How We Refactor, and How We Know It

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

Proceedings of the 31st International Conference on Software Engineering (ICSE '09)

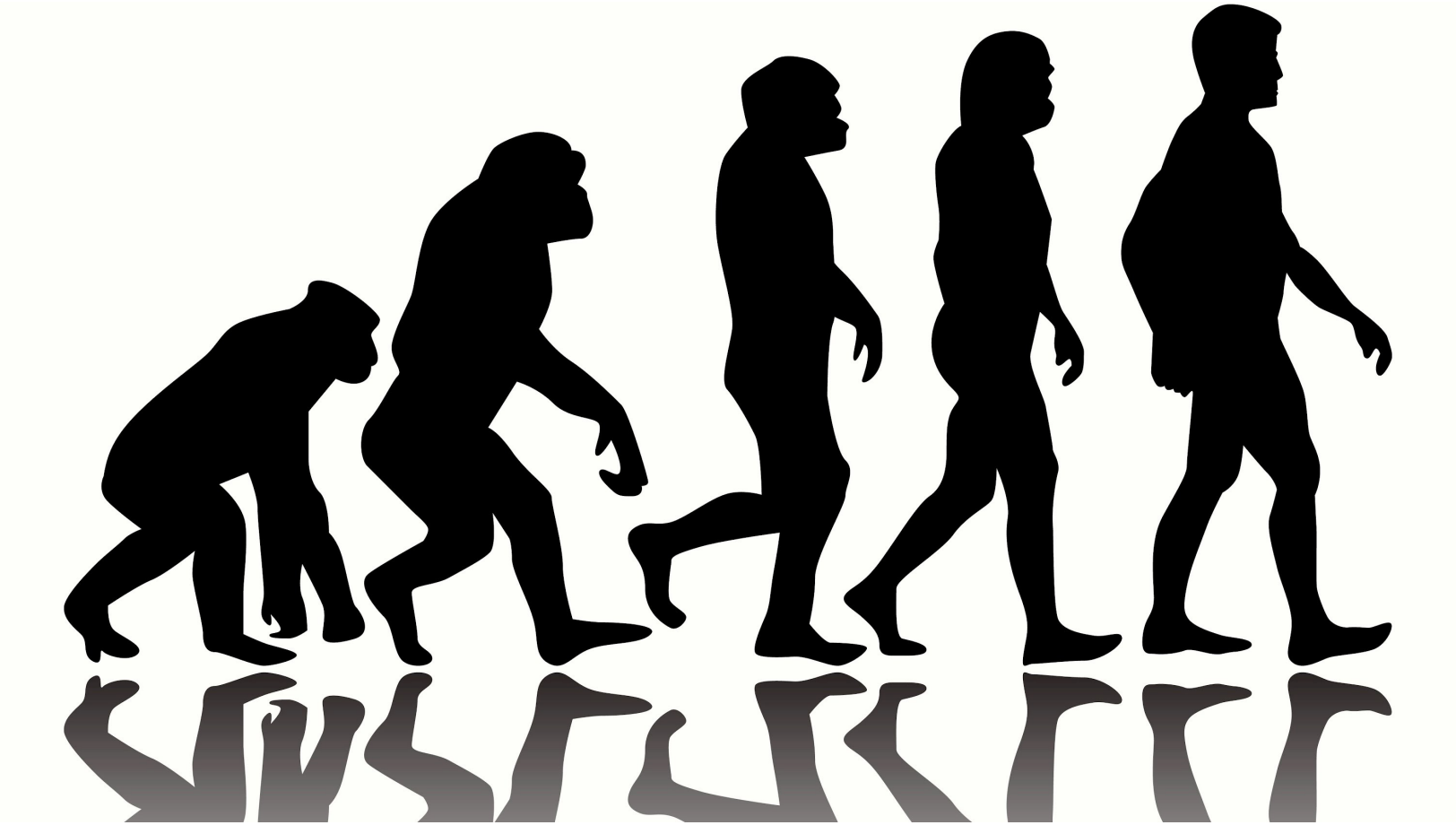
Presented by

Pablo Navarro

What is refactoring ?

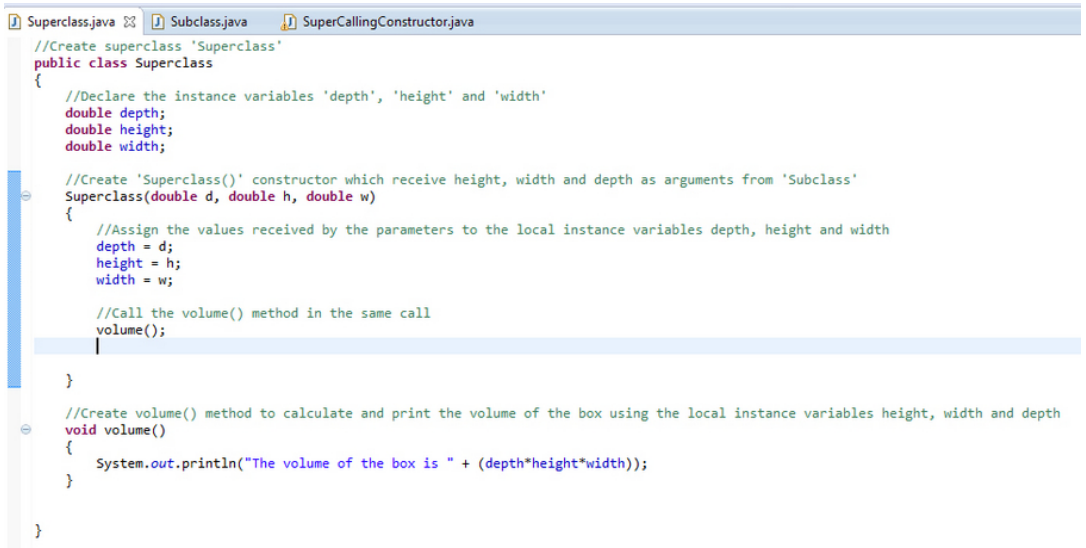
- Refactoring is the process of changing the structure of a program without changing the way that it behaves.
- There are 72 different types of refactoring.
- Refactoring produces significant benefits:
  - Help programmers add functionality.
  - Fix bugs
  - Understand software

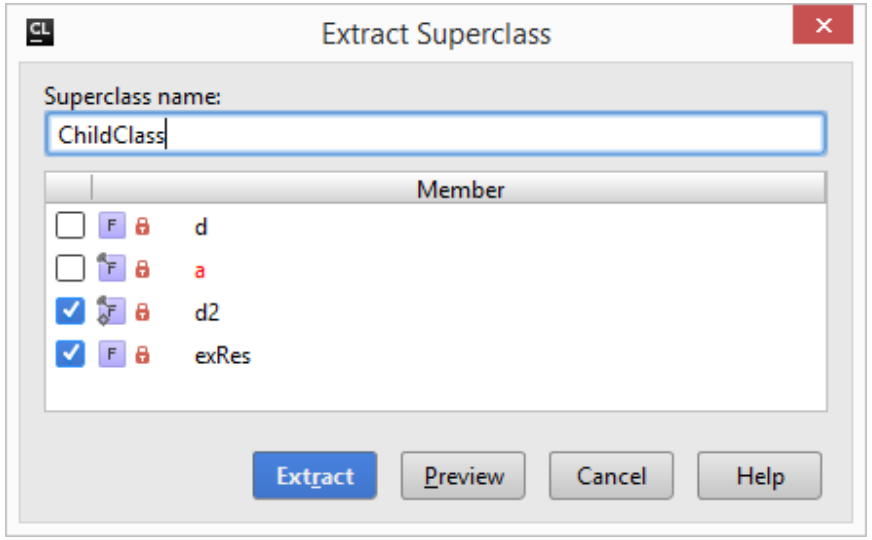
How refactoring looks



How programmers perform a refactoring task?

- Manual refactoring.
  - Regular coding.
  - Slow.
  - Error prone.
  - Higher fine detail control.
- Automated tools for refactoring.
  - GUI based.
  - Fast.
  - Less errors.
  - Less fine detail control.





Research performed

- Some data sources were analyzed to look for answers to some development questions.

The data sources

- ***Users*** : originally collected in the latter half of 2005 using the Mylyn Monitor tool to capture and analyze fine-grained usage data from 41 volunteer programmers in the wild using the Eclipse.
- ***Everyone***: publicly available from the Eclipse Usage Collector 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.
- ***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.
- ***Eclipse CVS***: the version history of the Eclipse and Junit code bases as extracted from their Concurrent Versioning System (CVS) repositories. Required a lot of preprocessing because it was very unstructured.

Toolsmiths and Users Differ

- The toolsmiths use a broader array of refactoring types compared to the average users.
  - Most average users just use two types of automatic refactoring.
- It’s hard to claim this is universally true because the datasets compared were not obtained following the same criteria.
- This claim needs to have additional validation from the research community to have definitive conclusions.

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%

Programmers Repeat Refactorings

- Programmers tend to make batches of many refactorings of the same type together.
  - Almost 50% of the refactoring were performed as part of batches.

- The way these batches were counted could be improved.
- The way the refactorings were counted could also de improved.

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%

Programmers often 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

- The authors tried to infer refactoring commits from the commit messages with a 50% success rate.
- Only commits that didn't change the function and were pure refactoring showed good results.

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)

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.

Floss Refactoring is Common

- Floss refactoring vs Root canal refactoring.
  - Floss refactoring :
    - Small.
    - Frequent.
    - Mixed with some other tasks (it's not an exclusive task to refactor).
    - Keeps code healthy.
    - Perceived as the best practice.
  - Root canal refactoring :
    - Big.
    - Not frequent.
    - It is performed as just refactoring.
    - Corrective process.
    - Perceived as an emergency procedure.

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)

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.

Many Refactorings are Medium and Low-level

- High level refactorings are those that change the signatures of classes, methods, and fields.
- Medium level refactorings are those that change the signatures of classes, methods, and fields and also significantly change blocks of code.
- Low level refactorings are those that make changes to only blocks of code.

Many Refactorings are Medium and Low-level

- Counts for refactors should take into account High level refactors.

	Eclipse CVS	*Toolsmiths*
Low	18%	33%
Medium	22%	27%
High	60%	40%

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

Refactorings are Frequent.

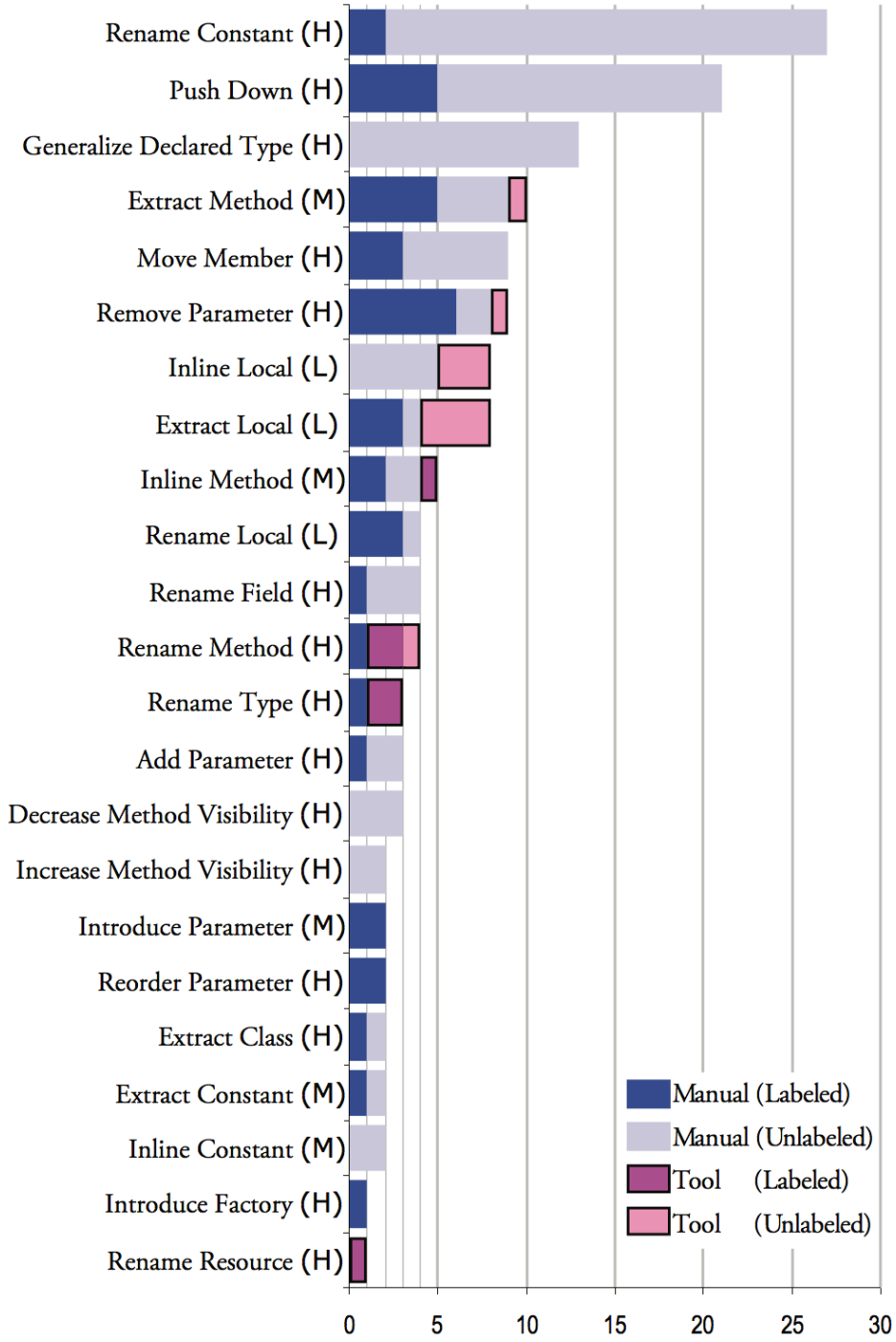
- ***Toolsmiths*** data: it was found that refactoring activity occurred throughout the Eclipse development cycle. In 2006, an average of 30 refactorings took place each week, in 2007, there were 46 refactorings per week.
- ***Users*** : the refactoring activity distributed throughout the programming 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. This analysis of the Users data suggests that **when programmers must make large changes to a code base, refactoring is a common way to prepare for those changes.**

Refactoring Tools are Underused

- ***Toolsmiths***: 89% of 145 observed refactorings could not be linked with any use of an automatic refactoring tool (also 89% when normalized)

Different Refactorings are Performed with and without Tools

- Eclipse CVS: Some refactoring types are more likely to be performed manually or some other are more likely to be performed using tools.


Rename Constant (H)
Push Down (H)
Generalize Declared Type (H)
Extract Method (M)
Move Member (H)
Remove Parameter (H)
Inline Local (L)
Extract Local (L)
Inline Method (M)
Rename Local (L)
Rename Field (H)
Rename Method (H)
Rename Type (H)
Add Parameter (H)
Decrease Method Visibility (H)
Increase Method Visibility (H)
Introduce Parameter (M)
Reorder Parameter (H)
Extract Class (H)
Extract Constant (M)
Inline Constant (M)
Introduce Factory (H)
Rename Resource (H)
0
5
10
15
20
25
30
Manual (Labeled)
Manual (Unlabeled)
Tool (Labeled)
Tool (Unlabeled)

Findings

Tool-Usage Behavior

- Improvements are necessary in the automatic refactoring tools.
- Questions still remain for researchers to answer.
  - Why is the RENAME refactoring tool so much more popular than other refactoring tools?
  - Why do some refactorings tend to be batched while others do not?

Detecting Refactoring

- Future research can complement existing refactoring detection tools with refactoring logs from tools to increase recall of low-level refactorings.

Refactoring Practice

- Floss refactoring is most frequent than Root canal refactoring.
- Refactoring tools should support flossing by allowing the programmer to switch quickly between refactoring and other development activities, which is not always possible with existing refactoring tools.

Limitations of this Study

- The only programming language used for this study is Java.
  - Different languages can yield different results.
- ***Users*** and ***Toolsmiths*** may not be representative of the average user.
- ***Users*** and ***Everyone*** might be overlapping with the ***Toolsmiths*** since they were voluntary based.
  - Some of those voluntaries could also be a Toolsmith.

Conclusions

- Refactoring has been embraced by a large community of users, many of whom include refactoring as a constant companion to the development process.
- The authors have found evidence that suggests that researchers might have to reexamine certain assumptions about refactorings. Low and medium level refactorings are much more abundant, and commit messages less reliable, than previously supposed.
- Future research should investigate why certain refactoring tools are underused and consider how this knowledge can be used to rethink these tools.

Comments

- This is a very hard to topic to research.
- The authors used heterogeneous data sources.
  - This can be confusing.
- It is very hard to get clean data regarding refactoring.
- Refactoring can mean different things for different people.
- Refactoring is hard to isolate.
  - I think that changing or adding code to a program will require some kind refactoring sooner or later.
- There was not a central topic or an overarching element.
  - I think the authors took a dive into the data with a curious mindset and found the answers before the questions.
- And finally ….


LIMITATIONS
EVERYWHERE
imgflip.com

References

- Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. 2009. How we refactor, and how we know it. In Proceedings of the 31st International Conference on Software Engineering (ICSE '09). IEEE Computer Society, Washington, DC, USA, 287-297. DOI=10.1109/ICSE.2009.5070529 http://dx.doi.org/10.1109/ICSE.2009.5070529

Questions

- Did you learn to refactor before knowing what refactoring was ?
- Do you use refactoring tools for your code? Why or why not?
- Do you have some ideas to find information about refactoring ?
- What do you think could be improved in future research for this paper ?
- Comments in general.