An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution

International Conference on Software Engineering (2011)
Miryung Kim, Dongxiang Cai, Sunghun Kim

2013-09-10
Yoo Jin Lim

© KAIST SE LAB 2013
Refactoring

- Changes program’s internal structure without modifying its external functional behavior
- Improves software quality in software evolution
  - Making it easier to understand, maintain, and modify software

API-level refactoring

- Rename
- Move
- Signature changes
Hypotheses (Questions) about API-level Refactoring

H1: There are **more bug fixes** afterwards.  
→ Does the bug fix rate increase after refactorings?

H2: Refactoring improves developer’s **productivity**.  
→ Does the bug fix time decrease after refactorings?

H3: **If H1&2 are true, refactoring facilitates bug fixes.**  
→ Is a large number of refactoring revisions also including bug fixes or related to later bug fixes?

H4: Developers **avoid it if under time constraints.**  
→ Are there relatively fewer refactorings before major software release dates?
Introduction (3/3)

❖ Motivation

 The role of refactorings during software evolution has not been systematically investigated
  • Questions remain as to the true benefits of refactorings in terms of bug fixing and developer productivity.

❖ Goal

 Investigate the role of refactorings, using fine-grained revision history at the method level.
  • Examine the relationships between API-level refactorings, bug fixes, the time to resolve bugs, and release cycles.
# Related Work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Focus</strong></td>
<td>Relationship between refactorings and bugs</td>
<td>Relationship between refactorings and bugs</td>
</tr>
<tr>
<td><strong>Analysis Method</strong></td>
<td>Use N-day time window</td>
<td>Use time periods</td>
</tr>
<tr>
<td><strong>Result</strong></td>
<td>Number of bugs increases after refactorings</td>
<td>Number of bugs decreases after refactorings</td>
</tr>
</tbody>
</table>

- **This Study (Kim et al.)**
  - **Focus**: Relationship between API-level refactorings and bug fixes
    - ➔ bug fixes, bug fix times, and release cycles.
  - **Method**: Use K-revision sliding window and method-level locations
    - ➔ examine their relationship more accurately

ⓒ KAIST SE LAB 2013 6/27
Overall Approach

Q1: Are there more bug fixes after API-level refactorings?

Q2: Do refactorings improve productivity?

Q3: Do refactorings facilitate bug fixes?

Q4: Are there relatively fewer before major releases?
Subject programs

- Eclipse JDT, jEdit, and Columba
  - Large open-source software with high quality change logs

<table>
<thead>
<tr>
<th></th>
<th>Eclipse JDT core</th>
<th>jEdit</th>
<th>Columba</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>IDE</td>
<td>Text editor</td>
<td>Email client</td>
</tr>
<tr>
<td># Revisions</td>
<td>15000</td>
<td>11102</td>
<td>421</td>
</tr>
<tr>
<td># Refactoring Revisions</td>
<td>1089</td>
<td>423</td>
<td>36</td>
</tr>
<tr>
<td># Bug Fix Revisions</td>
<td>3752</td>
<td>1073</td>
<td>150</td>
</tr>
</tbody>
</table>
Identification of API-level Refactoring Revisions

- Automatic refactoring reconstruction technique (M. Kim et al.)
  - Limited to API-level = changes at or above method header level

---

```c
// Version 1
class X {
    int Sub() {...};
    int Add() {...};
}

class Y {
    int Add_one() {...}
}

// Version 2
class X {
    int Subtract();
}

class Y {
    int Add() {...};
    int Add_one(int a) {...}
}
```
Identification of Bug Fix Revisions

- Bug history extraction technique (S.Kim et al.)
  - Heuristically mine change log comments for bug fix keywords

Revision 1.1
Date: 9/1/12
Name: Abe
Line #’s: 4
Comment: Changed M1(), see bug ID#42233

Revision 1.2
Date: 9/5/12
Name: Bob
Line #’s: 56 ~ 58
Comment: Updated license

Revision 1.3
Date: 9/15/12
Name: Cho
Line #’s: 9
Comment: Fixed M1()
Study Approach (4/6)

- Identification of Bug-introducing Changes
  - Tracing back in the revision history (Sliwerski et al.)
    - Use Diff outputs to track down the origin of flawed change
    - Identify Bug FixTime in days

<table>
<thead>
<tr>
<th>Revision 1.1</th>
<th>Revision 1.2</th>
<th>Revision 1.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date: 9/1/12</td>
<td>Date: 9/5/12</td>
<td>Date: 9/15/12</td>
</tr>
<tr>
<td>Name: Abe</td>
<td>Name: Bob</td>
<td>Name: Cho</td>
</tr>
<tr>
<td>Line #'s: 4</td>
<td>Line #'s: 56</td>
<td>Line #'s: 9</td>
</tr>
<tr>
<td>Comment: Changed M1(), see bug ID#42233</td>
<td>Comment: Updated license</td>
<td>Comment: Fixed M1()</td>
</tr>
</tbody>
</table>

Diff
Diff
Change Distilling

- Map the **line-level** location of a bug fix to its container **method**
  - ➔ To compare with API-level refactoring location

```java
public void Method2()
{
    ...
    Bug_Fixed();
    ...
}
```
Accuracy of MK and SK Techniques

Automated techniques Vs. Manual inspection

<table>
<thead>
<tr>
<th>Source</th>
<th>#</th>
<th>Prec.</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>API Refactoring Revision</strong></td>
<td>14</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>[MK] API-level refactoring detection</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspection of change logs only</td>
<td>2</td>
<td>0.50</td>
<td>0.07</td>
</tr>
<tr>
<td>Inspection of bug reports only</td>
<td>5</td>
<td>0.75</td>
<td>0.21</td>
</tr>
<tr>
<td><strong>Fix Revision</strong></td>
<td>62</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[SK] Fix-revision detection</td>
<td>49</td>
<td>0.96</td>
<td>0.76</td>
</tr>
<tr>
<td>Inspection of change logs only</td>
<td>55</td>
<td>0.92</td>
<td>0.81</td>
</tr>
<tr>
<td>Inspection of bug reports only</td>
<td>49</td>
<td>0.94</td>
<td>0.74</td>
</tr>
</tbody>
</table>

Overall, this study’s revision data are accurate enough
Q1: Are there more bug fixes after API-level refactorings?

- Compute a fix rate, \( \frac{|fix \ revisions|}{K} \), within K sliding window before and after each refactoring revision.

- Compute the fix rate for each method.
Q1: Are there more bug fixes after API-level refactorings?
- Fix rates increase after refactorings
Q1: Are there more bug fixes after API-level refactorings?
- Fix rates also increase after refactorings at the method level

There is a short-term increase in the number of bug fixes after refactorings.
Q1: Are there more bug fixes after API-level refactorings?

- Find refactoring revisions with at least one bug fix applied to the same method within K=20.
- Bugs introduced, which get fixed later
- Developers helped in identifying bugs
- Randomly select 50 of them for manual investigation

<table>
<thead>
<tr>
<th>Types</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Refactoring with a bug fix is <strong>incomplete</strong>, inducing a later fix.</td>
<td>50%</td>
</tr>
<tr>
<td>2 <strong>Incorrect</strong> refactorings cause a bug, inducing a later fix.</td>
<td>6%</td>
</tr>
<tr>
<td>3 To fix several related bugs, developer <strong>first refactors</strong>.</td>
<td>6%</td>
</tr>
<tr>
<td>4 After refactorings, developer <strong>discovers</strong> a new bug and fixes it.</td>
<td>2%</td>
</tr>
<tr>
<td>5 Bug fix happened after refactoring in the same location, but they are <strong>not related</strong>.</td>
<td>36%</td>
</tr>
</tbody>
</table>
Q2: Do refactorings improve developer productivity?

- Estimate time taken to fix bugs near the refactoring timing
  - Categorized into four classes based on Before and After refactoring

- Calculate time in each category for each refactoring using $K = 100$
Q2: Do refactorings improve developer productivity?

WHY?
- Fix is easier since code is refactored
- Refactoring mistakes take quick and easy supplementary fixes

The average fix time tends to decrease after refactorings.
Q3: Do API-level refactorings facilitate bug fixes?
- Are many refactorings done as part of bug fix or related to later fix?
  - Identify revisions including both refactoring and bug fix

Fixes and refactorings often appear in the same revision.
Results (8/11)

Q3: Do API-level refactorings facilitate bug fixes?

- Are many refactorings done as part of bug fix or related to later fix?
  - Find refactoring revisions with at least one bug fix applied to same method within K=20.

Why?

- Refactorings applied to fix hard bugs
- Refactoring mistakes corrected later

Refactoring are followed by related bug fixes more often than non-refactoring.
Q4: Are there relatively fewer API-level refactorings before major releases?

- Measure refactorings rates before and after each release.
  - 15 major releases in Eclipse JDT and 4 in jEdit
- Measure bug fix rates also

Results (9/11)
Q4: Are there relatively fewer API-level refactorings before major releases?

- Result 1: Bug fix rate
  - No more adding new features
  - Focus on bug fixing
Q4: Are there relatively fewer API-level refactorings before major releases?
- Result 2: Refactoring rate

WHY?
- Developers apply refactorings to implement bug fixes to be shipped with a new release

There are more refactorings and bug fixes before major version releases.
Refactorings occur more frequently before major releases than after.

Results Overview

Q1&2: What happens to Bug Fixes After Refactorings?
- There is an increase in bug fixes.
- Bug fix time is shorter.

Motivated to ask

Q3: Do Refactorings Facilitate Bug Fixes?
- Refactoring revisions are related to later bug fixes.
- Refactoring revisions often include bug fixes at the same time.

Q4: Are there Fewer Before Releases?
- Refactorings occur more frequently before major releases than after.
- Developers apply refactorings to implement bug fixes to be shipped with new release.
Conclusion

❖ Contributions
  ▪ Revealed interesting relationships between refactorings and bug fixes
    • API-level refactoring serves the role of both facilitating bug fixes and inducing bugs

❖ Future Work
  ▪ Cost-benefit analysis of refactoring investment
  ▪ Development of new tools that correct inconsistent refactorings
Discussion

Limitations

- Refactoring reconstruction coverage and accuracy
  - Only captures a small subset of refactorings

- Phases and activity levels
  - K-sliding window method may be an inconsistent map

- Development practice
  - A few developers’ practices may have strongly affected study results
Thank You.
Appendix

- Bug Reports

![Diagram showing BugDB, Bug 42233, and Fixed Bug 42233 linked to Change History with version numbers 1.17, 1.18, and 1.19. The diagram emphasizes linking bugs with changes.]
Appendix

- Identification of Bug-introducing Changes

Revision 1.1
Date: 9/1/12
Name: Abe
Line #'s: 4
Comment: Changed M1(), see bug ID#42233

Revision 1.2
Date: 9/5/12
Name: Bob
Line #'s: 56 ~ 58
Comment: Updated license

Revision 1.3
Date: 9/15/12
Name: Cho
Line #'s: 9
Comment: Fixed M1()

---

8 : 1.0 public int M1(){
9 : 1.1    return i/0;
...        ...

SVN blame

© KAIST SE LAB 2013
Appendix

Accuracy of MK and SK Techniques

- Precision
  - Percentage of identified fix revisions which are actual fix revisions

- Recall
  - Percentage of actual fix revisions which are identified by the tool

Automated techniques Vs. Manual inspection

<table>
<thead>
<tr>
<th>Source</th>
<th>Manual Count</th>
<th>#</th>
<th>Prec.</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>[MK] API-level refactoring detection</td>
<td>14</td>
<td>14</td>
<td>0.93</td>
<td>0.93</td>
</tr>
<tr>
<td>Inspection of change logs only</td>
<td>2</td>
<td>2</td>
<td>0.50</td>
<td>0.07</td>
</tr>
<tr>
<td>Inspection of bug reports only</td>
<td>5</td>
<td>5</td>
<td>0.75</td>
<td>0.21</td>
</tr>
<tr>
<td>[SK] Fix-revision detection</td>
<td>62</td>
<td>49</td>
<td>0.96</td>
<td>0.76</td>
</tr>
<tr>
<td>Inspection of change logs only</td>
<td>55</td>
<td>55</td>
<td>0.92</td>
<td>0.81</td>
</tr>
<tr>
<td>Inspection of bug reports only</td>
<td>49</td>
<td>49</td>
<td>0.94</td>
<td>0.74</td>
</tr>
</tbody>
</table>
Appendix

Q1 result: Latent bugs

- A bug which is not fixed and is already uncovered in earlier versions and is still present.
- Bugs that remain dormant are latent bugs. These bugs are not found until one or more releases of the product.
Appendix

Q3: Do API-level refactorings facilitate bug fixes?

Q3.1: How many refactorings are done as part of bug fix?
- Identify revisions including both refactoring and bug fix

| Subject    | P(F) | P(R) | P(R|F) | P(R|not F) | P(F|R) | P(F|not R) |
|------------|------|------|--------|------------|--------|------------|
| Eclipse JDT | 25.0%| 7.3% | 12.0%  | 5.7%       | 41.5%  | 23.7%      |
| JEdit      | 9.7% | 3.8% | 11.5%  | 3.0%       | 29.1%  | 8.9%       |
| Columba    | 35.7%| 8.6% | 10.7%  | 7.4%       | 44.4%  | 34.8%      |

Probability of fixes and refactorings at revision level

| Subject    | P(F) | P(R) | P(R|F) | P(R|not F) | P(F|R) | P(F|not R) |
|------------|------|------|--------|------------|--------|------------|
| Eclipse JDT | 54.9%| 28.6%| 32.7%  | 23.6%      | 62.8%  | 51.7%      |
| JEdit      | 41.4%| 39.5%| 31.5%  | 45.1%      | 33.0%  | 46.9%      |
| Columba    | 34.0%| 19.4%| 13.8%  | 22.2%      | 24.3%  | 36.3%      |

Probability of fixes and refactorings at method level

Fixes and refactorings often appear in the same revision.