

# Flossing Regularly – The Key to Healthy Software



Emerson Murphy-Hill & Andrew P. Black, {emerson,black}@cs.pdx.edu  
Maseeh College of Engineering & Computer Science



<b>Floss Refactoring</b> Programmers refactoring constantly to maintain healthy software	<b>TWO KINDS OF REFACTORING</b>	<b>Root Canal Refactoring</b> Programmers refactoring in clumps to fix unhealthy software.
---	---------------------------------	---

### HOW TO REFACTOR

Refactoring is interleaved with programming. Refactoring is done whenever and wherever programmers decide it should be done. Refactoring is usually done in small, manageable pieces. Refactorings must immediately contribute to a programming goal.	Programming and refactoring are distinct activities. Refactoring is put off; software accumulates <i>technical debt</i> . Refactoring is done in big chunks when (and if) programmer cycles can be spared. Refactorings contribute to potential future goals or to avoiding anticipated pitfalls.
--	--

### RECOMMENDATIONS AND WARNINGS

"Refactoring is something you do all the time in little bursts. You don't decide to refactor, you refactor because you want to do something else, and refactoring helps you do that other thing." — Fowler, 1999	"Avoid the temptation to stop work and refactor for several weeks. Even the most disciplined team inadvertently takes on design debt, so eliminating debt needs to be an ongoing activity. Have your team get used to refactoring as part of their daily work." — Shore, 2004	"Harder to apply than expected, ... both time consuming and error prone, ... the usefulness of refactoring for restructuring purposes without concrete need is doubtful because it is unclear whether the increased beauty will simplify or aggravate future changes." — Pizka, 2004, describing a 6-month refactoring effort.	"The 'number of duplicated code blocks... increased dramatically. This increase could be tracked down to some of our refactoring activities." — Borquin and Keller, 2007, describing a 7-month refactoring effort.
---	--	---	---

### WHEN PROGRAMMERS REFACTOR

Refactoring happens every day during the life of software. 	Refactoring happens in big clumps during certain, set-aside periods. 
--	--

CHARACTERISTICS OF FLOSS TOOLS	Symbols indicate characteristic of tool (shape) and which kind of refactoring is supported (color)	CHARACTERISTICS OF ROOT CANAL TOOLS
Tool enables fast task switching between editing and refactoring		Tool enables execution of multiple refactorings in sequence
Programmer directs which program elements should be refactored		Tool directs or advises which elements should be refactored
Tool automates what the programmer would normally do manually		Tool suggests refactorings to perform

### CODE SMELL DETECTORS

<b>jCosmo</b> Detector runs over entire source code and displays all smells at once. 	<b>Hayashi and Colleagues</b> Smells based on programmers' copy and paste behavior. 
---	--

### REFACTORING ACTIVATORS

<b>UML</b> Refactorings activated by dragging UML objects 	<b>Pie Menu</b> Refactorings activated in-context. 
--	---

### REFACTORING SCRIPTS

<b>Roberts and Brant</b> Transformation scripts written by hand. 	<b>iXj</b> Transformation scripts written by generalizing examples from code. 
---	--

### TRANSFORMATION ENGINES

<b>Guru</b> Hierarchy wide changes caused by long-running, tool-directed refactorings. 	<b>X-Develop Extract Method</b> Non-modal configuration, in-line rename for programmer-directed refactoring. 
---	---

M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, Refactoring: Improving the Design of Existing Code: Addison-Wesley Professional, 1999.  
 J. Shore, "Design Debt", Software Proficiency Newsletter, February 2004.  
 E. Van Emden and L. Moonen, "Java Quality Assurance by Detecting Code Smells," presented at WCSE 02: Proceedings of the Ninth Working Conference on Reverse Engineering, 2002.  
 S. Hayashi, M. Saeki, and M. Kurahara, "Supporting Refactoring Activities Using Histories of Program Modification," IEICE Transactions on Information and Systems, 2006.  
 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.  
 M. Boshemian, S. Graham, and M. Hearst, "Aligning Development Tools with the Way Programmers Think About Code Changes," presented at 2007 SIGCHI Conference on Human Factors in Computing Systems, 2007.  
 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. CSREA Press.  
 F. Borquin and R. Keller, "High-impact refactoring based on architecture violations," Proceedings of OSMR 2007.  
 D. Astels, "Refactoring with UML," In Proc. 3rd Int'l Conference on eXtreme Programming and Flexible Processes in Software Engineering, pp. 67-70, Alghero, Italy (2002).  
 Pie Menu Research in Progress. (http://multiview.cs.pdx.edu/refactoringactivator/)  
 L. Moore, "Automatic inheritance hierarchy restructuring and method refactoring," In Proceedings of the 11th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, ACM Press, New York, NY, 235-250, 1996.  
 X-Develop. Omnicore Software, 2007.