How We Refactor, and How We Know it

Emerson Murphy-Hill, Chris Parnin, Andrew P. Black

ICSE 2009

Introduction

- Refactoring is the process of changing the structure of a program without changing the way that it behaves.
- To help put refactoring research on a sound scientific basis, we draw conclusions using four data sets spanning:
  - More than 13 000 developers
  - 240 000 tool-assisted refactorings
  - 2500 developer hours
  - 3400 version control commits.
- Using these data, we cast doubt on several previously stated assumptions about how programmers refactor, while validating others

The Data that We Analyzed

- The work described in this paper is based on four sets of data
- The first set we will call **Users**
  - It was originally collected in the latter half of 2005 by Murphy and colleagues who used the Mylyn Monitor tool to capture and analyze fine-grained usage data from 41 volunteer programmers in the wild using Eclipse
  - These data capture an average of 66 hours of development time per programmer
  - About 95 percent of the programmers wrote in Java.
  - The data include information on which Eclipse commands were executed, and at what time

The Data that We Analyzed

- The second set of data we will call **Everyone**
  - It is publicly available from the Eclipse Usage Collector
  - It includes data from every user of the Eclipse Ganymede release who consented to an automated request to send the data back to the Eclipse Foundation.
  - These data aggregate activity from over 13 000 Java developers between April 2008 and January 2009, but also include non-Java developers.
  - The data count how many programmers have used each Eclipse command, including refactoring commands, and how many times each command was executed.
- We know of no other research that has used these data for characterizing programmer behavior

The Data that We Analyzed

- The third set of data we will call **Toolsmiths**
  - It includes refactoring histories from 4 developers who primarily maintain Eclipse's refactoring tools.
  - These data include detailed histories of which refactorings were executed, when they were performed, and with what configuration parameters
- The fourth set of data we will call **Eclipse CVS**
  - It is the version history of the Eclipse and JUnit extracted from CVS repositories.

Findings on Refactoring Behavior

Toolsmiths and Users Differ

Refactoring Tool	Toolsmiths				Users				Everyone	
	Uses	Use %	Batched	Batched %	Uses	Use %	Batched	Batched %	Uses	Use %
Rename	670	28.7%	283	42.2%	1862	61.5%	1009	54.2%	179871	74.8%
Extract Local Variable	568	24.4%	127	22.4%	322	10.6%	106	32.9%	13523	5.6%
Inline	349	15.0%	132	37.8%	137	4.5%	52	38.0%	4102	1.7%
Extract Method	280	12.0%	28	10.0%	259	8.6%	57	22.0%	10581	4.4%
Move	147	6.3%	50	34.0%	171	5.6%	98	57.3%	13208	5.5%
Change Method Signature	93	4.0%	26	28.0%	55	1.8%	20	36.4%	4764	2.0%
Convert Local To Field	92	3.9%	12	13.0%	27	0.9%	10	37.0%	1603	0.7%
Introduce Parameter	41	1.8%	20	48.8%	16	0.5%	11	68.8%	416	0.2%
Extract Constant	22	0.9%	6	27.3%	81	2.7%	48	59.3%	3363	1.4%
Convert Anonymous To Nested	18	0.8%	0	0.0%	19	0.6%	7	36.8%	269	0.1%
Move Member Type to New File	15	0.6%	0	0.0%	12	0.4%	5	41.7%	838	0.3%
Pull Up	12	0.5%	0	0.0%	36	1.2%	4	11.1%	1134	0.5%
Encapsulate Field	11	0.5%	8	72.7%	4	0.1%	2	50.0%	1739	0.7%
Extract Interface	2	0.1%	0	0.0%	15	0.5%	0	0.0%	1612	0.7%
Generalize Declared Type	2	0.1%	0	0.0%	4	0.1%	2	50.0%	173	0.1%
Push Down	1	0.0%	0	0.0%	1	0.0%	0	0.0%	279	0.1%
Infer Generic Type Arguments	0	0.0%	0	-	3	0.1%	0	0.0%	703	0.3%
Use Supertype Where Possible	0	0.0%	0	-	2	0.1%	0	0.0%	143	0.1%
Introduce Factory	0	0.0%	0	-	1	0.0%	0	0.0%	121	0.1%
Extract Superclass	7	0.3%	0	0.0%	*	-	*	*	558	0.2%
Extract Class	1	0.0%	0	0.0%	*	-	*	*	983	0.4%
Introduce Parameter Object	0	0.0%	0	-	*	-	*	*	208	0.1%
Introduce Indirection	0	0.0%	0	-	*	-	*	*	145	0.1%
Total	2331	100%	692	29.7%	3027	100%	1431	47.3%	240336	100%

Table 1. Refactoring tool usage in Eclipse. Some tool logging began in the middle of the *Toolsmiths* data collection (shown in light grey) and after the *Users* data collection (denoted with a *).

Programmers Repeat Refactorings

- We hypothesize that when programmers perform a refactoring, they typically perform several refactorings of the same kind within a short time period.
  - We used the Toolsmiths and the Users data to measure the temporal proximity of refactorings to one another.
  - We say that refactorings of the same kind that execute within 60 seconds of each another form a batch.
  - In Table 1, each “Batched” column indicates the number of refactorings that appeared as part of a batch
  - In total, we see that 30% of Toolsmiths refactorings and 47% of Users refactorings appear as part of a batch.

Programmers don’t Configure Refactoring Tools

- Refactoring tools are typically of two kinds:
  - They either force the programmer to provide configuration information, such as whether a newly created method should be public or private
  - Or they quickly perform a refactoring without allowing any configuration
- “Change Frequency” refers to how often a user used a configuration option other than the default.
- The data suggest that refactoring tools are configured very little: the overall mean change frequency for these options is just under 10%.

Programmers don’t Configure Refactoring Tools

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

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

Commit Messages don’t predict Refactoring

- Several researchers have used messages attached to commits

into a version control as indicators of refactoring activity.

- Murphy-Hill and Parnin compared 40 commits over the span of about 6 hours.

Change	Labeled	Unlabeled
Pure Whitespace	1	3
No Refactoring	8	11
Some Refactoring	5 (1,4,11,15,17)	6 (2,9,11,23,30,37)
Pure Refactoring	6 (1,1,2,3,3,5)	0
Total	20(63)	20(112)

Floss Refactoring is Common

- In previous work, we distinguished two tactics that programmers use when refactoring: floss refactoring and root-canal refactoring.
- During floss refactoring, the programmer uses refactoring as a means to reach a specific end, such as adding a feature or fixing a bug.
- Thus, during floss refactoring the programmer intersperses refactoring with other kinds of program changes to keep code healthy.
- Root-canal refactoring, in contrast, is used for correcting deteriorated code and involves a protracted process consisting of exclusive refactoring.

Floss Refactoring is Common

- For convenience, we let a programming session be the period of time between consecutive commits to CVS by a single programmer.
- In a particular session, if a programmer both refactors and makes a semantic change, then we say that that the programmer is floss refactoring.
- If a programmer refactors during a session but does not change the semantics of the program, then we say that the programmer is root-canal refactoring
- Commits indicating floss refactoring would account for 30% of commits while commits indicating root-canal would account for only 3% of commits

Floss Refactoring is Common

- In no more than 10 out of 2671 commits did programmers use a refactoring tool without also manually editing their program.
- In other words, in less that 0.4% of commits did we observe the possibility of root-canal refactoring using only refactoring tools.

Many Refactorings are Medium and Low-level

- We divided the refactorings into three levels — High, Medium and Low.
- High level refactorings are those that change the signatures of classes, methods, and fields;
  - Examples: RENAME CLASS, MOVE STATIC FIELD, and ADD PARAMETER.
- Medium level refactorings are those that change the signatures of classes, methods, and fields and also significantly change blocks of code
  - Example: EXTRACT METHOD, INLINE CONSTANT, and CONVERT ANONYMOUS TYPE TO NESTED TYPE.

Many Refactorings are Medium and Low-level

- Low level refactorings are those that make changes to only blocks of code
  - Example: EXTRACT LOCAL VARIABLE, RENAME LOCAL VARIABLE, and ADD ASSERTION

Many Refactorings are Medium and Low-level

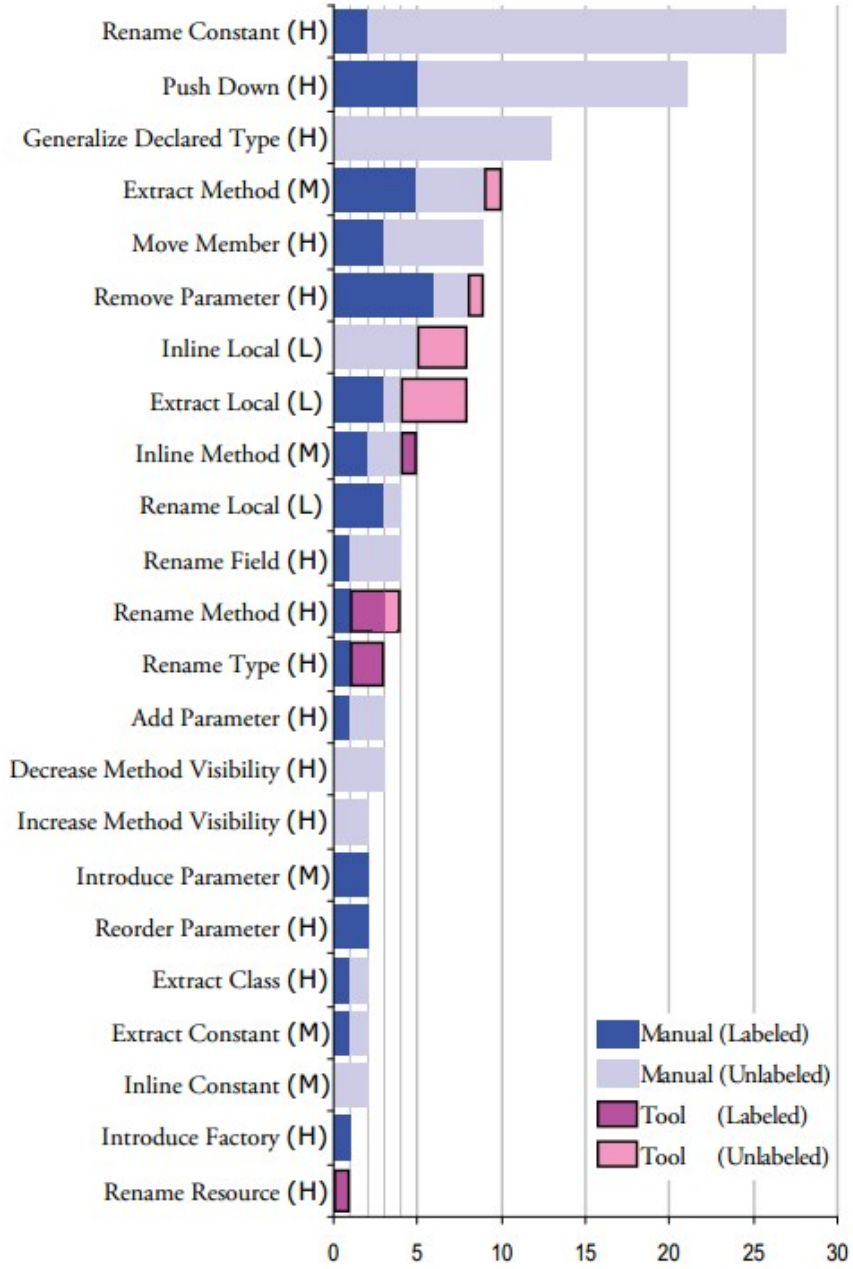

Figure 2. Refactorings over 40 sessions.

Refactorings are Frequent

- In the **Toolsmiths** data, we found that refactoring activity occurred throughout the Eclipse development cycle.
  - In 2006, an average of 30 refactorings took place each week;
  - In 2007, there were 46 refactorings per week.
  - Only two weeks in 2006 did not have any refactoring activity, and one of these was a winter holiday week.
  - In 2007, refactoring occurred every week.

Refactorings are Frequent

- In the **Users** data set, we found refactoring activity distributed throughout the programming sessions.
  - Overall, 41% of programming sessions contained refactoring activity.
  - More interestingly, sessions that did not have refactoring activity contained an order of magnitude fewer edits than sessions with refactoring, on average.
  - The sessions that contained refactoring also contained, on average, 71% of the total edits made by the programmer

Refactoring Tools are Underused

- Several of our findings have reflected on the behavior of programmers using refactoring tools.
- For example, our finding about how **Toolsmiths** differ from regular programmers in terms of refactoring tool usage (Section 3.1) suggests that most kinds of refactorings will not be used as frequently as the **Toolsmiths** hoped, when compared to the frequently used RENAME refactoring.
- For the **Toolsmith**, this means that improving the underused tools or their documentation (especially the tool for EXTRACT LOCAL VARIABLE) may increase tool use.

Refactorings With and Without Tools

- Comparing these results, we inferred that the EXTRACT METHOD tool is underused: the refactoring is instead being performed manually.
- However, it is unclear what other refactoring tools are underused. Moreover, there may be some refactorings that must be performed manually because no tool yet exists.
- We suspect that the reason that some kinds of refactoring — especially RENAME— are more often performed with tools is because these tools have simpler user interfaces.

Conclusions and Contributions

- 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

Conclusions and Contributions

- 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
- Almost 90% of refactorings are performed manually, without the help of tools
- The kind of refactoring performed with tools differs from the kind performed manually

Questions ?

Renato Diniz