An Empirical Study on Reducing Omission Errors in Practice

Jihun Park¹, Miryung Kim², Doo-Hwan Bae¹

1. KAIST, South Korea
2. University of California, Los Angeles (UCLA), USA
Can we predict an additional change location in a transaction?

- Change coupling (mining SW repositories): Zimmermann et al., Ying et al., Hassan and Holt, Herzig and Zeller
- Structural dependency: Robillard, Saul et al.
- Cloning-based relationship: Nguyen et al.
Predicting omission errors

How can we predict the supplementary change location, given the initial change location?
Key contributions

• To systematically investigate a real-world supplementary patch data set, we suggest a graph representation *change relationship graph (CRG)*.

1. While a single trait is inadequate, combining multiple traits is limited as well.
2. A boosting approach does not significantly improve the accuracy.
3. There is no package or developer specific pattern.
4. There is no repeated mistake.
Change Relationship Graph (CRG)

- Graph Nodes
  - Classes
  - Methods
- Graph Edges
  - Extends
  - Contains
  - Method invocation (calls, called by)
  - Historical co-change
  - Code clone
  - Name similarity

Study subjects: Eclipse JDT core, Eclipse SWT, and Equinox p2

- Class
  - contains
  - contains
  - Code clone
  - An initial change location
  - Method
  - The supplementary change location

* M.K. Ripon Saha et al. A graph-based framework for reasoning about relationships among software modifications. TR 2014
Observation 1: While a single trait is inadequate, combining multiple traits is limited as well.

- Only 10% to 20% of supplementary change locations can be connected with one edge from initial change location.
- Combining multiple traits as a prediction rule shows at most 10% accuracy.
Observation 2: A boosting approach does not improve the accuracy.

- We design a boosting approach that sums up trained accuracy of rules connecting initial and supplementary change locations to calculate **prediction score**

- This approach cannot accurately predict supplementary change location (at most 7% precision).

**Boosting approach based on the past prediction accuracy also cannot accurately predict supplementary change locations.**
Observation 3: There is no package or developer specific pattern.

- Package or developer specific rules might improve the prediction accuracy.
  - Package A
    - Accuracy of code clone: 40%
    - Accuracy of co-change: 10%

- We make boosting approaches based on package and developer specific prediction rules.

- The improvements is negligible; the highest accuracy improvement is only 1.2%

No package or developer specific pattern between initial and supplementary change locations exists.
Observation 4: There is no repeated mistake.

- There might be an uncovered relationship which can result in *repeated patterns*.

- The majority of patterns (78% ~ 96%) appear only once.
- 69% to 84% of initial change locations appear only once.

Developers rarely make repeated mistakes at the same location.
Conclusion

• We systematically study omission errors using a real-world supplementary patch data set.

• Version history based pattern mining cannot be accurate at finding supplementary change locations.

• Past prediction accuracy, and package or developer specific information does not help.

• We share our skepticism that reducing real-world omission errors is inherently challenging.
Thank you for listening

An Empirical Study on Reducing Omission Errors in Practice

Jihun Park¹, Miryung Kim², Doo-Hwan Bae¹

1. KAIST, South Korea
2. University of California, Los Angeles (UCLA), USA
Supplementary Data Set

The bug IDs that were mentioned only one commit.

The bug IDs that were mentioned in multiple fix revisions.

Bug reports

Type 1 bug

Bug 22

... Fix #22

Type 2 bug

Bug 31

... Fix #31

Fix commits

An initial (incomplete) patch

Supplementary patches

Development history

• We use Eclipse JDT core, Eclipse SWT, and Equinox p2
• Total 16 years, 13259 bugs (24.8% are Type 2 bugs on average)
## Subject projects

<table>
<thead>
<tr>
<th></th>
<th>Eclipse JDT core</th>
<th>Eclipse SWT</th>
<th>Equinox p2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Study period</td>
<td>2001/06 ~ 2007/12</td>
<td>2001/05 ~ 2008/12</td>
<td>2006/01 ~ 2009/12</td>
</tr>
<tr>
<td>Total revisions</td>
<td>17009 revisions</td>
<td>21530 revisions</td>
<td>6761 revisions</td>
</tr>
<tr>
<td># of bugs</td>
<td>1812</td>
<td>1256</td>
<td>1783</td>
</tr>
<tr>
<td>Type 1 bugs</td>
<td>2930 (77.04%)</td>
<td>3458 (74.00%)</td>
<td>1328 (74.48%)</td>
</tr>
<tr>
<td>Type 2 bugs</td>
<td>873 (22.96%)</td>
<td>1215 (26.00%)</td>
<td>455 (25.52%)</td>
</tr>
</tbody>
</table>
Evaluating a prediction method

• Precision, recall, and f-score
  – Predicted set \( P \) and Suggested set \( S \)
  – \( \text{Precision} = \frac{|P \cap S|}{|P|}, \text{Recall} = \frac{|P \cap S|}{|S|} \)
  – \( F - \text{score} = \frac{2 \times \text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \)

• Feedback
  – What portion of initial changes can obtain at least one suggestion?
  – \( P^m_b \) is derived using a prediction method \( m \) for bug \( b \),
  – \( \text{Feedback} = \frac{|\{ b \in \text{Typellbugs} \mid 1 \leq |P^m_b| \}|}{|\{\text{Typellbugs}\}|} \)