A Field Study of Refactoring Benefits and Challenges

Miryung Kim  
University of Texas at Austin

Thomas Zimmermann, Nachiappan Nagappan  
Microsoft Research
Contradicting Beliefs on Refactoring Benefits

- Refactoring improves **software quality** and **maintainability**
- A lack of refactoring incurs **technical debt**

**VS.**

- Refactorings do not provide immediate benefits unlike bug fixes and new features
Conflicting Evidences on Refactoring Benefits

- **Bug fix time decreases** after refactoring [Carriere et al.]
- **Defect density decreases** after refactoring [Ratzinger et al.]

**VS.**

- Inconsistent refactorings **cause bugs** [Görg and Weiβgerber, Kim et al.]
- Code **churns** are correlated with defect density [Nagappan & Ball]
Key Findings

- Refactoring is not confined to behavior preserving transformation.
- Developers perceive that refactoring involves **substantial cost** and **risk**.
- Refactored modules experienced significant reduction in inter-module dependencies and post-release **defects**.
Outline

- A Survey of Refactoring Practices
- Interviews with Windows Refactoring Team
- Quantitative Analysis of Windows 7 Version History
Survey Participants

- **Target:** 1290 engineers whose check-in comments include a keyword ‘refactor*’ in the last 2 years
  - Windows, exchange, ocs, office, Win7mobile,

- **Participants:** 328 engineers
  - 6.35 years at MS
  - 9.74 years in software industry

- 22 multiple choice and free form questions
Finding 1. Refactoring is not confined to behavior-preserving transformations

- 46% did not mention preservation of behavior, semantics, or functionality
- 78% define refactoring improves *some aspects of program behavior*
- 71% said basic refactorings are often a *part of larger, architecture level effort*
29% pointed out a lack of support for refactoring integration, code reviews targeting refactoring edits, and custom refactoring engine.

“Cross-branch integration was the biggest problem.”

“Refactoring typically increases the number of lines involved in a check-in. That burdens code reviewers.”

When a regression test suite is inadequate, there is no safety net for checking the correctness of refactoring.
Finding 3. Refactoring engines are not used much

- Developers do 86% of refactorings manually, despite awareness of automated tools.
Finding 4. Refactoring is driven by immediate, concrete needs.

- **46%** refactor code as a part of bug fixes and feature additions.
- **More than 95%** of developers refactor code across all milestones not only in quality milestones (MQ).
Finding 5. Refactoring involves substantial cost and risks

- 75% perceive that refactoring has a risk of functionality regression and **bugs**.

![Bar chart showing various costs and risks associated with refactoring.](image-url)
Outline

A Survey of Refactoring Practices

Interviews with Windows Refactoring Team

Quantitative Analysis of Windows 7 Version History
Details on Interviewees

- Architect (90 mins)
- Architect / Dev Manager (30 mins)
- Dev Team Lead (75 mins)
- Dev Team Lead (85 mins)
- Developer (75 mins)
- Researcher (60 mins)
A designated team initiated refactoring effort to improve modularity and parallel development efficiency.

Driven by foresights to repurpose Windows to target different execution environments.

Conducted analysis of de-facto dependency structure and created a “layer map”.

Developed custom tools and processes such as MaX and “quality gate check” [Srivastava et al.]
Outline

- A Survey of Refactoring Practices
- Interviews with Windows Refactoring Team
- Quantitative Analysis of Windows 7 Version History
Research Questions

- Q1: Where was Windows 7 refactoring effort focused on?
- Q2: Did refactoring reduce binary-level dependencies?
- Q3: Are refactored binaries more defect-prone than non-refactored binaries?
- Q4: Did refactoring reduce post-release defects?
Windows 7 Refactoring Study Method

perf_dev_foo

refactor_dev

refactor

media_core

winmain
Identified branches where the refactoring team made frequent commits
The refactoring team confirmed refactoring branches
Categorize all Windows 7 commits into refactorings vs. non-refactorings
**Windows 7 Refactoring Study Method**

<table>
<thead>
<tr>
<th></th>
<th>refactor</th>
<th>Non-refactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOO32.dll</td>
<td><img src="image" alt="Blue Circles" /></td>
<td><img src="image" alt="Red Circles" /></td>
</tr>
<tr>
<td>FOO.dll</td>
<td><img src="image" alt="Blue Circles" /></td>
<td><img src="image" alt="Red Circles" /></td>
</tr>
<tr>
<td>...</td>
<td><img src="image" alt="Blue Circle" /></td>
<td><img src="image" alt="Red Circles" /></td>
</tr>
</tbody>
</table>

Map commits to DLLs (binary modules)
## Windows 7 Refactoring Changes

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Refactor Branches</th>
<th>Non Refactor Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commits</td>
<td>1.27%</td>
<td>98.73%</td>
</tr>
<tr>
<td>Authors</td>
<td>2.04%</td>
<td>99.84%</td>
</tr>
<tr>
<td>Binary Modules</td>
<td>94.64%</td>
<td>99.05%</td>
</tr>
</tbody>
</table>
Q1. Where was the refactoring effort focused on?

Top 25% of most frequently refactored DLLs cover 53% of all neighboring dependency counts in Vista for modified DLLs.
Q2. Did refactoring reduce binary-level dependencies?

Cumulative Dependency Ratio
(Only modified DLLs are considered)

Top 12.8% of refactored DLLs

Ratio of Refactored DLLs
Q3. Are refactored binaries more likely defect-prone than non-refactored binaries?

No, Top 20% of most frequently refactored DLLs are responsible for 42% of all Win 7 post release defects, while top 20% of most modified DLLs are responsible for 55%.
Q4. Did refactoring reduce post release defects more?

Reduction of Post-Release Defects (Vista vs. Win 7)

-112.2%  -104.4%  -97.8%  -89.1%  -100.8%

Top 25%  Top 25 to 50%  Top 50 to 75%  Top 75 to 100%  All DLLs
We present a **three-pronged view** of refactoring in a **large company** through a survey, interviews, and version history analysis.

The **definition** of refactoring in practice is broader than behavior-preserving program transformations.

Developers perceive that refactoring involves **substantial cost and risks**.

Developers need various types of tool support **beyond automated refactoring** within IDEs.
Centralized, system-wide refactoring was facilitated by custom tools and processes such as MaX and quality gate check.

Refactored modules experienced higher reduction in the number of inter-module dependencies and post-release defects than other changed modules.
Anonymous survey and interview participants

Thanks to Galen Hunt, Chris Bird, Mike Barnett, Tom Ball, Rob DeLine, Andy Begel, ESE and RISE friends at MSR

This research is in part supported by National Science Foundation, CAREER-1117902, CCF-1149391, and CCF-1043810 and Microsoft SEIF award.

Thank You!