

Why Don't People Use Refactoring Tools?

Emerson Murphy-Hill and Andrew P. Black
Portland State University
{emerson,black}@cs.pdx.edu

Abstract

Tools that perform refactoring are currently under-utilized by programmers. As more advanced refactoring tools are designed, a great chasm widens between how the tools must be used and how programmers want to use them. In this position paper, we characterize the dominant process of refactoring, demonstrate that many research tools do not support this process, and initiate a call to action for designers of future refactoring tools.

1. Refactoring Tools are Underutilized

Since the original Refactoring Browser [11], programming environments have seen a remarkable integration of tools that perform semi-automatic refactoring. Programmers have their choice of refactoring tools in most mainstream languages such as Java and C#.

However, we believe that people just aren't using refactoring tools as much as they could. During a controlled experiment, we asked 16 object-oriented programming students whether they had used refactoring tools – only two said they had, reporting using them only 20% and 60% of the time [7]. Of the 31 users of Eclipse 3.2 on Portland State University computers in the last 9 months, only 2 users logged any refactoring activity. In a survey we conducted at the Agile Open Northwest 2007 conference, 112 people self-reported on their use of refactoring tools. When available, they chose to use refactoring tools 68% of the time when tools were available, an estimate which is likely optimistically high. Murphy and colleagues' data on Eclipse usage characterizes 41 programmers who were early tool adopters, and who used Eclipse for a significant amount of Java programming [6]. According to this data, over a mean period of about 66 hours per programmer, the median number of different refactoring tools used was just 4, with Rename and Move as the only refactorings practiced by the majority of subjects.

While it is difficult to tell when people are using refactoring tools and when they *could* be using refactoring tools, this second hand evidence leads us

to believe that refactoring tools are currently not used as much as they could be.

2. When do Programmers Refactor?

We believe that explaining when programmers refactor also explains why programmers don't use refactoring tools, especially tools produced by researchers.

There are two different occasions when programmers refactor. The first kind occurs interweaved with normal program development, arising whenever and wherever design problems arise. For example, if a programmer introduces (or is about to introduce) duplication when adding a feature, then the programmer removes that duplication. Fowler originally argued strongly for this kind of refactoring [1], and more recently, Hayashi and colleagues [3] and Parnin and Görg [8] stated they believed this was a common refactoring process. This kind of refactoring, done frequently to maintain healthy software, we shall call **floss refactoring**.

The other kind of refactoring occurs when time is set aside. For example, a programmer may want to remove as much duplication as possible from an existing program. This sort of refactoring has been described by Kataoka and colleagues [4], Pizka [9], and Borquin and Keller [1]. This kind of refactoring, done after software has become unhealthy, we shall call **root canal refactoring**.

Floss refactoring appears to be more effective, currently more widely used, and likely to be more widely used in the future. Both Pizka [9] and Borquin and Keller [1] note distinct negative consequences when performing root canal refactoring. Over the history of 3 large open-source projects, Weißgerber and Diehl were surprised to find that development contained no days of only refactorings [13]; if a day contained only refactorings, it would have indicated root canal refactoring was taking place. Likewise, Eclipse usage data from Murphy and colleagues [6] show that on only one occasion out of thousands did a programmer perform only refactoring iterations between version control commits. Furthermore, because floss refactoring is a central part of Agile

methodologies, as more programmers become Agile, we expect more programmers to adopt floss refactoring.

3. Tool Support for Floss Refactoring

Even though floss refactoring appears to be a more popular strategy than root canal refactoring, many (if not most) tools for refactoring described in the literature are built for root canal usage.

Smell detectors, fully automated refactoring tools, and refactoring scripts are examples of refactoring tools are typically built for root canal refactoring. For instance, jCosmo takes a significant amount of time and reports system-wide smells [12], making it inappropriate for floss refactoring. Guru restructures an entire inheritance hierarchy without regard to what a programmer is having trouble modifying or understanding [5], making this tool unsuitable to floss refactoring as well. Refactoring Browser scripts [10] may be too viscous for a programmer to use to perform an impromptu restructuring during floss refactoring.

While we are only able to point out a few examples due to space constraints, we believe that the majority of tools described in the literature are designed for root canal refactoring. Some exceptions do exist, such as Hayashi and colleagues' tool, which suggests refactoring candidates based on programmers' copy and paste behavior [3].

4. Future Work

We suggest that future work on refactoring tools should pay more attention to floss refactoring. Many refactoring tools can be built in a way that supports either floss or root-canal refactoring; we suggest tool builders be cognizant of which one their tool supports.

A good way to determine what kind of refactoring your tool supports is to conduct user studies. These studies can be as simple as having a few undergraduates try to refactor some open-source code. In our research, we have found that such studies are invaluable in determining the preferred usage and the limitations of our tools.

5. Acknowledgements

This research supported by the National Science Foundation under grant number CCF-0520346.

6. References

- [1] F. Bourquin and R. Keller, "High-Impact refactoring based on architecture violations," Proceedings of CSMR 2007.
- [2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code: Addison-Wesley Professional, 1999.
- [3] S. Hayashi, M. Saeki, and M. Kurihara, "Supporting Refactoring Activities Using Histories of Program Modification," IEICE Transactions on Information and Systems, 2006.
- [4] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," presented at International Conference on Software Maintenance, 2002.
- [5] I. Moore. "Automatic inheritance hierarchy restructuring and method refactoring," In Proceedings of Object-Oriented Programming, Systems, Languages, and Applications. ACM Press, New York, NY, 235-250, 1996.
- [6] G. Murphy, M. Kersten, and L. Findlater, "How Are Java Software Developers Using the Eclipse IDE?," IEEE Software, 2006.
- [7] E. Murphy-Hill. Improving Refactoring with Alternate Program Views. Technical Report TR-06-05, Portland State University, Portland, OR, 2006.
- [8] C. Parnin, G. and C. Görg, "Lightweight visualizations for inspecting code smells," Proceedings of the 2006 ACM Symposium on Software Visualization, 2006.
- [9] M. Pizka. "Straightening Spaghetti Code with Refactoring." In *Proc. of the Int. Conf. on Software Engineering Research and Practice - SERP*, pages 846-852, Las Vegas, NV, June 2004.
- [10] D. Roberts and J. Brant, "Tools for making impossible changes - experiences with a tool for transforming large Smalltalk programs," IEE Proceedings - Software, vol. 151, pp. 49-56, 2004.
- [11] D. Roberts, J. Brant, and R. Johnson, "A refactoring tool for Smalltalk," Theor. Pract. Object Syst., vol. 3, pp. 253-263, 1997.
- [12] E. Van Emden and L. Moonen, "Java Quality Assurance by Detecting Code Smells," Proceedings of the Ninth Working Conference on Reverse Engineering, 2002.
- [13] P. Weißgerber, and S. Diehl, "Are refactorings less error-prone than other changes?," presented at MSR '06: Proceedings of the 2006 international workshop on Mining software repositories, 2006.