Objectives

• Refactoring
• Wrapping up Software Tools

“How We Refactor, and How We Know It”

Murphy-Hill, Parnin, Black
ICSE 2009

• Problem?
  ➢ Limitations of state of the art
  ➢ Goals

Goals

• Study refactoring from several perspectives
• Confirm or refute previous assumptions/results
• Provide results to guide tool developers in developing new refactoring tools

Introduction to Refactoring

Refactoring: a change in code that does not affect program behavior

Goal: Code is easier to read/understand/extend/use
Introduction to Refactoring

```
class Foo {
}
class Bar {
    int a;
    public Bar(int a) {
        this.a = a;
    }
    public static Bar create(int a){
        return new Bar(a);
    }
}
```

Correctness & Speed

Murphy-Hill, ICSE 09

“How We Refactor, and How We Know It”

• Problem?
  ➤ Limitations of state of the art
  ➤ Goals
• Approach?
  ➤ Benefit?

May 29, 09
Smyrkko - CS293

Approach

• Study four sets of user data about refactoring

<table>
<thead>
<tr>
<th>Name</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users</td>
<td>• Mylyn Monitor tool</td>
</tr>
<tr>
<td></td>
<td>• 43 programmers in Eclipse</td>
</tr>
<tr>
<td></td>
<td>• Eclipse commands</td>
</tr>
<tr>
<td>Everyone</td>
<td>• Eclipse Usage Collector</td>
</tr>
<tr>
<td></td>
<td>• 130,000 Java developers</td>
</tr>
<tr>
<td></td>
<td>• Eclipse commands</td>
</tr>
<tr>
<td>Toolsmiths</td>
<td>• Refactoring histories</td>
</tr>
<tr>
<td></td>
<td>• 4 developers of Eclipse refactoring tools</td>
</tr>
<tr>
<td>Eclipse CVS</td>
<td>• Version history of Eclipse and Junit</td>
</tr>
<tr>
<td></td>
<td>• Same developers as Toolsmiths</td>
</tr>
</tbody>
</table>

May 29, 09
Smyrkko - CS293

“How We Refactor, and How We Know It”

• Problem?
  ➤ Limitations of state of the art
  ➤ Goals
• Approach?
  ➤ Benefit?
• Evaluation?
  ➤ Results?

Most surprising/interesting/important result?

May 29, 09
Smyrkko - CS293
3 Research Questions

Do Programmers Usually Floss Refactor?  
- root-canal refactoring  
- Do Programmers Refactor Often?  
- Do Programmers Use Refactoring Tools Often?  

Do Programmers Refactor Often?  

Do Programmers Use Refactoring Tools Often?  

Importance

• If floss refactoring is more common, then tools should support floss refactoring

A tool user interface optimized for "root canals"  
A tool user interface optimized for "flossings"

What We Already Know

Floss refactoring better by prescription:

- In almost all cases, I'm opposed to setting aside time for refactoring. In my view refactoring is not an activity you set aside time to do. Refactoring is something you do all the time in little bursts.  
  ~ Martin Fowler

- Avoid the temptation to stop work and refactor for several weeks... Have your team get used to refactoring as part of their daily work.  
  ~ James Shore

Case studies describe root canal refactorings:

- Pizka [2004]
- Bourqun and Keller [2007]

What they did and results

91% of refactorings occurred during floss refactoring  
65% of Commits were Floss Refactoring in HC

Find the differences:

Root Canal  
(Fpure) Refactoring  
Floss (Mixed) Refactoring

3 Research Questions

Do Programmers Usually Floss Refactor?  
- root-canal refactoring  
- Do Programmers Refactor Often?  
- Do Programmers Use Refactoring Tools Often?
If programmers refactor often:

- refactoring research matters
- refactoring tools may significantly speed up and improve the correctness of programming

According to Xing and Stroulia [2006]:

Refactoring is frequent in the Eclipse project

But their use of an automated detection tool means the results are only a rough estimate

But this is tool-based refactoring. If many refactorings are done without tools, then refactoring is even more frequent.

Do Programmers Usually Floss Refactor?

Yes

Do Programmers Refactor Often?

Yes

Do Programmers Use Refactoring Tools Often?

If many refactorings are done without tools, then refactoring is even more frequent.

From their previous work:

2 in 16 programming students used them (and even then, 20% and 60% of the time)

Agile developers (in 2007, n=112) estimate that they use them only 68% of the time

6 of 42 people who used Eclipse on networked PSU computers used refactoring tools

If programmers underuse refactoring tools:

- potential for tool improvements
- programmers are needlessly introducing errors or refactoring slowly
Do Programmers Use Refactoring Tools Often?

What They Did

- Dev. 1 Commit
  - Feb. 1, 15:01

- Dev. 1 Commit
  - Feb. 1, 15:23

Jan. 29, 08:40

Dev. 2

Feb. 1, 15:11

Dev. 1

Feb. 1, 15:59

Feb. 1, 16:00

Feb. 1, 12:23

Do Programmers Use Refactoring Tools Often?

Results

- 89% of refactorings were done without a tool among toolsmiths.

3 Research Questions

- Do Programmers Usually Floss Refactor?
  - Yes

- Do Programmers Refactor Often?
  - Yes

- Do Programmers Use Refactoring Tools Often?
  - No

6 More Findings

The kind of refactoring performed with tools differs from the kind performed manually.

Toolsmiths use a wider array of refactoring tools than tool users.

About 40% of tool-initiated refactorings occur in batches.

About 90% of tool-initiated refactorings do not require configuration of the tool.

Messages written by programmers in version histories are unreliable indicators of refactoring.

About 40% of refactorings will not be detected by most mining tools that detect refactoring from version histories.

“How We Refactor, and How We Know It”

- Problem?
  - Limitations of state of the art
  - Goals

- Approach?
  - Benefit?

- Evaluation?
  - Results?

- Limitations?

- Conclusions?
Limitations

- Subject (data) issues
  - May not be representative
  - May not be comprehensive
  - Manual determination may be inaccurate (conservative/optimistic)

Work in Progress

- "Making Refactoring Tools Part of the Programming Workflow"
  - By Emerson Murphy-Hill and Andrew P. Black

  Refactoring is a frequent practice, but tools that automate refactoring are seldom used; this is a problem because manual refactoring is slow and error-prone. One reason for the underuse is that the tools have poor usability: instead of fitting into programmers’ workflow, the tools get in the way. We propose guidelines for improving the usability of refactoring tools, and then apply these guidelines to the design of two independent user interfaces for tool activation—interfaces that are designed to make the tool part of the programming workflow.

Conclusion

- The more ways we look at how people refactor, the more confidence we have that we understand how people refactor
- In studying refactoring from new perspectives, our findings have confirmed – and refuted – previous knowledge about how people refactor

Discussion

- All their hypotheses were validated?
- Take a closer look at configuration hypothesis
- Composite refactoring

Why I Chose This Paper

- One of four best paper awards at ICSE 2009
  - Just two weeks ago! Hot off the presses!
- Different type of paper/research
  - Analyze how people use tools
  - Based on analysis → how to improve tools
  - Based on a couple large data sets
- Contradicted previous research
  - Better idea of what questions to ask about research
  - Healthy skepticism
- Pulled together many things we’ve talked about this semester
  - Eclipse (usage data), CVS, Mylyn
  - How to build useful tools that people will use

CONCLUDING SOFTWARE TOOLS
Our Focus

• How do tools fit into the model?
  ➢ How do they help us?
  ➢ What are tools’ strengths and limitations?

• Automate common and/or tedious tasks
  ➢ More efficient
  ➢ Improve productivity
  ➢ Get “better” results

• Still need to understand concepts/reason for the tool
• Think for us!
  ➢ Still need to design code to be robust/adaptable/efficient
  ➢ Still need to code the logic
  ➢ Still need to try to prevent bugs
• Just because we can do something with a tool doesn’t always mean we should

Goal: Productivity

• Many available tools
  ➢ UNIX & UNIX-like systems (e.g., Linux)
  ➢ Open-source (Gnu, Apache, Eclipse)
  ➢ Proprietary
  ➢ Variety of purposes
• Know what (free) tools are available, what they do, how to use them
Goal: Automation
- Often have to do a task over and over again
  - Time-intensive to do by hand
  - Shortcuts aren’t enough
- What we want
  - Tools to make tasks easier
  - Scripts to be able to repeat the tasks easier

Course Objectives
- At the end of this course, you will be able to
  - Use a variety of Unix tools
  - Apply a variety of tools to automate many tasks
  - Describe the use of state-of-the-art software tools for developing and maintaining large software systems, based on hands-on experience
  - Discuss when best to use different tools, the limitations of the tools, and what they have to offer
  - Discuss the challenges and strategies in building software tools
  - Communicate technical content in both oral and written forms

Non-Syllabus Goals
- Improve your productivity
- Unix confidence/proficiency
  - To intermediate user
- Tool confidence
  - Less intimidated by installing, learning new tools
- Resume builder!
  - Impress potential employers, advisors

Before End of Finals
- Complete survey

COOKIE FEST!!!