How We Refactor, and How We Know It

Emerson Murphy-Hill, Chris Parnin, Andrew P. Black.
Published in year 2009
Presented by Sandarbh Bhadauria
Overview

• Previous research on how programmers refactor in the wild is based on studies that examine just a few software projects.
• The assumption made in previous researches have rarely been put to test by attempting to replicate the earlier studies.
• This paper attempts to put refactoring research on scientific basis, authors conducted research using 4 data sets. Spanning more than 13 000 developers, 240 000 tool-assisted refactorings, 2500 developer hours, and 3400 version control commits.
• Found evidence in support of some of the assumptions and against some.
Introduction

• Refactoring -
  • Refactoring is a controlled technique for improving the design of an existing code base
    – M. Fowler
Example

```java
package poclasses;

public class LoginAction {

    public String findUser(String userId) {
        // check empty
        if (userId == null || userId.isEmpty()) {
            return "User input missing";
        }
        // check for sql injection
        String Quotes = "";
        if (userId.contains(Quotes)) {
            return "SQL Injection attack";
        }
        // find the user for the userId in database
        // if found return the user:
        return "findUserName";
    }
}
```
Refactor tool in eclipse

```java
package p ooclasses;

public class LoginAction {

    public String findUser(String userId) {
        // check empty
        if (userId == null || userId.isEmpty()) {
            return "User input missing";
        }
        // check for sql injection
        String Quotes = "\"; if (userId.contains(Quotes)) {
            return "SQL Injection attack";
        }
        // find the user for the userId in database
        if (found) return "FundUserName";
    }
}
```
package poclasses;

public class LoginAction {

  public String findUser(String userId) {
    // check empty
    if (!validateUser(userId).equals("validUser")) {
      return "invalid user";
    }
    // find the user for the userId in database
    // if found return the user;
    return "foundUserName";
  }

  private String validateUser(String userId) {
    if (userId == null || userId.isEmpty()) {
      return "User input missing";
    }
    // check for sql injection
    String Quotes = "\";  
    if (userId.contains(Quotes)) {
      return "SQL Injection attack";
    }
    return "validUser";
  }

}
Introduction

• Refactoring -
  • Why refactor? When code works as it is?
• It can help programmers add functionality, fix bugs, and understand software
• Improve code metrics (Coupling, Cohesion, size etc.). contd.
Introduction

• Provide an experimental method to test the conclusions that have been published before.
• Take data from 4 different source, apply refactoring detection technique and test 9 hypotheses and draw conclusion about them.
• The 9 hypotheses are -
Introduction

9 Hypotheses

• The RENAME refactoring tool is used much more frequently by ordinary programmers than by the developers of refactoring tools.
• About 40% of refactorings performed using a tool occur in batches.
• About 90% of configuration defaults of refactoring tools remain unchanged when programmers use the tools.
• Messages written by programmers in commit logs do not reliably indicate the presence of refactoring.
• Programmers frequently floss refactor, that is, they interleave refactoring with other types of programming activity.
• About half of refactorings are not high-level, so refactoring detection tools that look exclusively for high-level refactorings will not detect them.
• Not listed all of them here.
Four DataSets

Users:

• It was originally collected in the latter half of 2005 by Murphy and colleagues.

• 41 volunteer programmers in the wild using the Eclipse Java development environment. 66 hours of development time per programmer.

• The data include information on which Eclipse commands were executed, and at what time.
Four DataSets

Everyone:
- Publicly available from the Eclipse Usage Collector.
- Automated request to send the data back to the Eclipse Foundation from everyone who consented.
- Data collected from 13k devs, during April 2008 – Jan 2009.
- Java and Non Java devs.
- Data includes eclipse command, refactoring commands used, how many time they were used etc.
Four DataSets

Toolsmiths:

- This data set is not publicly available.
- Data includes detailed refactoring histories from 4 developers who primarily maintain Eclipse’s refactoring tools.
- This dataset included all the information necessary to recreate the usage of refactoring tool.
Four DataSets

Eclipse CVS:

- Version history of the same projects, and the same developers represented in Toolsmiths.
- CVS does not maintain records showing which file revisions were committed as a single transaction.

  Find revisions committed by the same developer with the same commit message within a small time window

- Randomly sampled from about 3400 source file commits and inferred which refactorings were performed by comparing adjacent commits manually by comparing refactoring tool logs with code histories.
Findings

• **Toolsmiths and Users Differ** –
• H: We hypothesize that the refactoring behavior of the programmers who develop the Eclipse refactoring tools differs from that of the programmers who use them.
• No one else has looked at it from the point of view of refactoring tool.
• Compared the *refactoring tool usage* in the **Toolsmith** data set against the tool usage in the **User** and **Everyone** data sets.
  (Compare Experts vs User). “Refactoring tool usage” means use of each option provided by eclipse to refactor the code.
• Lets look at the table comparing the data. Specifically at Rename. Experts use other options more.
## Findings

- **Toolsmiths and Users Differ** –

<table>
<thead>
<tr>
<th>Refactoring Tool</th>
<th>Toolsmiths</th>
<th>Users</th>
<th>Everyone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Uses</td>
<td>Use %</td>
<td>Batched</td>
</tr>
<tr>
<td>Rename</td>
<td>670</td>
<td>28.7%</td>
<td>283</td>
</tr>
<tr>
<td>Extract Local Variable</td>
<td>568</td>
<td>28.4%</td>
<td>127</td>
</tr>
<tr>
<td>Inline</td>
<td>349</td>
<td>15.0%</td>
<td>132</td>
</tr>
<tr>
<td>Extract Method</td>
<td>280</td>
<td>12.0%</td>
<td>28</td>
</tr>
<tr>
<td>Move</td>
<td>147</td>
<td>6.3%</td>
<td>50</td>
</tr>
<tr>
<td>Change Method Signature</td>
<td>93</td>
<td>4.0%</td>
<td>26</td>
</tr>
<tr>
<td>Convert Local To Field</td>
<td>92</td>
<td>3.9%</td>
<td>12</td>
</tr>
<tr>
<td>Introduce Parameter</td>
<td>41</td>
<td>1.8%</td>
<td>20</td>
</tr>
<tr>
<td>Extract Constant</td>
<td>22</td>
<td>0.9%</td>
<td>6</td>
</tr>
<tr>
<td>Convert Anonymous To Nested</td>
<td>18</td>
<td>0.8%</td>
<td>0</td>
</tr>
<tr>
<td>Move Member Type to New File</td>
<td>15</td>
<td>0.6%</td>
<td>0</td>
</tr>
<tr>
<td>Pull Up</td>
<td>12</td>
<td>0.5%</td>
<td>0</td>
</tr>
<tr>
<td>Encapsulate Field</td>
<td>11</td>
<td>0.5%</td>
<td>8</td>
</tr>
<tr>
<td>Extract Interface</td>
<td>2</td>
<td>0.1%</td>
<td>0</td>
</tr>
<tr>
<td>Generalize Declared Type</td>
<td>2</td>
<td>0.1%</td>
<td>0</td>
</tr>
<tr>
<td>Push Down</td>
<td>1</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Infer Generic Type Arguments</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Use Supertype Where Possible</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Introduce Factory</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Extract Superclass</td>
<td>7</td>
<td>0.3%</td>
<td>0</td>
</tr>
<tr>
<td>Extract Class</td>
<td>1</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Introduce Parameter Object</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
</tr>
<tr>
<td>Introduce Indirection</td>
<td>0</td>
<td>0.0%</td>
<td>0</td>
</tr>
</tbody>
</table>

Total Uses: 2331, Total Use %: 100%, Batched Uses: 692, Batched Use %: 29.7%
Findings

• **Toolsmiths and Users Differ** –

• **Limitations** –

• Each data set was gathered over a different period of time, and the tools themselves may have changed between those periods.
Findings

- **Programmers Repeat Refactorings**
  - H: when programmers perform a refactoring, they typically perform several refactorings of the same kind within a short time period.
  - For example - a programmer may perform several *EXTRACT LOCAL VARIABLES* in preparation for a single *EXTRACT METHOD*.
  - On anecdotal evidence – Refactor same type of things to achieve a composite refactoring.
  - On *Toolsmith* and user data, the refactorings of the same kind that were executed within 60 seconds, considered as done in batch. 60 seconds window - just a heuristic.
  - Lets look at the graph of % of refactorings that appear as batch for *toolsmith vs user*. 
Findings

• Programmers Repeat Refactorings

Figure 1. Percentage of refactorings that appear in batches as a function of batch threshold, in seconds. 60-seconds, the batch size used in Table 1, is drawn in green.
Findings

• **Programmers Repeat Refactorings**
• Let’s go back to the table1.
• Overall, we can see that certain refactorings, such as RENAME, INTRODUCE PARAMETER, and ENCAPSULATE FIELD, are more likely to appear as part of a batch for both Toolsmiths and Users, while others, such as EXTRACT METHOD and PULL UP, are less likely to appear in a batch. In total, we see that 30% of Toolsmiths refactorings and 47% of Users refactorings appear as part of a batch.
Findings

- Programmers Repeat Refactorings
- Limitations.
- The 60 second window is just a heuristic. Some refactoring done inside that window may not be related and some which are related may fall out of that window.
Findings

- Programmers often don't Configure Refactoring Tools
- H: Programmers don't configure tools.
- Meaning of configuring tool: Say, you configure the extract method option to order the parameter by name and set the visibility of method to private.
- Reason could be – Anecdotal - programmer feel that it is easier to manually edit the refactored code later rather than provide configuration detail.
- Authors conducted a survey in Portland Java user group (8 Devs). They supplied configuration info to refactoring tool only 25% of the time.
Findings

- **Programmers often don't Configure Refactoring Tools**
- H: Programmers don't configure tools.
- On Toolsmith data set, look at how often the default value was changed. Remember *toolsmiths are experts*. They should know more about configurations.
- Lets look at the table showing that data. The overall mean change frequency is just over 10%.
Findings

• Programmers often don't Configure Refactoring Tools

<table>
<thead>
<tr>
<th>Refactoring Tool</th>
<th>Configuration Option</th>
<th>Default Value</th>
<th>Change Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extract Local Variable</td>
<td>Declare the local variable as ‘final’</td>
<td>false</td>
<td>5%</td>
</tr>
<tr>
<td>Extract Method</td>
<td>New method visibility</td>
<td>private</td>
<td>6%</td>
</tr>
<tr>
<td></td>
<td>Declare thrown runtime exceptions</td>
<td>false</td>
<td>24%</td>
</tr>
<tr>
<td></td>
<td>Generate method comment</td>
<td>false</td>
<td>9%</td>
</tr>
<tr>
<td>Rename Type</td>
<td>Update references</td>
<td>true</td>
<td>3%</td>
</tr>
<tr>
<td></td>
<td>Update similarly named variables and methods</td>
<td>false</td>
<td>24%</td>
</tr>
<tr>
<td></td>
<td>Update textual occurrences in comments and strings</td>
<td>false</td>
<td>15%</td>
</tr>
<tr>
<td></td>
<td>Update fully qualified names in non-Java text files</td>
<td>true</td>
<td>7%</td>
</tr>
<tr>
<td>Rename Method</td>
<td>Update references</td>
<td>true</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Keep original method as delegate to renamed method</td>
<td>false</td>
<td>1%</td>
</tr>
<tr>
<td>Inline Method</td>
<td>Delete method declaration</td>
<td>true</td>
<td>9%</td>
</tr>
</tbody>
</table>

Table 2. Refactoring tool configuration in Eclipse from Toolsmiths.
Findings

• **Programmers often don't Configure Refactoring Tools**
• Limitations:
  • 1. Only toolsmith dataset was looked at
  • 2. Only *limited configurations* could be looked at (and that too only for most popular refactoring types) .
Findings

• **Commit Messages don't predict Refactoring**
  
  Few previous research hypothesized that messages attached to commit history can be used to predict if the commit had refactored code in it.

• H: This assumption is false.

• Reason could be – Devs may do refactoring subconsciously and may not enter comments because they were implementing a feature not explicitly refactoring.
Findings

• **Commit Messages don't predict Refactoring**

• Previous research Retzinger reported that you can look for certain keywords like 'move', 'rename' and exclude things like 'need refactoring' to look for commits that contain refactored code. They reported that they were able to predict this to the accuracy of 95% in two projects in a 100 randomly chosen commits.

• **Replicated** Ratzinger's experiment with eclipse cvs data.
Findings

• **Commit Messages don't predict Refactoring.**

• Because the projects itself were related to refactoring tool, there were some comments attached to these commits which had words like 'implemented X to refactor Y'. Such comments were manually removed from the commits.

• The selected commits were grouped into labeled and not labeled.

• 20 commits were then randomly pulled out of these two 'classified' commits. To review if and how many refactorings these commits contained.
Findings

- Commit Messages don't predict Refactoring.

<table>
<thead>
<tr>
<th>Change</th>
<th>Labeled</th>
<th>Unlabeled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure Whitespace</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>No Refactoring</td>
<td>8</td>
<td>11</td>
</tr>
<tr>
<td>Some Refactoring</td>
<td>5 (1,4,11,15,17)</td>
<td>6 (2,9,11,23,30,37)</td>
</tr>
<tr>
<td>Pure Refactoring</td>
<td>6 (1,2,3,3,5)</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>20 (65)</td>
<td>20 (112)</td>
</tr>
</tbody>
</table>

Table 3. Refactoring between commits in Eclipse CVS. Plain numbers count commits in the given category; tuples contain the number of refactorings in each commit.
Findings

• **Commit Messages don't predict Refactoring.**
• The labelled group should have captured more refactoring than the not labeled group.
• Commits labeled with a refactoring keyword contained far fewer refactorings (63, or 36% of the total) than those not so labeled (112, or 64%)
• Commit message does not provide a complete picture of refactoring activity
Findings

• **Commit Messages don't predict Refactoring.**

• **Limitations:**
  
  1. Could not mirror the experiment exactly. Conversation with Ratzinger revealed that included' and excluded keywords were project specific.

  2. **Difference in project type** could have played a part

  3. Small sample size - only 40 commits
Findings

• **Floss Refactoring is Common**-
  Floss refactoring - Refactor as you code along. Keep the code clean.
  Root Canal refactoring - Refactor just to refactor. Exclusive Refactoring.

• So far most of the case studies and literature has been on root canal. But if it is not frequently done in practice then focus of research should change?
Findings

- **Floss Refactoring is Common**-
- Definition of flossing, root canal coincide with table 3 classifications - some factoring, pure refactoring.
- We see that only 3% commits reflect root canal and 30% indicate floss.
- Within **user dataset** - they observed session (time b/w two consecutive commits by a single programmer) data for refactoring tool usage. If for a session only refactoring tool was used and no manual edits were done then that is root canal otherwise considered floss. Which is a shaky definition but only 0.4% were found to be root canal under such definition.
Findings

- **Floss Refactoring is Common**-
- Limitations –
- The definition of session is shaky.
- Only the refactoring done by tool was considered. Manual refactoring was not considered, which may have changed number for floss vs root canal.
- The limitation from ‘commit messages don’t capture refactoring’ apply here as well because we used the same data.
Findings

- Many Refactorings are Medium and Low-level
  - Low level refactorings - are those that make changes to only blocks of code.
  - Ex – Rename
  - Medium level - are those that change the signatures of classes, methods, and fields and also significantly change blocks of code.
  - Ex – Extract Method
  - High level - change the signatures of classes, methods, and field
  - Ex- Move class.
Findings

• Many Refactorings are Medium and Low-level
  • Past research has often drawn conclusion based on high level refactoring.
  • Any research that relies on automatic refactoring detection tool which can only
detect high level refactoring may be missing the low and medium level refactoring. If
these are also frequent enough then the results from these studies may not be
applicable in all cases.
  • Used eclipse cvs data and toolsmith data to classify the refactoring commits into
these levels and as seen in the table, the low and medium level make up about 40% of
these changes. Table up next.
Findings

• Many Refactorings are Medium and Low-level

<table>
<thead>
<tr>
<th></th>
<th>Eclipse CVS</th>
<th>Toolsmiths</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low</td>
<td>18%</td>
<td>33%</td>
</tr>
<tr>
<td>Medium</td>
<td>22%</td>
<td>27%</td>
</tr>
<tr>
<td>High</td>
<td>60%</td>
<td>40%</td>
</tr>
</tbody>
</table>

Table 4. Refactoring level percentages in the Eclipse CVS and the Toolsmiths data.
Findings

• **Refactorings are frequent** -
• Even past research suggest that practice of refactoring is frequent (based on open source project).
• Limitation in the past research was that - the low level refactoring was not considered. **Imprecision** in distinguishing **semantic changes** from **signature changes**.
Findings

• Refactorings are frequent –
  • Refactoring should be habitual and not done all at the release time.
  • In toolsmith dataset – on an average 30 refactoring per week during year 2006 and 46 per week during year 2007.
  • In user data set – Refactoring should be widespread across session and not done by just few select developers.
  • 41% of session contained refactoring. 71% of code that was changed was changed during these sessions when refactoring occurred.
  • The analysis of user dataset suggest that whenever a programmer had to change a large piece of code, refactoring is a common way to prepare for those changes.
Findings

• Refactorings are frequent –
• Limitations –
• All the inspected refactoring were the **ones achieved by a tool.** If **manual refactoring** was considered, the number will only **go higher up.**
Findings

• **Refactoring tools are underused** –
  
  • H: Programmers often do not use refactoring tools.

  • **Ideally**, a developer would like to use the refactoring tool because the refactoring done with the tool is less prone to errors and faster than manual. **But**

  • In a survey of 116 agile enthusiasts, it was reported that developers used tools to refactor code only **68% of time**.

  • In another survey of 16 students **only 2 reported** having used refactoring tool.

  • This could be the result of **poor user interface** that the refactoring tool presents.
Findings

• **Refactoring tools are underused –**
• How do you validate the hypothesis?
• Idea - Idea is to look for refactoring that were found manually in eclipse cvs data set and co-relate them with the tool usage in toolsmith dataset. Remember that this cvs data is of the same project in toolsmith dataset.
• How ? Look for tool usage data in toolsmith dataset between the commits that were found to have refactoring.
• If related tool data usage is not found then the refactoring was done manually.
Findings

• Refactoring tools are underused –
• 89% of 145 refactoring could not be related to tool usage. toolsmith like to perform refactoring manually.
Findings

- Refactoring tools are underused –
- Limitations –
  1. Some tool usage data may be missing. (As shown in the table in gray rows)
  2. Refactoring that occurred earlier may not be checked in until days later. This would result in not being able to relate the tool usage with commit. (limited the effect of this drawback by looking back days after the commit to try and relate the refactoring with the tool usage)
Findings

• **Different Refactorings are Performed with and without Tools -**
  
  • Some refactorings are more prone to be done manually.
  
  • Discrepancy between how programmers want to refactor and how they actually do it. Developer wants to refactor extract method but the most frequent one is rename method. may be because its UI is much simpler to use.

  • In the next image you will see that Extract method is done 9 times with manually refactoring but just once with a tool. However Rename is almost all the time used with tool. Some are exclusively done manually (because there exist no support in refactoring tool)
Findings
Findings – All 9 Hypotheses

• The RENAME refactoring tool is used much more frequently by ordinary programmers than by the developers of refactoring tools.
• About 40% of refactorings performed using a tool occur in batches.
• About 90% of configuration defaults of refactoring tools remain unchanged when programmers use the tools.
• Messages written by programmers in commit logs do not reliably indicate the presence of refactoring.
• Programmers frequently floss refactor, that is, they interleave refactoring with other types of programming activity.
• About half of refactorings are not high-level, so refactoring detection tools that look exclusively for high-level refactorings will not detect them.
• refactorings are performed frequently performed
• almost 90% of refactorings are manually, without the help of tools
• the kind of refactoring performed with tools differs from the kind performed manually
Discussion

- **Tool-Usage Behavior** – most of the tools are unused. Experts are slightly better at using them. Improve the under used tool or provide better documentation.

- **Min Config** – Create tools that require minimum configurations.

- **Detecting Refactoring** –
  1. Commit logs - not reliable.
  2. Don’t forget to account for low and medium level refactoring.
  3. Make more project specific detection technique (example – removing the words like ‘created refactoring class’ if you are studying the toolsmtih like project)
Discussion

- Refactoring practice –
  1. Refactoring is frequent. It's just that tools are not being used to do it. Improve the tools and documentation for tools.
  2. Provide support for floss refactoring – Minimum work switch from refactoring to go back to working on functionality.
  3. Developers refactor in batches, provide support for it. Example – Detect code duplication in class which can be replaced by same extracted methods.
  4. Toolsmiths need to come up with better interfaces and improve the flow of tools. Example – remind developer to extract local variable before extract method is done.
Limitations

• Other than what has been discussed in each section
  1. Only eclipse and Java based refactoring is looked at
  2. User and programmers may not represent programmers in general.
Conclusion

1. Good news Refactoring is frequent. Bad news – very few people use tools
2. Toolsmiths need to provide support for things like – Batch refactoring etc.
3. Researchers need to alter their thinking about certain assumption that have been proved wrong in this paper.
Discussion points

• As admitted in the paper – They looked only at eclipse and Java. Would these results hold up for Dynamically typed languages? refactoring code for them should be lot different.

• Dataset ‘everyone’ is heavily unused, which is largest of them all.

• Why are these results important? For example Why do we care about that programs repeat refactoring - one use could be to improve tool support and suggestion stuff.. or may be machine learn about refactoring.

But if it is about improving tool, the tool itself as the paper reports is rather unused because of bad user interface. A study on user interface would prove more helpful toward that goal?

• Do you find any results that are counter intuitive? Or surprising? What is the significance of this paper?
Discussion points

• Why did they choose expert's project. It just introduced unwanted level of complexity.
• In general, Why is it important to understand the difference between how exert use the tool vs how non-experts use it?
• Manually detected the refactorings