Automated Bug Management: Reflections & the Road Ahead

David Lo
Self-Introduction
Self-Introduction
Self-Introduction
Singapore Management University

- Third university in Singapore
- Number of students:
  - 9000+ (UG)
  - 2000+ (PG)
- Schools:
  - Computing & IS
  - Economics
  - Law
  - Business
  - Accountancy
  - Social Science
School of Computing and Information Systems

- Undergraduates: 1800+
- Master students: 500+
- Doctoral students: 70+
RISE Center (2023-2028)

Center for Research on Intelligent Software Engineering (RISE)
10 faculty members, 40+ staffs and students

**AI4SE**: AI for SE (*)
**SE4AI**: SE for AI (*)
**SE4ET**: SE for Emerging Tech (+)
**SSE**: Sustainable SE (#)
**SEPE**: SE Practice Excellence (#)

(*) Established
(+) Growing
(#) To be developed
SOftware Analytics Research (SOAR) Group

https://soarsmu.github.io/
SSoftware Analytics Research (SOAR) Group
SOftware Analytics Research (SOAR) Group

- Code Summarization
- Smart Contract Analysis
- Defect Prediction
- Empirical Studies
- Automated Test Generation
- Bug Finding & Repair
- Bug Report Analysis
- Vulnerability Discovery and Repair
- Usable Security
- AI Fairness
- AI System Engineering
- AI Testing
- Question Answering
- Pattern / Specification Mining
- Recommender Systems
- Information Linking
- Open Source Security Risk
- Binary Code Analysis
- Android Permission and Sandboxing
- Android Side Channel Analysis
- Usable Security
Automated Bug Management: Reflections & the Road Ahead

David Lo
“There are two ways to write error-free programs; only the third works.”
-- Alan J. Perlis

“Up to a point, it is better to just let the snags [bugs] be there than to spend such time in design that there are none.”
-- Alan M. Turing
Bugs and Economy

- Software bugs are prevalent
- Although most bugs are less “harmful”, many have serious impact
  - Collectively responsible for trillions of dollars

Study: Buggy software costs users, vendors nearly $60B annually

By Patrick Thibodeau
Senior Editor, Computerworld | JUN 25, 2002 12:00 AM PST

Our 2022 update report estimates that the cost of poor software quality in the US has grown to at least $2.41 trillion, but not in similar proportions as seen in 2020. The accumulated software Technical Debt (TD) has grown to ~$1.52 trillion.
Bug Management and Need for Automation

- Developers often receive more bugs than they can handle
- Developers spend a lot of time debugging

Survey: Fixing Bugs Stealing Time from Development

BY: MIKE VIZARD ON FEBRUARY 16, 2021 — 4 COMMENTS

A global survey of 950 developers published today finds more than a third (38%) of developers spend up to a quarter of their time fixing software bugs, with slightly more than a quarter (26%) spending up to half their time fixing bugs.
Bug Management and Need for Automation

Coping with an Open Bug Repository

John Anvik, Lyndon Hiew and Gail C. Murphy
Department of Computer Science
University of British Columbia
{janki, lyndon, murphy}@cs.ubc.ca

ETX 2005

"Everyday, almost 300 bugs appear that need triaging. This is far too much for only the Mozilla programmers to handle." – Mozilla Developer
Bug Management and Need for Automation

Emerging App Issue Identification from User Feedback: Experience on WeChat

Cuiyun Gao†, Wujie Zheng§*, Yuetang Deng§, David Lo‡, Jichuan Zeng†, Michael R. Lyu†, Irwin King†
†Dept. of Computer Science and Engineering, The Chinese University of Hong Kong, China
‡School of Information Systems, Singapore Management University, Singapore
§Tencent, Inc., China

{wujiezheng.yuetangdeng}@tencent.com, {cygao,jczeng,lyu,king}@cse.cuhk.edu.hk, davidlo@smu.edu.sg

ICSE 2019

“Unfortunately, the large quantity of user reviews (e.g., WeChat receives around 60,000 reviews per day) makes manual analysis inefficient and unrealistic.”
Key Question

Can we help developers by **automating** some tasks in the bug report management process?
Outline

I. Motivation

II. Data

III. Tasks

IV. Bug Localization

V. Duplicate Detection

VI. Open Challenges
I. What Data Can We Mine?

- People report errors and incidents that they encounter when using a software

- These reports often include:
  - Description of the issue
  - Steps to reproduce the issue
  - Severity level
  - Parts of the system affected by the issue
  - Failure traces

- Come in various “shapes and sizes”
Issue Report – Bugzilla, Manually Submitted

On First opening Firefox, the software hangs for 10 to 20 seconds when anything is entered into the address bar

Title

Informative Fields

- **Product:** Firefox
- **Component:** Address Bar
- **Version:** 68 Branch
- **Type:** defect
- **Priority:** P3  **Severity:** S3
- **Points:** 5

**Status:** UNCONFIRMED
Issue Report – Bugzilla, Manually Submitted

User Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0)
Gecko/20100101 Firefox/68.0

Steps to reproduce:

We have Firefox 68.5.0ESR. Installed using the .msi file via SCCM. We have a proxy server. GPO sets the following:
We can then set the following policies:
• Set the ImportEnterpriseRoots key to true.
• Under Proxy, set “Mode” to “autoDetect”
• DisableAppUpdate to true
• Under Homepage, setting URL to [intranet page], and StartPage to “homepage”
• Under PopupBlocking, setting “allow” to [intranet page];
• Under FlashPlugin, setting “allow” to [intranet page]
• Under OverrideFirstRunPage, set to Blank, so it will default to the homepage
• Under OverridePostUpdatePage, set to Blank, so it will default to the homepage
**Title**

Fatal exception starting Glassfish 6.1

**Informative Fields**

- **Type:** Bug
- **Status:** OPEN
- **Priority:**
- **Resolution:** Unresolved

**Detailed Description**

When I try to start Glassfish-6.1 within the Services-Server-Tab with JDK-1.8 it does not start, instead the following is displayed:

Error: Could not create the Java Virtual Machine.
Issue Report – Bugzilla, Submitted by Bot

I've submitted a try run for this commit: https://treeherder.mozilla.org/#!/job

The try push is done, we found jobs with unclassified failures.

Known Issues (From Push Health):

browser/components/extensions/test/browser/browser_ext_ - 4 of 4 failed on the same (retriggered) task

CI Failures
Wesley
★★★☆☆ July 2, 2021

The muted feedback option is bugged I'm sure. Before I could preview videos without sound and I was ok with that. But now there's an option to either have it on with sound playing when you look at the preview which at that point you might as well just click the video or turn it off but it turns off ...
Issue Report – WeChat Feedback Form

Feedback

Describe your issue using at least 10 characters so that we can help troubleshoot your issue more quickly

Screenshots for troubleshooting (Optional)

Mobile Contact

optional

Submit
Commits Linked to Issue Reports

Camel / CAMEL-16574

camel-salesforce - NPE On Response Callback - DefaultCompositeApiClient

✓ CAMEL-16574: Fixed NPE Exception On Empty Response (#5502)

main (#5502)

camel-3.11.0 camel-3.10.0

جزاء-ر3 committed on 5 May

Verified
II. What Tasks Can We Automate?

How Practitioners Perceive Automated Bug Report Management Techniques

Weiqin Zou, David Lo, Zhenyu Chen, Xin Xia, Yang Feng, Baowen Xu

Abstract—Bug reports play an important role in the process of debugging and fixing bugs. To reduce the burden of bug report managers and facilitate the process of bug fixing, a great amount of software engineering research has been invested toward automated bug report management techniques. However, the verdict is still open whether such techniques are actually required and applicable outside the domain of theoretical research. To fill this gap, we conducted a survey among 327 practitioners to gain their insights into various categories of automated bug report management techniques. Specifically, we asked the respondents to rate the importance of such techniques and provide the rationale. To get deeper insights into practitioners' perspective, we conducted follow-up interviews with 25 interviewees selected from the survey respondents. Through the survey and the interviews, we gained a better understanding of the perceived usefulness (or its lack) of different categories of automated bug report management techniques. Based on our findings, we summarized some potential research directions in developing techniques to help developers better manage bug reports.

TSE 2020
Literature Review

- **Paper Selection**
  - 7 journals and 10 conferences
  - Years (2006-2017)
  - Regular Papers
  - Non-Empirical Studies
  - Card Sorting

- **Journals**
  - TOSEM, TSE, EMSE, ASEJ, JSS, IST, TRel

- **Conferences**
  - ICSE, FSE, ASE, ICSME, ICPC, ISSTA, SANER, ESEM, ICST, MSR

115 papers of 10 categories
# Literature Review

<table>
<thead>
<tr>
<th>ID</th>
<th>Category</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Bug localization</td>
<td>24</td>
</tr>
<tr>
<td>T2</td>
<td>Bug assignment</td>
<td>22</td>
</tr>
<tr>
<td>T3</td>
<td>Duplicate/similar bug detection</td>
<td>14</td>
</tr>
<tr>
<td>T4</td>
<td>Bug categorization</td>
<td>12</td>
</tr>
<tr>
<td>T5</td>
<td>Bug fixing time prediction</td>
<td>10</td>
</tr>
<tr>
<td>T6</td>
<td>Bug severity/priority prediction</td>
<td>8</td>
</tr>
<tr>
<td>T7</td>
<td>Bug report completion/refinement</td>
<td>8</td>
</tr>
<tr>
<td>T8</td>
<td>Bug-commit linking</td>
<td>7</td>
</tr>
<tr>
<td>T9</td>
<td>Bug report summarization/visualization</td>
<td>5</td>
</tr>
<tr>
<td>T10</td>
<td>Reopened bug prediction</td>
<td>5</td>
</tr>
</tbody>
</table>

*List of papers:*
Automatable Tasks (T1)

Bug Localization

These techniques process a bug report, and locate relevant program elements that possibly contain the bug. Some of these techniques also recommend candidate repairs.

Where Should the Bugs Be Fixed?
More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports

Jian Zhou¹, Hongyu Zhang¹,* and David Lo²
¹School of Software, Tsinghua University, Beijing 100084, China
Tsinghua National Laboratory for Information Science and Technology (TNList)
zhoutian1286@yahoo.com.cn, hongyu@tsinghua.edu.cn
²School of Information Systems, Singapore Management University, Singapore
davidlo@smu.edu.sg

ICSE 2012
Automatable Tasks (T2)

Bug Assignment

These techniques process a bug report, and recommend the most appropriate developers to address it.

Who Should Fix This Bug?

John Anvik, Lyndon Hiew and Gail C. Murphy
Department of Computer Science
University of British Columbia
{janvik, lyndonh, murphy}@cs.ubc.ca

ABSTRACT

Open source development projects typically support an open bug repository to which both developers and users can re-

However, this potential advantage also comes with a significant cost. Each bug that is reported must be triaged to determine if it describes a meaningful new problem or whether it represents a known defect that has already been addressed.
Automatable Tasks (T3)

Duplicate / Similar Bug Detection

These techniques detect **duplicate / similar reports** in issue tracking systems.

---

**A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval**

Chengnian Sun\(^1\), David Lo\(^2\), Xiaoyin Wang\(^3\), Jing Jiang\(^3\), Siau-Cheng Khoo\(^4\)

\(^1\)School of Computing, National University of Singapore
\(^2\)School of Information Systems, Singapore Management University
\(^3\)Key laboratory of High Confidence Software Technologies (Peking University), Ministry of Education
suncn@comp.nus.edu.sg, davidlo@smu.edu.sg, wangxy06@sei.pku.edu.cn, jingjiang@smu.edu.sg, khoosc@comp.nus.edu.sg

**ABSTRACT**

Bug repositories are usually maintained in software projects. Testers or users submit bug reports to identify various issues with systems. Sometimes two or more bug reports correspond to the same defect. To address the problem with duplicate bug reports, a person called a triager needs to manually label these bug reports as duplicates, and link them. Due to complexities of systems built, software often comes with defects. Software defects have caused billions of dollars lost \([20]\). Fixing defects is one of the most frequent reasons for software maintenance activities which also goes to 70 billion US dollars in the United States alone \([19]\).

In order to help track software defects and build more reliable systems, bug tracking tools have been introduced.
Automatable Tasks (T4)

Bug Categorization

These techniques process a bug report, and classify it into different categories (e.g. invalid or not, bug or feature request, security bug report or not etc.)

Automatic Defect Categorization

Ferdian Thung, David Lo, and Lingxiao Jiang
School of Information Systems
Singapore Management University, Singapore
{ferdianthung,davidlo,lxiang}@smu.edu.sg

Abstract—Defects are prevalent in software systems. In order to understand defects better, industry practitioners often categorize bugs into various types. One common kind of categorization is the IBM’s Orthogonal Defect Classification (ODC). ODC proposes various orthogonal classification of defects based on much information about the defects, such as the symptoms and semantics of the defects, the root cause analysis of the defects, and many more. With these category labels, developers can better

In this paper, we propose a classification-based approach that categorizes defects into three families: control and data flow, structural, and non-functional. Our goal is to automatically classify a defect into one of the three families according to the content of the bug report and the associated code changes made to fix the bug. Given a large set of data about known defects and their fixes, various textual features corresponding
Bug Fixing Time Prediction

These techniques process a bug report, and predict how long it will take to fix the bug.

Bug-fix Time Prediction Models: Can We Do Better?

Pamela Bhattacharya  Iulian Neamtiu
Department of Computer Science and Engineering
University of California, Riverside, CA, USA
{pamelab, neamtiu}@cs.ucr.edu

ABSTRACT
Predicting bug-fix time is useful in several areas of software evolution, such as predicting software quality or coordinating development effort during bug triaging. Prior work has proposed bug-fix time prediction models that use various bug report attributes (e.g., number of developers who participated in fixing the bug, bug severity, number of patches, bug-opener’s reputation) for estimating the time it will take to fix a newly-reported bug. In this paper we take a

1. INTRODUCTION
Predicting bug-fix time is useful in several areas of software evolution, such as predicting software quality [9] or coordinating effort during bug triaging [8]. To this end, prior efforts have constructed bug-fix time prediction models, based on machine learning algorithms, on both open source and commercial projects. Prior studies on open source projects [8, 5, 1] have used various bug report attributes (e.g., number of developers involved in fixing
Automatable Tasks (T6)

Bug Severity / Priority Prediction

These techniques process a bug report, and predict its severity / priority.

Automated Severity Assessment of Software Defect Reports

Tim Menzies
Lane Department of Computer Science,
West Virginia University
PO Box 6109, Morgantown, WV, 26506
304 293 0405
tim@menzies.us

Andrian Marcus
Department of Computer Science
Wayne State University
Detroit, MI 48202
313 577 5408
amarcus@wayne.edu

Abstract

In mission critical systems, such as those developed by NASA, it is very important that the test engineers properly recognize the severity of each issue they identify during testing. Proper severity assessment is essential for appropriate resource allocation and planning for fixing activities and additional testing. However, when dealing with projects that have many concurrent changes, the specific configuration of the information captured about an issue was tailored by the IV&V project to meet its needs. This has created consistency problems when metrics data is pulled across projects. While there was a set of required data fields, the majorities of those fields do not provide information in regards to the quality of the issue and are not very suitable for...
Automatable Tasks (T7)

Bug Report Completion / Refinement

These techniques aim to generate a high-quality bug report. Some of these techniques automatically **generate a new bug report when software crashes**. Some others **enrich / modify** an existing one.

---

Auto-completing Bug Reports for Android Applications

Kevin Moran, Mario Linares-Vásquez, Carlos Bernal-Cárdenas, Denys Poshyvanyk

College of William & Mary
Department of Computer Science
Williamsburg, VA 23187-8795, USA
{kpmoran, milarev, cebernal, denys}@cs.wm.edu

ABSTRACT

The modern software development landscape has seen a shift in focus toward mobile applications as tablets and smartphones near ubiquitous adoption. Due to this trend, the complexity of these “apps” has been increasing, making development and maintenance challenging. Additionally, current bug tracking systems are not able to effectively support construction of reports with actionable information that directly lead to a bug’s resolution. To address the need for an improved reporting system, we introduce a novel solution,

1. INTRODUCTION

Smartphones and mobile computing have skyrocketed in popularity in recent years, and adoption has reached near-ubiquitous levels with over 2.7 billion active smartphone users in 2014 [26]. An increased demand for high-quality, robust mobile applications is being driven by a growing user base that performs an increasing number of computing tasks on “smart” devices. Due to this demand, the complexity of mobile applications has been increasing, making development and maintenance challenging. The intense competition
Automatable Tasks (T8)

Bug-Commit Linking

These techniques aim to link bug reports with bug fixing commits or bug inducing commits. With these techniques, developers can better understand which commits fix the bug and why/how/when the bug is introduced.

RCLinker: Automated Linking of Issue Reports and Commits Leveraging Rich Contextual Information

Tien-Duy B. Le*, Mario Linares-Vásquez†, David Lo*, and Denys Poshyvanyk†

*School of Information Systems, Singapore Management University
†Computer Science Department, The College of William and Mary
{bdle2012,davidlo}@smu.edu.sg, {mlinares,denys}@cs.wm.edu

Abstract—Links between issue reports and their corresponding commits in version control systems are often missing. However, these links are important for measuring the quality of a software system, predicting defects, and many other tasks. Several approaches have been designed to solve this problem by automatically linking bug reports to source code commits via comparison of textual information in commit messages and bug reports. Yet, the effectiveness of these techniques is approaches named ReLink [60] and MLink [47] respectively. These approaches enumerate a set of potential links and remove the ones that do not satisfy some criteria defined based on a set of thresholds. These thresholds are learned by heuristically enumerating various values based on a training and/or a validation dataset. A main operation in ReLink and MLink is the computation of similarity between the textual
Automatable Tasks (T9)

Bug Report Summarization / Visualization

These techniques process a bug report, and summarize it into a much shorter form. Some of these techniques also help developers better navigate / understand bug reports through visualization.

Automatic Summarization of Bug Reports

Sarah Rastkar, Gail C. Murphy, Member, IEEE, and Gabriel Murray

Abstract—Software developers access bug reports in a project’s bug repository to help with a number of different tasks, including understanding how previous changes have been made and understanding multiple aspects of particular defects. A developer’s interaction with existing bug reports often requires perusing a substantial amount of text. In this article, we investigate whether it is possible to summarize bug reports automatically so that developers can perform their tasks by consulting shorter summaries instead of entire bug reports. We investigated whether existing conversation-based automated summarizers are applicable to bug reports and found that the quality of generated summaries is similar to summaries produced for e-mail threads and other conversations. We also trained a summarizer on a bug report corpus. This summarizer produces summaries that are statistically better than summaries produced by existing conversation-based generators. To determine if automatically produced bug report summaries can help a developer with their work, we conducted a task-based evaluation that considered the use of summaries for bug report duplicate detection tasks. We found that summaries helped the study participants save time, that there was no evidence that accuracy degraded when summaries were used and that most participants preferred working with summaries to working with original bug reports.
Automatable Tasks (T10)

Reopened Bug Prediction

These techniques process a closed bug report, and predict whether it is likely to be reopened.

Characterizing and Predicting Which Bugs Get Reopened

Thomas Zimmermann ¹  Nachiappan Nagappan ¹  Philip J. Guo ²  Brendan Murphy ³
tzimmer@microsoft.com  nachin@microsoft.com  pg@cs.stanford.edu  bmurphy@microsoft.com
¹ Microsoft Research, USA ² Stanford University, USA ³ Microsoft Research, UK

Abstract—Fixing bugs is an important part of the software development process. An underlying aspect is the effectiveness of fixes: if a fair number of fixed bugs are reopened, it could indicate instability in the software system. To the best of our knowledge there has been little prior work on understanding the dynamics of bug reopens. Towards that end, in this the likelihood of being reopened, but to characterize the overall reopen process. In order to do so we employ a more foundational approach wherein we first survey a large population of experienced developers on the fundamental reasons for bug reopens and qualitatively analyze the responses. We then assess the reasons for reopens from a quantitative per-
Practitioners’ Perceptions

Potential gap between research and practice in automated bug report management:

• Are these techniques appreciated by practitioners?
• What are practitioners’ complaints and challenges?

This study:

• Investigates practitioner views of existing research work
• Identifies new research directions by learning from practice
Methodology

1. Literature Review
2. Survey
3. Interview
4. Data Analysis

Categories
Ratings + Comments
Statistical Analysis
Thematic Analysis
Surveys

Questionnaire

• Demographic questions

• 10 closed-ended questions for T1-T10 (very important->very unimportant)

• Up to 2 open-ended questions asking rationales of important/unimportant ratings

Participants

327 respondents

Industrial professionals

Open-source developers
Interviews

~1 hour, using video conferencing or in-person

Procedure

4 out of 10 categories with each interviewee

Participants

25 out of 107 survey respondents who left email addresses in the anonymous survey
Findings

\[\sim 70\% \text{ of ratings (}>80\% \text{ for testers)}\]

are important / very important
# Findings

<table>
<thead>
<tr>
<th>ID</th>
<th>Technique</th>
<th>Very Important</th>
<th>Very Unimportant</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>T1</td>
<td>Bug localization</td>
<td><strong>49.2%</strong></td>
<td>33.4%</td>
</tr>
<tr>
<td>T2</td>
<td>Bug assignment</td>
<td>33.4%</td>
<td><strong>38.9%</strong></td>
</tr>
<tr>
<td>T3</td>
<td>Duplicate / similar bug detection</td>
<td>35.8%</td>
<td><strong>41.6%</strong></td>
</tr>
<tr>
<td>T4</td>
<td>Bug categorization</td>
<td>33.3%</td>
<td><strong>41.0%</strong></td>
</tr>
<tr>
<td>T5</td>
<td>Bug fixing time prediction</td>
<td>17.8%</td>
<td>28.3%</td>
</tr>
</tbody>
</table>
## Findings

<table>
<thead>
<tr>
<th>ID</th>
<th>Technique</th>
<th>Very Important</th>
<th>Very Unimportant</th>
</tr>
</thead>
<tbody>
<tr>
<td>T6</td>
<td>Bug severity / priority prediction</td>
<td>25.6% 37.3%</td>
<td>26.9% 8.3% 1.9%</td>
</tr>
<tr>
<td>T7</td>
<td>Bug report completion / refinement</td>
<td>34.5% 42.8%</td>
<td>18.8% 3.1% 0.9%</td>
</tr>
<tr>
<td>T8</td>
<td>Bug-commit linking</td>
<td>36.7% 44.4%</td>
<td>15.1% 2.8% 0.9%</td>
</tr>
<tr>
<td>T9</td>
<td>Bug report summarization / visualization</td>
<td>25.9% 34.9%</td>
<td>25.6% 10.8% 2.8%</td>
</tr>
<tr>
<td>T10</td>
<td>Reopened bug prediction</td>
<td>17.8% 29.1% 32.2%</td>
<td>15.0% 5.9%</td>
</tr>
</tbody>
</table>
Findings

Ranks of ten categories of automated bug report management techniques according to the Scott-Knott ESD tests for all respondents.

<table>
<thead>
<tr>
<th>Group</th>
<th>Category of Bug Report Management Technique</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>T1 (Bug localization)</td>
</tr>
<tr>
<td></td>
<td>T3 (Duplicate/similar bug detection)</td>
</tr>
<tr>
<td></td>
<td>T7 (Bug report completion/refinement)</td>
</tr>
<tr>
<td></td>
<td>T8 (Bug-commit linking)</td>
</tr>
<tr>
<td>2</td>
<td>T2 (Bug assignment)</td>
</tr>
<tr>
<td></td>
<td>T4 (Bug categorization)</td>
</tr>
<tr>
<td>3</td>
<td>T9 (Bug report summarization/visualization)</td>
</tr>
<tr>
<td></td>
<td>T6 (Bug severity/priority prediction)</td>
</tr>
<tr>
<td>4</td>
<td>T5 (Bug fixing time prediction)</td>
</tr>
<tr>
<td></td>
<td>T10 (Re-opened bug prediction)</td>
</tr>
</tbody>
</table>

Highest Rank

Lowest Rank
Summary of **Green** Segment (Parts II, III)

- Bug reports come in various shapes and sizes
- Hundreds of papers on AI for issue management
  - Since “Who Should Fix this Bug?” (MIP ICSE 2016)
  - Categorized into 10 categories
- Perceived as *important* by practitioners (70-80%); top-4:
  - Bug Localization
  - Duplicate / Similar Bug Report Detection
  - Bug Report Completion / Refinement
  - Bug-Commit Linking
Outline

Motivation

I

II

III

IV

V

VI

Open Challenges

Duplicate Detection

Bug Localization

Data

Tasks

Tasks

Outline
IV. Bug Localization

(Billions of) Source Code Files

Ranked List of Files

File 1
File 2
File 3
...

Bug Report

IR-Based Bug Localization Technique

(Thousands of) Source Code Files
Popular Early Work

Where Should the Bugs Be Fixed?
More Accurate Information Retrieval-Based Bug Localization Based on Bug Reports

Jian Zhou¹, Hongyu Zhang¹,* and David Lo²

ICSE 2012
BugLocator

Source Code Files -> Indexing -> Index

File Size

New Bug Report -> Query Construction

Previously Fixed Bug Reports

Analyzing Past Similar Bugs -> Ranked Files (SimiRank)

Retrieval & Ranking with rVSM

Ranked Files (rVSMRank)

Query

Final Ranked Files (FinalRank)

FinalScore = (1 - \alpha) \times N(rVSMScore) + \alpha \times N(SimiScore)
rVSM Score

\[ rVSMScore(q, d) = g(#\text{term}) \times \cos(q, d) \]

\[ \cos(q, d) = \frac{\vec{V}_q \cdot \vec{V}_d}{|\vec{V}_q||\vec{V}_d|} \]

Classical VSM with

\[ tf(t,d) = \log(f_{td}) + 1 \]
\[ idf(t) = \log \left( \frac{\#docs}{n_t} \right) \]

Account for the fact that large files often contain bugs

\[ g(#\text{terms}) = \frac{1}{1 + e^{-N(#\text{terms})}} \]
SimiScore

Layer 1

\[ B \text{ (a bug to be located)} \]

Layer 2

A link represents the similarity between \( S_i \) and \( B \)

\[ S \text{ (all similar bugs of } B) \]

Layer 3

A link indicates the impact of a bug on a file

\[ F \text{ (source code files)} \]

\[
\text{SimiScore} = \sum_{\text{All } S_i \text{ that connect to } F_j} \left( \frac{\text{Similarity}(B, S_i)}{n_i} \right)
\]
## Subject Programs

<table>
<thead>
<tr>
<th>Project</th>
<th>Description</th>
<th>Study Period</th>
<th>#Fixed Bugs</th>
<th>#Source Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>SWT (v3.1)</td>
<td>An open source widget toolkit for Java</td>
<td>Oct 2004 - Apr 2010</td>
<td>98</td>
<td>484</td>
</tr>
<tr>
<td>AspectJ</td>
<td>An aspect-oriented extension to the Java programming language</td>
<td>Jul 2002 - Oct 2006</td>
<td>286</td>
<td>6485</td>
</tr>
<tr>
<td>ZXing</td>
<td>A barcode image processing library for Android applications</td>
<td>Mar 2010-Sep 2010</td>
<td>20</td>
<td>391</td>
</tr>
</tbody>
</table>
Results

![Bar Chart]

- **BugLocator**
- **SUM**

**Percentage**

- **Top 1**
- **Top 5**
- **Top 10**
- **MAP**

SMU Classification: Restricted
Recent Work with Industry

Legion: Massively Composing Rankers for Improved Bug Localization at Adobe

Darryl Jarman, Jeffrey Berry, Riley Smith, Ferdian Thung, and David Lo

TSE 2022
BugLocator’s Performance on Adobe

![Bar chart showing performance metrics for BugLocator on Adobe, comparing Open Source and AA versions. Metrics include MRR, MAP, Top 1, Top 5, and Top 10.]
Adobe’s Applicability Requirement

- Consulted 4 developers at Adobe Analytics
- For developers **new** to a repository:
  - the tool should identify a buggy file in the top 10 recommendations at least 70% of the time (*Top 10 score ≥ 70%*)
- For developers **familiar** with a repository
  - the tool should identify a buggy file in the top 5 recommendations at least 80% of the time (*Top 5 score ≥ 80%*)
BL+: Extending BL with Additional Corpora

- Pre-processing
- SimiScore past comments
- FinalScore $\alpha$ values

BugLocator

rVSMScore Corpora Options
- [x] Source code file content
- [ ] Source code differences
- [ ] Commit messages
- [x] Bug report summary
- [ ] Bug report description
- [ ] Bug report past comments

2,772 configurations
Legion: Composing BL Configurations

1. Run all BL configurations
   - BL+1
   - BL+2111
   - BL+888
   - BL+1333
   - BL+1951
   - ... BL+2772

2. Generate stacked scores
   - FinalScore_1
   - FinalScore_2
   - FinalScore_3
   - ... FinalScore_2772
   - Stacked Score

3. Learn a supervised model
   - FinalScore_1
   - FinalScore_2
   - ... FinalScore_2772
   - Stacked Score
   - Random Forest
Legion Performance

Evaluation Metric

Score

<table>
<thead>
<tr>
<th>Metric</th>
<th>Legion</th>
<th>BL+</th>
<th>BL</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top 10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Other Works

Network-Clustered Multi-Modal Bug Localization

Thong Hoang, Richard J. Oentaryo, Tien-Duy B. Le, and David Lo
School of Information Systems, Singapore Management University
{vdthoang.2016, roentaryo, btdle.2012, davidlo}@smu.edu.sg

TSE 2019

Deep Transfer Bug Localization

Xuan Huo*, Ferdian Thung, Ming Li*, Member, IEEE, David Lo*, Member, IEEE, and Shu-Ting Shi

TSE 2021
Outline

I. Motivation

II. Data

III. Tasks

IV. Open Challenges

V. Duplicate Detection

VI. Bug Localization
### IV. Duplicate Bug Report Detection

**Duplicate Bug Report Detection (DBRD) Process**

**new report** \( Q \)

**Buckets**

- Master-1: dup-1.1, dup-2.1, …
- Master-2: dup-2.1, dup-2.2, …
- Master-3: dup-3.1, dup-3.2, …
- …
- Master-M: dup-M.1, dup-M.2, …

**Similarity Measure**

- Returns
  - Top 1
  - Top 2
  - …
  - Top K
Popular Early Work

ASE 2011

Towards More Accurate Retrieval of Duplicate Bug Reports

Chengnian Sun*, David Lo†, Siau-Cheng Khoo*, Jing Jiang†

*School of Computing, National University of Singapore
†School of Information Systems, Singapore Management University

sunen@comp.nus.edu.sg, davidlo@smu.edu.sg, khoose@comp.nus.edu.sg, jingjiang@smu.edu.sg

Most Cited Research Paper of ASE 2011
REP: Lightweight, Learning-Based DBRD

- We want to design an approach that can learn from historical data
- The approach needs to consider specific properties of bug reports
- The approach needs to be lightweight enough

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Summ</td>
<td>Summary: concise description of the issue</td>
</tr>
<tr>
<td>Desc</td>
<td>Description: detailed outline of the issue, such as what is the issue and how it happens</td>
</tr>
<tr>
<td>Prod</td>
<td>Product: which product the issue is about</td>
</tr>
<tr>
<td>Comp</td>
<td>Component: which component the issue is about</td>
</tr>
<tr>
<td>Vers</td>
<td>Version: the version of the product the issue is about</td>
</tr>
<tr>
<td>Prio</td>
<td>Priority: the priority of the report, i.e., P1, P2, P3, ⋯</td>
</tr>
<tr>
<td>Type</td>
<td>Type: the type of the report, i.e., defect, task, feature</td>
</tr>
</tbody>
</table>
REP: Lightweight Learning-Based DBRD

\[ REP(d, q) = \sum_{i=1}^{7} w_i \times feature_i \]

19 parameters tuned based on historical human decisions on training data

\[ \sum_{t \in d \cap q} IDF(t) \times \frac{TF_D(d, t)}{k_1 + TF_D(d, t)} \times W_Q \]

where \[ W_Q = \frac{(k_3 + 1) \times TF_Q(q, t)}{k_3 + TF_Q(q, t)} \]

\[ \begin{cases} 1, & \text{if } d.prod = q.prod \\ 0, & \text{otherwise} \end{cases} \]
\[ \begin{cases} 1, & \text{if } d.comp = q.comp \\ 0, & \text{otherwise} \end{cases} \]
\[ \begin{cases} 1, & \text{if } d.type = q.type \\ 0, & \text{otherwise} \end{cases} \]
\[ \frac{1}{1 + |d.prio - q.prio|} \]
\[ \frac{1}{1 + |d.vers - q.vers|} \]

Modification: BM25F is designed for short queries, while bug reports can be long
## Experiment - Dataset

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Size</th>
<th>Period</th>
<th>Training Reports</th>
<th>Testing Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>From</td>
<td>To</td>
<td>#Duplicate</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>31,138</td>
<td>2008-01-01</td>
<td>2010-12-21</td>
<td>200</td>
</tr>
<tr>
<td>Mozilla</td>
<td>75,653</td>
<td>2010-01-01</td>
<td>2010-12-31</td>
<td>200</td>
</tr>
<tr>
<td>Eclipse</td>
<td>45,234</td>
<td>2008-01-01</td>
<td>2008-12-31</td>
<td>200</td>
</tr>
<tr>
<td>Large Eclipse</td>
<td>209,058</td>
<td>2001-10-10</td>
<td>2007-12-14</td>
<td>200</td>
</tr>
</tbody>
</table>
BM25F\textsubscript{ext} is more effective than BM25F
Experiment - Result

REP is more effective and efficient than ICSE’10 (SVM)
Benchmarking Study

TOSEM 2022

Duplicate Bug Report Detection: How Far Are We?

TING ZHANG, Singapore Management University, Singapore
DONGGYUN HAN, Royal Holloway, University of London, United Kingdom
VENKATESH VINAYAKARAO, Chennai Mathematical Institute, India
IVANA CLAIRINE IRSAN, Singapore Management University, Singapore
BOWEN XU*, Singapore Management University, Singapore
FERDIAN THUNG, Singapore Management University, Singapore
DAVID LO, Singapore Management University, Singapore
LINGXIAO JIANG, Singapore Management University, Singapore
Motivation

- Limitations of existing datasets: old bug reports from Bugzilla with latest status, e.g., SABD\(^1\) was trained and evaluated on the following dataset:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Period</th>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Duplicate</td>
<td>All</td>
<td>Duplicate</td>
<td>All</td>
</tr>
<tr>
<td>Eclipse</td>
<td>10/10/01 - 31/12/08</td>
<td>27,481</td>
<td>198,183</td>
<td>1,446</td>
<td>14,703</td>
</tr>
<tr>
<td>Mozilla</td>
<td>07/04/98 - 31/12/10</td>
<td>122,199</td>
<td>438,806</td>
<td>6,431</td>
<td>44,014</td>
</tr>
<tr>
<td>OpenOffice</td>
<td>16/10/00 - 31/12/10</td>
<td>13,570</td>
<td>80,786</td>
<td>714</td>
<td>4,109</td>
</tr>
<tr>
<td>Netbeans</td>
<td>21/08/98 - 31/12/08</td>
<td>16,639</td>
<td>116,351</td>
<td>875</td>
<td>5,548</td>
</tr>
</tbody>
</table>

- Lack of comparison among
  - Recent research tools, e.g., SABD\(^1\), DC-CNN\(^2\), HINDBR\(^3\)
  - Research and industrial tools

---


Motivation

We aim to:

- Provide a benchmark that addresses the limitations of existing datasets
- Compare research tools on the same dataset
- Compare research and industrial tools
Research Questions

**RQ1**: How significant are the potential biases on the evaluation of DBRD techniques?

**RQ2**: How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?

**RQ3**: How do the DBRD approaches proposed in research literature compare to those used in practice?
Research Questions

RQ1: How significant are the potential biases on the evaluation of DBRD techniques?

RQ2: How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?

RQ3: How do the DBRD approaches proposed in research literature compare to those used in practice?
### RQ1 - Dataset


<table>
<thead>
<tr>
<th>Project</th>
<th>Age</th>
<th>Train</th>
<th></th>
<th>Test</th>
<th></th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># BRs (% Dup)</td>
<td># Dup Pairs</td>
<td># BRs (% Dup)</td>
<td># BRs (% Dup)</td>
<td># Master BRs</td>
</tr>
<tr>
<td>Mozilla</td>
<td>Old</td>
<td>198,653 (9.9%)</td>
<td>35,474</td>
<td>139,502 (9.9%)</td>
<td>338,155 (9.9%)</td>
<td>21,554</td>
</tr>
<tr>
<td></td>
<td>Recent</td>
<td>137,886 (10.1%)</td>
<td>60,498</td>
<td>55,701 (11.2%)</td>
<td>193,587 (10.4%)</td>
<td>10,702</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Old</td>
<td>49,355 (5.5%)</td>
<td>4,482</td>
<td>25,021 (12.1%)</td>
<td>74,376 (7.7%)</td>
<td>3,254</td>
</tr>
<tr>
<td></td>
<td>Recent</td>
<td>19,607 (4.7%)</td>
<td>1,725</td>
<td>7,976 (6.5%)</td>
<td>27,583 (5.2%)</td>
<td>959</td>
</tr>
</tbody>
</table>

#### State bias: The percentage of BRs changed the corresponding state in 2018–2020

<table>
<thead>
<tr>
<th>Platform</th>
<th>Summary</th>
<th>Description</th>
<th>Product</th>
<th>Component</th>
<th>Priority</th>
<th>Severity</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>10.8%</td>
<td>-</td>
<td>7.8%</td>
<td>11.7%</td>
<td>1.2%</td>
<td>5.6%</td>
<td>8.6%</td>
</tr>
<tr>
<td>Mozilla</td>
<td>11.8%</td>
<td>-</td>
<td>21.4%</td>
<td>24.5%</td>
<td>24.5%</td>
<td>5.4%</td>
<td>4.2%</td>
</tr>
</tbody>
</table>
## RQ1 - Dataset

<table>
<thead>
<tr>
<th>ITS</th>
<th>Project</th>
<th>Train</th>
<th>Test</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td># BRs (% Dup)</td>
<td># Dup Pairs</td>
<td># BRs (% Dup)</td>
</tr>
<tr>
<td></td>
<td></td>
<td># BRs (% Dup)</td>
<td># Dup Pairs</td>
<td># BRs (% Dup)</td>
</tr>
<tr>
<td></td>
<td></td>
<td># BRs (% Dup)</td>
<td># Dup Pairs</td>
<td># BRs (% Dup)</td>
</tr>
<tr>
<td></td>
<td></td>
<td># BRs (% Dup)</td>
<td># Dup Pairs</td>
<td># BRs (% Dup)</td>
</tr>
</tbody>
</table>

**ITS bias:** Statistics of data in the six projects
### RQ1 - Result

| Bias      | Approach | Data       | \( p \)-value | \( |d| \)       |
|-----------|----------|------------|----------------|--------------|
| Age       | REP      | Eclipse    | 0.003          | 0.78 (large) |
|           |          | Mozilla    | 0.005          | 0.72 (large) |
|           | Siamese Pair | Eclipse  | < 0.001        | 1 (large)    |
|           |          | Mozilla    | 0.003          | 0.76 (large) |
|           | SABD     | Eclipse    | 0.001          | 0.82 (large) |
|           |          | Mozilla    | 0.012          | 0.66 (large) |
| State     | REP      | Eclipse    | 0.105          | 0.44 (medium) |
|           |          | Mozilla    | 0.190          | 0.36 (medium) |
|           | Siamese Pair | Eclipse  | 0.063          | 0.5 (large)  |
|           |          | Mozilla    | 0.190          | 0.36 (medium) |
|           | SABD     | Eclipse    | 0.315          | 0.28 (small) |
|           |          | Mozilla    | 0.315          | 0.28 (small) |
| ITS       | REP      | Jira       | 0.056          | 0.36 (medium) |
|           |          | GitHub     | < 0.001        | 0.66 (large) |
|           | Siamese Pair | Jira    | < 0.001        | 1 (large)    |
|           |          | GitHub     | < 0.001        | 0.97 (large) |
|           | SABD     | Jira       | < 0.001        | 0.97 (large) |
|           |          | GitHub     | < 0.001        | 0.77 (large) |

**Answer:**

Age Bias and ITS Bias have a statistically significant impact, while State Bias does not.
Research Questions

**RQ1:** How significant are the potential biases on the evaluation of DBRD techniques?

**RQ2:** How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?

**RQ3:** How do the DBRD approaches proposed in research literature compare to those used in practice?
RQ2 – Result

Lightweight & overall best performer in 2022, over various industry and research tools on recent diverse datasets

Answer:
Overall, REP performs the best, especially for typical bug repositories with <10k bug reports.¹

¹ Average number of issues in 994 repositories: 2,365: Joshi, Saket Dattatray, and Sridhar Chimalakonda. "Rapidrelease-a dataset of projects and issues on github with rapid releases." In MSR 2019.
Research Questions

**RQ1:** How significant are the potential biases on the evaluation of DBRD techniques?

**RQ2:** How do state-of-the-art DBRD research tools perform on recent data from diverse ITSs?

**RQ3:** How do the DBRD approaches proposed in research literature compare to those used in practice?
RQ3 - VSCode dataset

Recall Rate@$k$ comparing the tools in research and in practice on the VSCode data
Can we do better?

Combining REP and ChatGPT for better DBRD
Motivation

ChatGPT is latest advanced generative AI technique
Process

1. Applying Selection Rules
   - Selected
   - Unselected

2. Prompt Template

3. REP
Methodology

▪ Selection Rules to select bug reports for ChatGPT to process:
  ▪ *Length*: long bug reports
  ▪ *Content*: complex structure

▪ Prompt template:

```
Prompt Template:
I have a bug report which contains summary and
→ description. I want you to select keywords
→ from both parts which keep the main meaning
→ of the bug report. These keywords would be
→ used for duplicate bug report detection.
→ Output format: "Summary: Selected Keywords
→ \n Description: Selected Keywords" \n
-> Summary: [Summary] \n-> Description: [Description]
```
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Total Bugs</th>
<th>Training Pairs</th>
<th>Validation Pairs</th>
<th>Testing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>9,579</td>
<td>626</td>
<td>26</td>
<td>81</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>59 (72%)</td>
</tr>
<tr>
<td>Hadoop</td>
<td>14,016</td>
<td>626</td>
<td>26</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>57 (62%)</td>
</tr>
<tr>
<td>Kibana</td>
<td>17,016</td>
<td>724</td>
<td>28</td>
<td>184</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>114 (62%)</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RR@1</th>
<th>RR@3</th>
<th>RR@5</th>
<th>RR@10</th>
<th>Improv. Over SOTA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spark</td>
<td>0.346</td>
<td>0.432</td>
<td>0.481</td>
<td>0.593</td>
<td>6.7%</td>
</tr>
<tr>
<td>Hadoop</td>
<td>0.391</td>
<td>0.565</td>
<td>0.609</td>
<td>0.652</td>
<td>7%</td>
</tr>
<tr>
<td>Kibana</td>
<td>0.408</td>
<td>0.571</td>
<td>0.62</td>
<td>0.674</td>
<td>8.7%</td>
</tr>
</tbody>
</table>
Description
Builds are failing in PR with following exception in native client.

[WARNING] make[2]: Leaving directory `/home/jenkins/jenkins-slave/workspace/hadoop-multibranch_PR-2084/
[WARNING] /opt/cmake/bin/cmake -E cmake_progress_report /home/jenkins/jen
[WARNING] [ 288] Built target common.ob
[WARNING] make[2]: Leaving directory `/home/jenkins/jenkins-slave/workspace/hadoop-multibranch_PR-2084/
[WARNING] /opt/cmake/bin/cmake -E cmake_progress_report /home/jenkins/jen
[WARNING] [ 288] Built target gmock_main.obj
[WARNING] make[1]: Leaving directory `/home/jenkins/jenkins-slave/workspace/hadoop-multibranch_PR-2084/
[WARNING] Makefile:127: recipe for target `all' failed
[WARNING] make[2]: *** No rule to make target `/home/jenkins/jenkins-slave
[WARNING] make[1]: *** [main/native/libhdfspp/lib/proto/CMakeFiles/proto_...]
[WARNING] make[1]: *** [all] Error 2
[INFO] Reactor Summary:

[INFO] Apache Hadoop Main SUCCESS [ 0].
[INFO] Apache Hadoop Build Tools SUCCESS [ 1].
[INFO] Apache Hadoop Project POM SUCCESS [ 0].
[INFO] Apache Hadoop Annotations SUCCESS [ 1].
[INFO] Apache Hadoop Project Dist POM SUCCESS [ 0].
[INFO] Apache Hadoop Assemblies SUCCESS [ 0].
[INFO] Apache Hadoop Maven Plugins SUCCESS [ 4].

Creating this ticket, as couple of pull requests had the same issue.

e.g https://builds.apache.org/job/hadoop-multibranch/job/PR-1591/2/artifact/out/patch-compile-root.txt

Correct Master Bug Report
Description
This issue is to run mvn javadoc: javadoc successfully in Apache Hadoop with Java 11.
Now there are many errors.

ChatGPT keywords: Javadoc, HTML version, HTML4, HTML5, warning, comments, valid, GeneratedMessageV3, package, not found, error
Summary of **Yellow Segment** (Parts IV, V)

- **Bug localization**
  - Leverage multiple notions of *similarity*
  - *Diverse artifacts* in repos can be used
  - Meaning of similarity can be *learned* from history

- **Duplicate bug report detection**
  - *Historical* data can be used to tune detectors
  - *Biases* can affect experiment results
  - *LLM* can be helpful
Outline

Motivation

I

II

III

IV

V

VI

Open Challenges

Duplicate Detection

Bug Localization

Data

Tasks

Tasks
Open Challenges I

**Explainable** Automated Bug Report Management
Open Challenges II

Tight Integration with Developer Workflow

(Experimental duplicate detection)
Thanks for submitting this issue. Please also check if it is already covered by an existing one, like:

- issue (#130958)
- bugs (#130911)
- setting is not work (#127954)
- Terminal not working (#129507)
- Cannot find file (#129006)
## Open Challenges III

### Holistic Analysis of Multiple Tasks

<table>
<thead>
<tr>
<th>ID</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Bug localization</td>
</tr>
<tr>
<td>T2</td>
<td>Bug assignment</td>
</tr>
<tr>
<td>T3</td>
<td>Duplicate/similar bug detection</td>
</tr>
<tr>
<td>T4</td>
<td>Bug categorization</td>
</tr>
<tr>
<td>T5</td>
<td>Bug fixing time prediction</td>
</tr>
<tr>
<td>T6</td>
<td>Bug severity/priority prediction</td>
</tr>
<tr>
<td>T7</td>
<td>Bug report completion/refinement</td>
</tr>
<tr>
<td>T8</td>
<td>Bug-commit linking</td>
</tr>
<tr>
<td>T9</td>
<td>Bug report summarization/visualization</td>
</tr>
<tr>
<td>T10</td>
<td>Re-opened bug prediction</td>
</tr>
</tbody>
</table>
Open Challenges III

Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction

Yuan Tian¹, David Lo¹, and Chengnian Sun²
¹Singapore Management University, Singapore
²National University of Singapore, Singapore
{yuan.tian.2011,davidlo}@smu.edu.sg, suncn@comp.nus.edu.sg

WCRE 2012

“Duplicate bug reports are utilized to determine what bug report features, be it textual, ordinal, or categorical, are important.”

Won Most Influential Paper Award @ SANER 2022
Open Challenges IV

Prioritizing User Feedback from Twitter: A Survey Report

Emilia Guzman
Mohamed Ibrahim
Martin Glinz

CSI-SE 2017

Screen capture tool with video recording

- Add images or videos with voice over to your bug reports
- Add comments directly to the snapshot to provide effective feedback
- Create instant bug report & transfer it to ReQtest
- Get the bug report to the right person – right away!

https://reqtest.com/

Handle Rich Media
Emerging App Issue Identification from User Feedback: Experience on WeChat

Cuiyun Gao†, Wujie Zheng§*, Yuetang Deng§, David Lo†, Jichuan Zeng†, Michael R. Lyu†, Irwin King†

A Machine Learning Approach for Vulnerability Curation

Yang Chen
Veracode
ychen@veracode.com

Abhishek Sharma
Veracode
absharma@veracode.com

Andrew E. Santosa
Veracode
asantosa@veracode.com

Asankhaya Sharma
Veracode
asharma@veracode.com

Ang Ming Yi
Veracode
mang@veracode.com

David Lo
Singapore Management University
davidlo@smu.edu.sg

Won ACM SIGSOFT Distinguished Paper Award

ICSE 2019

MSR 2020

Industrial Collaborations
Automating App Review Response Generation

Cuiyun Gao† Jichuan Zeng† Xin Xia‡ David Lo§ Michael R. Lyu† Irwin King†
†Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, China
‡Faculty of Information Technology, Monash University, Australia
§School of Information Systems, Singapore Management University, Singapore
{cygao,jczeng,lyu,king}@cse.cuhk.edu.hk xin.xia@monash.edu davidlo@smu.edu.sg

ASE 2019

New Tasks
TrustedSEERs: Trusted SE Expert advisoRs
Building trusted bots towards Software Engineering 2.0

NRF Investigatorship project, 2023-2028 ($3.2M)
Individual research grant, similar to ERC Advanced
Thank you!

Questions? Comments? Advice?
davidlo@smu.edu.sg