:   if (args.length < 4) {
3:     System.err.println("Invalid options.");
4:     System.err.println("<url> <user> <password>
   <project root>");
5:     System.exit(2);
6:     }
7:   String url = args[0];
8:   String user = args[1];
9:   String password = args[2];
10:   String project = args[3];
11:   BugLocationService service= setupService(url, user,
   password);
12:   for (String bug :  service.listBugs(project)) {
13:     Map<String, Object> objMap = (Map<String, Objec-
   t>) service.useBug(bug);
14:     List<String> relevantFileLists = ser-
   vice.getBugLocation(bug);
15:     List<String> fileList = predictBugs(objMap, new
   File(project));
16:     ...
17:     }
18: }
```

Figure 2: Sample Java Code for Local Debugging.

Using the local debugging component, researchers can interact with *BOAT* in their own local environments. *BOAT* allows registered researchers to fetch a limited number of bug reports (with the contents of all their fields), the code before these bugs are fixed, and the defective source code files via a restful web service. This web service can be accessed by programs written in many programming languages and thus *BOAT* does not put much restriction in the environment in which researchers can develop their own bug localization techniques. Figure 2 presents an example of Java code that interacts with *BOAT*'s local debugging component. At Line 11, the code creates a connection to *BOAT*'s local debugging component. At Lines 12-14, the code iterates through the bug reports, code before the bugs are fixed, and the corresponding buggy files that *BOAT*'s local debugging component allows researchers to fetch.

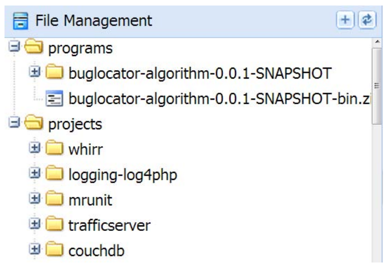

File Management
programs
buglocator-algorithm-0.0.1-SNAPSHOT
buglocator-algorithm-0.0.1-SNAPSHOT-bin.z
projects
whirr
logging-log4php
mrunit
trafficserver
couchdb

Figure 3: An Example Working Directory.

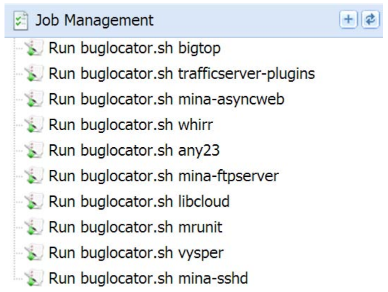
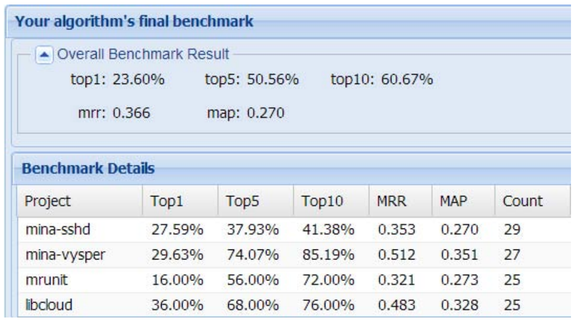

Job Management
Run buglocator.sh bigtop
Run buglocator.sh trafficserver-plugins
Run buglocator.sh mina-asyncweb
Run buglocator.sh whirr
Run buglocator.sh any23
Run buglocator.sh mina-ftpserver
Run buglocator.sh libcloud
Run buglocator.sh mrunit
Run buglocator.sh vysper
Run buglocator.sh mina-sshd

Figure 4: An Example Configuration.

After a researcher completes the development of a bug localization technique, he/she can upload the executable of the technique to *BOAT* via *BOAT*'s web interface. Each registered researcher is given a working directory which contains executables that he/she uploads and data collected by *BOAT*'s data collection component. An example of a working directory is shown in Figure 3. In the figure, under "projects" subfolder is collected bug reports, and under "programs" subfolder is uploaded executables. Researcher can then set some configuration options that would govern how *BOAT* runs the uploaded executables. Figure 4 presents an example of a configuration setting. In this configuration, several jobs are specified. Each job specifies a bug localization algorithm that would be run and a software project whose bug reports would be used to evaluate the algorithm. After a configuration is specified, the remote execution component would then run selected executables against bug reports from selected projects. An email would be sent to the researcher to inform whether each of the jobs is successful or not. If a job fails, the researcher could find a detailed information of the failure in an execution log file that is sent with the email. If a job is successful, the remote execution component evaluates the ranked lists of potentially buggy files returned by the bug localization algorithm in terms of standard measures such as: recall-rate@1[1] (top1), recall-rate@5 (top5), recall-rate@10 (top10), mean reciprocal rank (mrr) and mean average precision (map) – c.f., [8, 3, 6]. The experiment results are displayed in the researcher's dashboard in *BOAT*'s web interface. Figure 5 shows an example experiment results that is displayed in a researcher's dashboard.


Your algorithm's final benchmark

Overall Benchmark Result

top1: 23.60% top5: 50.56% top10: 60.67%

mrr: 0.366 map: 0.270

Benchmark Details

Project	Top1	Top5	Top10	MRR	MAP	Count
mina-sshd	27.59%	37.93%	41.38%	0.353	0.270	29
mina-vysper	29.63%	74.07%	85.19%	0.512	0.351	27
mrunit	16.00%	56.00%	72.00%	0.321	0.273	25
libcloud	36.00%	68.00%	76.00%	0.483	0.328	25

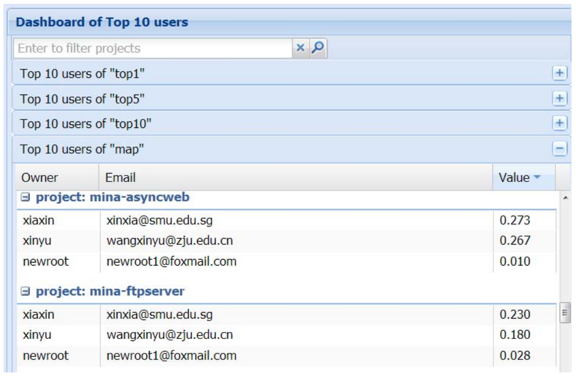
Figure 5: An Example Experimental Results.

After a successful completion of a job, the remote execution component also computes/updates a list of top-10 researchers with best performing bug localization techniques. This list would be made available in the dashboards of registered researchers. With this list, researchers can compare the performances of various bug localization techniques on various datasets. Figure 6 shows a snapshot of *B*OAT's interface which shows lists of top-10 researchers (users) for various datasets and various evaluation metrics (top1, top5, top10, mrr, or map).

3. CURRENT DATASET AND TECHNIQUES

BOAT currently contains 6,080 bug reports and their corresponding fixes from 29 open-source projects. To ensure the quality of the data, we have manually sampled a subset of the bug reports (i.e., 300 bugs), and checked their corresponding fixes. We find that these bug reports are of good quality. Table 1 presents the statistics of bug reports collected from the projects. The columns correspond to the project name (Project), the number of bug reports (# Bugs), the number of source code files in the projects (# Files), and the time period of these collected bug reports (Time Period).

[1]It is also known as top n rank in [8]


Dashboard of Top 10 users

Top 10 users of "top1"

Top 10 users of "top5"

Top 10 users of "top10"

Top 10 users of "map"

Owner	Email	Value
project: mina-asyncweb		
xiaxin	xinxia@smu.edu.sg	0.273
xinyu	wangxinyu@zju.edu.cn	0.267
newroot	newroot1@foxmail.com	0.010
project: mina-ftpserver		
xiaxin	xinxia@smu.edu.sg	0.230
xinyu	wangxinyu@zju.edu.cn	0.180
newroot	newroot1@foxmail.com	0.028

Figure 6: An Example Top 10 Users.

Table 1: Statistics of Collected Datasets.

Project	# Bugs	# Files	Time Period
accumulo	181	1,376	2011.10 -2013.09
Activemq	175	1,580	2005.04 - 2007.12
any23	10	660	2012.01 - 2013.08
Bigtop	31	483	2011.08 - 2013.09
Camel	1,192	830	2007.08 - 2008.11
Couchdb	223	137	2008.03 - 2013.08
Crunch	92	292	2012.07 - 2013.02
deltaspike	41	135	2012.03 - 2013.05
Flume	281	557	2010.01 - 2012.05
Giraph	192	95	2011.08 - 2012.07
Isis	37	6,643	2010.12 - 2013.09
Kafka	209	428	2011.08 - 2013.08
Libcloud	25	117	2010.05 - 2013.08
logging-log4php	50	117	2010.05 - 2013.09
lucenenet	156	560	2007.09 - 2008.10
Mina	149	263	2005.12 - 2006.09
mina-asyncweb	4	218	2008.05 - 2009.01
mina-ftpserver	18	326	2008.01 - 2012.05
mina-sshd	29	137	2008.12 - 2011.05
mina-vvsper	27	308	2008.01 - 2013.01
Mrunit	25	44	2011.05 - 2013.04
Ode	145	1,725	2006.08 - 2013.09
Sgoop	96	405	2011.09 - 2013.09
tapestry-5	421	2,358	2008.09 - 2010.03
Thrift	331	596	2010.12 - 2011.05
trafficserver	551	1,775	2009.11 - 2012.01
trafficserver-plu	1	137	2012.02 - 2012.02
Whirr	8	159	2010.11 - 2012.08
Wicket	1,380	3,325	2007.09 - 2007.10

We have used *BOAT* to evaluate a popular and a state-of-the-art bug-localization technique, namely VSM proposed by Marcus et al. [4], and BugLocator proposed by Zhou et al. [8]. *BOAT* runs these two techniques on the bug reports from various projects. Table 2 shows the recall-rate@1 (top1), recall-rate@5 (top5), recall-rate@10 (top10), mrr and map of the two techniques for several projects as evaluated by *BOAT*. The implementations of these two techniques are made available in *BOAT* and researchers can easily compare the effectiveness of their techniques against these two techniques.

4. RELATED WORK

In this section, we first compare and contrast BOAT with a related experimental platform and a related benchmark.

We then briefly describe some existing studies on bug localization.

Table 2: Partial Results of VSM (V.) and BugLocator (B.) Tested using BOAT.

Project	Algo.	top1	top5	top10	MRR	MAP
couchdb	V.	0.19	0.45	0.61	0.321	0.243
	B.	0.24	0.54	0.69	0.377	0.297
Crunch	V.	0.10	0.36	0.51	0.228	0.139
	B.	0.15	0.42	0.55	0.280	0.195
flume	V.	0.17	0.44	0.59	0.303	0.238
	B.	0.21	0.52	0.67	0.360	0.282
giraph	V.	0.17	0.55	0.73	0.338	0.199
	B.	0.24	0.55	0.76	0.375	0.233
libcloud	V.	0.24	0.56	0.68	0.401	0.276
	B.	0.36	0.68	0.76	0.483	0.328
thrift	V.	0.09	0.30	0.41	0.195	0.146
	B.	0.12	0.39	0.50	0.250	0.203

TraceLab proposed by Keenan et al. [2], is a traceability experimental platform where researchers could design and execute experiments in its visual modeling environment. The following are several differences between BOAT and TraceLab:

1. **Target Problem and Datasets**: TraceLab is a platform for generic traceability recovery focusing on linking requirements to program code. Datasets that come with TraceLab (e.g., eAnci, EasyClinic, eTour, and SMOS) are small-scale datasets containing textual requirements and links between these requirements to program code. On the other hand, BOAT focuses on bug localization, and we collect thousands of bug reports from 29 projects and their fixes.
2. **Computing Resource Support**: To use TraceLab, researchers need to download and install it in their machine and use their own computing resources. On the other hand, BOAT provides computing resources for researchers and manages the execution of bug localization techniques uploaded by researchers.
3. **Intrinsic Support for Comparative Evaluation**: Researchers need to contribute to TraceLab to allow other techniques to compare with their proposed techniques. On the other hand, BOAT automatically compares and contrasts the various techniques that are uploaded by researchers and presents the top-10 registered researchers with the best techniques to all registered researchers.

To fill the need for a large scale feature location benchmark, Xing et al. propose the use of a Linux kernel based benchmark [7]. The following are several differences between our work and Xing et al.'s work:

1. **Experimental Platform versus Benchmark**: Different from Xing et al's work, we not only provide a benchmark, but also a platform to support researchers to comparatively and reproducibly evaluate bug localization algorithms.
2. **Datasets**: The Linux kernel dataset proposed by Xing et al. contains links between textual descriptions of Linux kernel features and code. Our datasets are bug reports and their fixes. To reduce the threat to external validity, our dataset does not come from only one software project, rather from 29 software projects.

There are a number of bug localization techniques. Poshyvanyk et al. propose a bug localization technique named PROMESIR, which uses Latent Semantic Indexing (LSI) and a probabilistic ranking technique to rank source code files [5]. Lukins et al. locate defective source code files by leveraging Latent Dirichlet Allocation (LDA) [3]. Rao and Kak perform a comparative study of different bug localization techniques, such as Unigram Model (UM), Vector Space Model (VSM), LSI, LDA, and Cluster-Based Document Model (CBDM). They conclude that UM and VSM achieve the best performance. Zhou et al. propose BugLocator which ranks source code files based on textual similarity between bug reports and files using revised Vector Space Model (rVSM), and also based on similar bug reports which are fixed before [8]. Our experimental framework makes it easier for researchers to compare bug localization techniques over a large number of bug reports and thus supports future research in this area.

5. CONCLUSION AND FUTURE WORK

In this paper, we describe *BOAT* which is a web-based experimental platform to enable researchers to comparatively and reproducibly evaluate the performance of bug localization techniques over a large dataset of bug reports. *BOAT* currently contains 6,080 bug reports from 29 projects and the implementations of a popular and a state-of-the-art bug localization technique. *BOAT* can be publicly accessed from: `http://www.vlis.zju.edu.cn/blp`. In the future, we plan to add more datasets into *BOAT*, implement more bug localization algorithms, integrate more evaluation metrics, support bug localization at additional levels of granularity (e.g., method or basic block granularity), and organize a bug localization competition using *BOAT*.

Acknowledgements. This research is sponsored in part by NSFC Program (No.61103032) and National Key Technology R&D Program of the Ministry of Science and Technology of China (2014BAH24F02).

6. REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy. Who should fix this bug? In *ICSE*, 2006.

[2] E. Keenan, A. Czauderna, G. Leach, J. Cleland-Huang, Y. Shin, E. Moritz, M. Gethers, D. Poshyvanyk, J. Maletic, J. Huffman Hayes, et al. Tracelab: An experimental workbench for equipping researchers to innovate, synthesize, and comparatively evaluate traceability solutions. In *ICSE*, 2012.

[3] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn. Bug localization using latent dirichlet allocation. *Information and Software Technology*, 2010.

[4] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *WCRE*, 2004.

[5] D. Poshyvanyk, Y.-G. Guéhéneuc, A. Marcus, G. Antoniol, and V. Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *TSE*, 2007.

[6] S. Rao and A. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *MSR*, 2011.

[7] Z. Xing, Y. Xue, and S. Jarzabek. A large scale linux-kernel based benchmark for feature location research. In *ICSE*, 2013.

[8] J. Zhou, H. Zhang, and D. Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *ICSE*, 2012.