It’s Not a Bug, It’s a Feature: How Misclassification Impacts Bug Prediction

Kim Herzig, Sascha Just and Andreas Zeller (Saarland Univ.)

ICSE 2013

2013. 10. 02.
Presented by Kim, Hanseok
Contents

- Introduction
- Overview of our approach
  - Design of the study
  - Results
  - Consequences of the study
- Conclusion
- Discussion
Bug predictions depend on the quality of the data from change and bug database.

A general challenge about mining is to separate bugs from non-bugs.

In a bug database, there still exists many misclassified issue reports [1]

- bug: corrective code maintenance
- non-bug: perfective / adaptive maintenance, refactoring, discussions, requests for help, and so on

Introduction (2/3)

Motivation

- Many issue reports are incorrectly classified in a bug category, even if they are in non-bug category

- Previous research\[1\] on this misclassification doesn’t answer the following questions:
  - How often does such misclassification occur?
  - Does it actually bias analysis and prediction?

1) G. Antoniol et al., Is it a bug or an enhancement? A text-based approach to classify change requests, 2008
Goal of this paper

- Extend the research\[1\]
  - By measuring the amount of data noise introduced by misclassified bug reports
  - By showing the possible impact of misclassified bug report types on bug mapping strategies

- Make a well-classified set of bug reports and features
  - It can leverage for future research
  - It will make developers more conscious of maintaining data quality

1) G. Antoniol et al., Is it a bug or an enhancement? A text-based approach to classify change requests, 2008
Overview of our approach

**Step 1. Manual Inspection**
- 5 open-source projects
- 2 bug tracking systems
- 7,401 closed & fixed issue reports

**Step 2. Data Preparation**

**Step 3. Analyzation**
- Overall noise rates (RQ1)
- Percentage of misclassified reports (RQ2)
- Impact on mapping (RQ3)
- Impact on bug count (RQ4)
Design of study (1/3)

_project details_

- Five open-source JAVA projects
  - Follow strict commit, bug fixing process similar to industry
  - Have homogeneous data set to ease the manual inspection
  - Use two popular bug tracking systems: Bugzilla and Jira
- 7,401 closed and fixed issue reports
  - 1,810 reports from Bugzilla
  - 5,591 reports from Jira

<table>
<thead>
<tr>
<th>Maintainer</th>
<th>Tracker type</th>
<th># reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPClient</td>
<td>APACHE</td>
<td>Jira</td>
</tr>
<tr>
<td>Jackrabbit</td>
<td>APACHE</td>
<td>Jira</td>
</tr>
<tr>
<td>Lucene-Java</td>
<td>APACHE</td>
<td>Jira</td>
</tr>
<tr>
<td>Rhino</td>
<td>MOZILLA</td>
<td>Bugzilla</td>
</tr>
<tr>
<td>Tomcat5</td>
<td>APACHE</td>
<td>Bugzilla</td>
</tr>
</tbody>
</table>
Manual inspection Process

- **Used**
  - Issue report itself
  - Attached comments / discussions
  - Code change in the source code

- **Categorized**
  - One of eleven *categories*
  - Used a **fixed set of rules** to classify issue reports

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BUG</strong></td>
<td>Issue reports documenting corrective maintenance tasks that require semantic changes to source code.</td>
</tr>
<tr>
<td><strong>RFE</strong></td>
<td>Issue reports documenting an adaptive maintenance task whose resolving patch(es) implemented new functionality (request for enhancement; feature request).</td>
</tr>
<tr>
<td><strong>IMPR</strong></td>
<td>Issue reports documenting a perfective maintenance task whose resolution improved the overall handling or performance of existing functionality.</td>
</tr>
<tr>
<td><strong>DOC</strong></td>
<td>Issue reports solved by updating external (e.g. website) or code documentation (e.g. JavaDoc).</td>
</tr>
<tr>
<td><strong>REFAC</strong></td>
<td>Issues reports resolved by refactoring source code. Typically, these reports were filed by developers.</td>
</tr>
<tr>
<td><strong>OTHER</strong></td>
<td>Any issue report that did not fit into any of the other categories.</td>
</tr>
</tbody>
</table>

A report is categorized as **BUG** (Fix Request) if...
1. it reports a *NullPointerException (NPE).*
2. the discussion concludes that code had to be changed semantically to perform a corrective maintenance task.
3. it fixes runtime or memory issues caused by defects (e.g. endless loops).

A report is categorized as **RFE** (Feature Request) if...
1. it requests to implement a new access/getter method.
2. it requests to add new functionality.
3. it requests to support new object types, specifications, or standards.

A report is categorized as **IMPR** (Improvement Request) if...
Design of study (3/3)

- Manual inspection Process (Cont’d)
  - Conducted
    - 3 phases manual report inspection process
      > with 7,401 issue reports and report classification rules
Results (1/9)

- Amount of data noise in the bug DB
  - Measured “false positive rates” as overall noise rate
    - Ratio between misclassified issue reports and all issue reports in the data set
  - [RQ 1]
    - Do bug databases contain data noise due to issue report misclassification, and how much?

<table>
<thead>
<tr>
<th>Project</th>
<th>Noise rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPClient</td>
<td>47.8%</td>
</tr>
<tr>
<td>Jackrabbit</td>
<td>37.6%</td>
</tr>
<tr>
<td>Lucene-Java</td>
<td>46.4%</td>
</tr>
<tr>
<td>Rhino</td>
<td>43.2%</td>
</tr>
<tr>
<td>Tomcat5</td>
<td>41.4%</td>
</tr>
<tr>
<td>All projects combined</td>
<td>42.6%</td>
</tr>
</tbody>
</table>

- Noise rates : 37% ~ 47%
- Overall noise rate : 42.6% (about 2/5)

Over all five projects researched, we found 42.6% of all issue reports to be wrongly typed!
Results (2/9)

- Amount of data noise in the bug DB (cont’d)
  - Sliced issue categories to increase the level of detail
  - [RQ 2]
    - Which percentage of issue reports associated with a category was marked as misclassified?
    - Which category do these misclassified reports actually belong to?

Every third bug report is no bug report!
Discussion of RQ1 and RQ2

Over all five projects researched, we found **42.6%** of all issue reports to be wrongly typed!

Every **third** bug report is no bug report!

- Why do bug tracking systems contain so many misclassified reports?
  - Users and developers have different views and insights on bug classification
    - Users (Reporter) tend to consider every problem as a bug!
    - Developers (Resolver) are interested in only resolution
    - And most developers don’t have no real motivation to change the issue category after fixing a problem
Discussion of RQ1 and RQ2 (Cont’d)

- Why do bug tracking systems contain so many misclassified reports?
  - Using their default configuration, many bug tracking systems set the report type to BUG by default
    - We are left with many BUG reports that should have been filed as IMPR or even RFE
  - The definition of whether an issue is a bug or not is a hard one
    - Not only differs between users and developers, but also between developers themselves!
Impact on mapping from report to source code

- [RQ 3]
  - What is the impact of misclassified issue report when mapping issue reports to source code changes?

How?

- Map issue reports to revision using the issue report mapping strategy\[1\]
- And then get the set of issue reports that caused a change within the source file
- Count two numbers and calculate four bias measurements
  - num_original_bugs, num_classified_bugs
  - MappingBiasRate, DiffBugNumRate, MissDefectRate, FalseDefectRate

1) T. Zimmermann et al., Predicting defects for Eclipse, 2007
Results (6/9)

- Every third bug report that can be associated to code changes is misclassified
- On average 37% of all sources files have biased bug count #
- Files that were falsely marked as defect-free is very low
- Files that were falsely marked as defect-prone is significant (20~70%)
  → Mapping bias is a real threat to any defect prone classification model

On average, 39% of all files marked as defective actually never had a bug!
Impact on bug count

[RQ 4]
- How does bug mapping bias introduced by misclassified issue reports impact the TOP 5%, 10%, 15%, 20% of most defect prone source files?

How?
- Duplicate the set of source files
- Sort each copy by two different criteria (descending order)
  - # of original bug reports (num_original_bugs)
  - # of manually classified bug reports (num_classified_bugs)
- Calculate \( \text{cutoff} \_\text{difference} \)
  \[
  \frac{\text{size of cutoff} - \text{size of intersection}}{\text{size of cutoff}}
  \]

Why?
- The most defect prone file is the top element
- We can reason about the impact of mapping bias on models using bug count
  - Low cutoff_difference is desirable
Results (8/9)

Impact on bug count (cont’d)

<table>
<thead>
<tr>
<th></th>
<th>TOP 5%</th>
<th>TOP 10%</th>
<th>TOP 15%</th>
<th>TOP 20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPClient</td>
<td>20%</td>
<td>20%</td>
<td>11%</td>
<td>25%</td>
</tr>
<tr>
<td>Jackrabbit</td>
<td>29%</td>
<td>40%</td>
<td>29%</td>
<td>35%</td>
</tr>
<tr>
<td>Lucene-Java</td>
<td>24%</td>
<td>20%</td>
<td>21%</td>
<td>18%</td>
</tr>
<tr>
<td>Rhino</td>
<td>11%</td>
<td>16%</td>
<td>14%</td>
<td>16%</td>
</tr>
<tr>
<td>Tomcat5</td>
<td>14%</td>
<td>21%</td>
<td>29%</td>
<td>21%</td>
</tr>
</tbody>
</table>

- The bias measured for all projects → Cutoffs lies well above 10%
- False positive rate (16~40%) for the top 10% most defect-prone files

When predicting the top 10% most defect-prone files, 16% to 40% of the files do not belong in this category because of misclassification
Overview from RQ1 to RQ4

Over all five projects researched, we found **42.6%** of all issue reports to be wrongly typed!

Every **third** bug report is no bug report!

On average, **39%** of all files marked as defective actually never had a bug!

When predicting the top 10% most defect-prone files, **16% to 40%** of the files do not belong in this category because of misclassification.
Automated quantitative analysis should always include human qualitative analysis of the input data
  • Should do manual validation of data quality before using bug datasets

Filtering out non-bugs (among bugs) when estimating code quality might even improve results

The categorization of bug reports is dependent on the perspective of the observer
  • Should validate whether the perspective of the prediction model matches the perspective of the bug creator
Conclusion

❖ Contribution
   ▪ Investigated the possible impact of misclassified bug reports using specific studies and experiments
     • Measured data noise and bias from the data noise, ...
   ▪ Constructed rectified data sets using manual report inspection process
     • Can encourage their use for further research

❖ Future work
   ▪ Exploring the quality of industrial data sets
   ▪ Exploring differences between industrial and open source projects
Discussion

Limitation

- The manual inspection can contain errors
- The set of classification is only one possibility to classify issue reports.
  - i.e., There exists no clear definition about feature and improvement requests
- The projects and bug tracking systems that we used might not be representative
  - It threatens the external validity of the findings
Thank you :)
Kim Herzig
- Joined Microsoft Research in Cambridge early 2013
- Graduated Saarland Univ. in Germany
- Interested in
  - Empirical software engineering
  - Mining software repositories

Andreas Zeller
- Professor at Saarland Univ. in Germany
- Interested in
  - Software testing
  - Specification mining
  - Automatic parallelization
  - Mining software archives
  - Automated debugging
## Issue report categories

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BUG</strong></td>
<td>Issue reports documenting corrective maintenance tasks that require semantic changes to source code.</td>
</tr>
<tr>
<td><strong>RFE</strong></td>
<td>Issue reports documenting an adaptive maintenance task whose resolving patch(es) implemented new functionality (request for enhancement; feature request).</td>
</tr>
<tr>
<td><strong>IMPR</strong></td>
<td>Issue reports documenting a perfective maintenance task whose resolution improved the overall handling or performance of existing functionality.</td>
</tr>
<tr>
<td><strong>DOC</strong></td>
<td>Issue reports solved by updating external (e.g. website) or code documentation (e.g. JavaDoc).</td>
</tr>
<tr>
<td><strong>REFAC</strong></td>
<td>Issues reports resolved by refactoring source code. Typically, these reports were filed by developers.</td>
</tr>
<tr>
<td><strong>OTHER</strong></td>
<td>Any issue report that did not fit into any of the other categories. This includes: reports requesting a backport (<strong>BACKPORT</strong>), code cleanups (<strong>CLEANUP</strong>), changes to specification (rather than documentation or code; <strong>SPEC</strong>), general development tasks (<strong>TASK</strong>), and issues regarding test cases (<strong>TEST</strong>). These subcategories are found in the public dataset accompanying this paper.</td>
</tr>
</tbody>
</table>
Classification rules

A report is categorized as **BUG** (Fix Request) if...

1) it reports a `NullPointerException` (`NPE`).
2) the discussion concludes that code had to be changed semantically to perform a corrective maintenance task.
3) it fixes runtime or memory issues cause by defects (e.g. endless loops).

**Example:** Tomcat5 report 28147 \(^1\) is categorized as **RFE** but reports a bug that causes a “JasperException for jsp files that are symbolic links”. The underlying issue was that tomcat used canonical instead of absolute paths. The applied fix touches one line replacing one method invocation. According to Rule 2, we classified the applied code change as a corrective maintenance task and thus the issue report as **BUG**.

A report is categorized as **RFE** (Feature Request) if...

1) it requests to implement a new access/getter method.
2) it requests to add new functionality.
3) it requests to support new object types, specifications, or standards.

**Example:** Lucene-Java report LUCENE-2074 \(^2\) is categorized as **BUG**. But the applied patch and the discussion unveil that a new versioning mechanism had to be implemented. The first comment by Uwe Schindler makes it explicit: “Here the patch. It uses an interface containing the needed methods to easily switch between both impl. The old one was deprecated [...]”. This is reclassified as **RFE** by Rule 2.
Appendix

Classification rules (Cont’d)

A report is categorized as IMPR (Improvement Request) if...

1) it discusses resource issues (time, memory) caused by non optimal algorithms or garbage collection strategies.
2) it discusses semantics-preserving changes (typos, formatting) to code, log messages, exception messages, or property fields.
3) it requests more or fewer log messages.
4) it requests changing the content of log messages.
5) it requests changing the type and/or the message of Exceptions to be thrown.
6) it requests changes supporting new input or output formats (e.g. for backward compatibility or user satisfaction).
7) it introduces concurrent versions of already existent functionalities.
8) it suggests upgrading or patching third party libraries to overcome issues caused by third party libraries.
9) it requests changes that correct/synchronize an already implemented feature according to specification/documentation.

Example: Jackrabbit report JCR-2892³ is filed as BUG under the title “Large fetch sizes have potentially deleterious effects on VM memory requirements when using Oracle”. The algorithm fetches data from a database with a large amount of columns and rows, which caused the Oracle driver to allocate a large buffer. The resolution was to develop a new algorithm consuming less memory. This is an IMPR according to Rule 1 since no new functionality was implemented and since the program did not contain any defect.
Appendix

Classification rules (Cont’d)

A report is categorized as **DOC** (Documentation Request) if...

1) its discussion unveils that the report was filed due to missing, ambiguous, or outdated documentation.

*Example: Tomcat5 bug report 30048* fixes the problem “Setting compressableMimeTypes is ignored.” by “Docs updated in CVS to reflect correct spelling.” This is a **DOC**.

A report is categorized as **REFAC** (Refactoring Request) if...

1) it requests to move code into other packages, classes, or methods.
2) it requests to rename variables, methods, classes, packages, or configuration options.

*Example: Tomcat5 report 28286* is filed as **BUG** and contains a patch adding a new interface SSOValve. But in comment 4, Remy Maucherat refuses to apply the patch and the idea to introduce a new interface. Instead, he commits a patch that refactors class AuthenticatorBase to allow subclassing. This is a **REFAC** as per Rule 2.

A report is categorized as **OTHER** if...

1) it reports violations of JAVA contracts without causing failures (e.g. “`equals()` but no `hashCode()`”).
2) complains about compatibility fixes (e.g. “should compile with GCJ”).
3) the task does not require changing source or documentation (like packaging, configuration, download, etc.)

*Example: Lucene-Java report LUCENE-1893* complains that “classes implement `equals()` but not `hashCode()`”. This violated JAVA contracts but does not cause failures. Lucene-Java report LUCENE-289* requests “better support gcj compilation”. According to our rules this is considered to be an compatibility improvement classified as **OTHER**.
num_original_bugs: # of distinct issue reports originally classified as BUG  
num_classified_bugs: # of distinct issue reports that were classified as BUG during manual inspection

MappingBiasRate: percentage of false positive original BUG reports that could be mapped to code files  
DiffBugNumRate: # of files for which num_original_bugs-num_classified_bugs!=0  
MissDefectRate: numMissDefect / numZeroOriginalDefect  
FalseDefectRate: numFalseDefect / numOriginalDefect

numMissDefect: # of source files for which no original bug report could be mapped but that have at least one manually classified bug report assigned  
numZeroOriginalDefect: # of source files having no original bug report assigned  
numFalseDefect: # of source files with at least one original bug report assigned but no manually classified bug report  
numOriginalDefect: # of source files that got at least one original bug report assigned