High Velocity Refactorings in Eclipse

Emerson Murphy-Hill and Andrew P. Black
Portland State University
1900 SW 4th Avenue
Portland, OR 97201
{emerson,black}@cs.pdx.edu

ABSTRACT
In Eclipse, and in most other development environments, refactorings are activated by selecting code, then using a menu or hotkey, and finally engaging in a dialog with a "wizard". However, selection is error-prone, menus are slow, hotkeys are hard to remember, and wizards are time-consuming. The problem is that as a consequence, refactoring tools disrupt the programmer's workflow and are perceived to be slower than refactoring by hand. In this paper we present two new user interfaces to Eclipse's existing refactoring engine: marking menus and refactoring cues. Both are designed to increase programming velocity by keeping the tool out of the programmer's way.

Categories and Subject Descriptors
D.2.3 [**Software Engineering**]: Coding Tools and Techniques;
D.2.6 [**Software Engineering**]: Programming Environments.

General Terms
Design, Reliability, Human Factors

Keywords
Refactoring, tools, usability, environments

1. INTRODUCTION
Refactoring is the process of changing the structure of code without changing the way that it behaves [3]. Refactoring accounts for a significant portion of software development effort. For instance, Xing and Stroulia estimate that in Eclipse's JDT project, 70% of changes may be due to refactoring [19]. Refactoring can be semi-automated with a tool that can, in principle, perform a refactoring faster than a programmer can do it by hand, and without inadvertently changing program behavior.

In Eclipse, several steps are normally required to use a refactoring tool. First, the programmer must select a program element to be refactored, either by selecting code in the editor or by selecting code in a view, such as the Outline or Package Explorer. Second, the programmer initiates the refactoring via a system menu, context menu, or hotkey. Eclipse responds by bringing up a several-step wizard before finally performing the refactoring.

The dominant method of refactoring, *floss refactoring* [12], takes place when programmers refactor frequently to maintain healthy code. Floss refactoring is subsidiary to higher-level programming goals [3] and is mixed with other programming tasks [18]. As a consequence, it is vital that refactoring tools do not slow programmers down or distract them from their primary goals.

Copyright is held by the author/owner.
OOPSLA Eclipse Technology Exchange October 21–22, 2007,
Montréal, Québec, Canada.
ACM 978-1-59593-865-7/07/0010.

2. THE PROBLEM
We have found that the user interface for refactoring in Eclipse does not always support floss refactoring. When we surveyed 28 programmers at Agile Open Northwest 2007 who spend at least 10 hours per week programming and have refactoring tools available at least 90% of the time, 40% reported that they sometimes did not use refactoring tools because they could refactor faster by hand. There are a number of possible reasons for this: it can be difficult to select code suitable as input to a refactoring tool; initiating a refactoring can be slow or unmemorable; and configuration of refactorings is sometimes unnecessarily complex. Eclipse's refactoring user-interface is typical of those in other integrated development environments, which have been typically been built in the same style as the original Refactoring Browser [16], so refactoring tool problems in Eclipse are representative of problems in most environments.

Selecting code as input to a refactoring tool is sometimes difficult because code can be long, complex, or unreadable, so a programmer may waste time trying to make an appropriate selection, or completely give up [11]. Furthermore, it is not always obvious what should be selected as the input to a refactoring tool. For example, exactly what should be selected when you want to extract an interface from a class: a class name, a class reference, the text of a whole class definition, or the file that defines the class? Eclipse also poses a difficulty for the programmer because selecting multiple program elements for the same refactoring is not consistently supported. For instance, while a programmer can use the outline view to select a dozen constants for moving to another class, selecting a dozen constants for renaming is not supported at all.

Initiating a refactoring can also be a problem. For instance, a programmer can initiate a refactoring using a system menu, but the refactoring system menu has become so long that it can be hard to navigate. One Eclipse user complained to us that the "menu is too big sometimes, so searching [for] the refactoring takes too long." The context menu suffers from the same problem, and also imposes the burden of searching a large parent menu for the "Refactoring" submenu. Hotkeys would seem to be the ideal alternative, but can be difficult to learn and recall. We believe programmers must make a triple cognitive mapping: from their mental model of the structural change that they want to make (like "put this code into a new method"), to a capriciously named

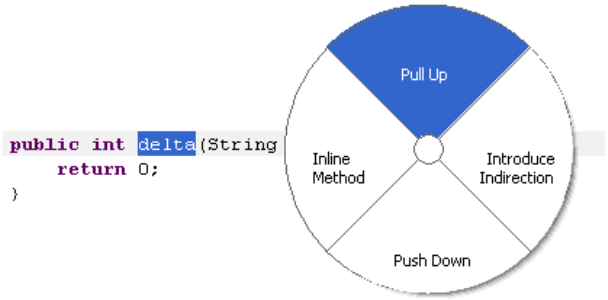


Figure 1. A marking menu for refactoring in Eclipse.

refactoring (EXTRACT METHOD), and then to a key combination (Alt+Shift+M, remembering that Alt+Shift means refactor, and M means EXTRACT METHOD rather than MOVE METHOD). The middle step can be especially confusing, given that refactorings are not assigned universally meaningful names (and we suspect that none exist). In practice, perhaps for these reasons, developers don't often use refactoring hotkeys. For instance, Murphy and colleagues showed that among 41 heavy Eclipse users, none ever used hotkeys to initiate the PULL UP refactoring, the fourth most popular refactoring in Eclipse [10].

Finally, configuring a refactoring can slow a programmer down. In Eclipse, refactorings are typically configured using a modal "wizard" interface; this forces programmers to go through configuration steps that may not be necessary, and which may induce visual disorientation [2]. One exception is Eclipse's in-line RENAME, an excellent example of a refactoring user interface that unobtrusively gathers the required configuration information. Unfortunately, it is not clear how to extend this interface to other refactorings.

3. ALTERNATIVE USER INTERFACES

We have prototyped two tools designed to alleviate the problems discussed in the last section. Because these tools have very dynamic interfaces, we have provided short screencasts at http://multiview.cs.pdx.edu/refactoring/activation.

3.1 Marking Menus

Marking menus (an extension of pie menus [1]) were originally designed as an alternative to context menus, applicable in a wide variety of applications [6]. Rather than having menus items displayed in a vertical list, marking menu items display radially. Items in a marking menu can be selected with the mouse just like items in context menus, but they can also be selected by invoking the menu and gesturing in the direction of the required item.

We implemented marking menus in SWT, based on a Swing implementation from the University of California [4]. Figure 1 shows an example of a refactoring marking menu; this menu was displayed after a programmer selected the method name in a method declaration, held down the center mouse button, and gestured upwards with the mouse. Up to four refactoring options are shown per refactoring marking menu, with no submenus. After a refactoring menu item is chosen, the refactoring is applied without a wizard. When the transformation is complete, the user may change the names of any generated variables using Eclipse's standard in-line rename.

As with context menus, the items in a marking menu depend on the current selection. For instance, the right menu item is EXTRACT METHOD when statements are selected, but EXTRACT LOCAL VARIABLE when an expression is selected. Table 1 shows the refactorings that are displayed in each direction when various pieces of code are selected. An empty cell means there is no refactoring assigned to that location. The design rationale is that "up" means move up the hierarchy, "down" means move down the hierarchy, "left" means specialize, and "right" means generalize.

Table 1. Refactoring assignments in marking menus

Selected Code	Up	Down	Left	Right
Method	PULL UP	PUSH DOWN	INLINE	INTRODUCE INDIRECTION
Field	PULL UP	PUSH DOWN		ENCAPSULATE
Local Variable			INLINE	CONVERT TO FIELD
Constructor				INTRODUCE FACTORY
Statement(s)				EXTRACT METHOD
Anonymous Class				CONVERT TO NESTED
Nested Class			CONVERT TO ANONYMOUS	CONVERT TO TOP LEVEL

3.1.1 Advantages of Marking Menus

In general, marking menus have two advantages over traditional context menus. First, marking menus can initiate operations faster than traditional context menus because the user does not have to precisely position the mouse in a small rectangle; instead, she need only aim in a particular direction. Furthermore, once the user has learned where the desired item is located, she does not have to read the menu or even wait for the menu to be painted on the screen: she need only make a mouse gesture in the desired direction. Second, the locations of items on marking menus become memorable with frequent use because marking menus exploit muscle memory.

When applied to refactoring, marking menus have some additional advantages. First, because our implementation limits the number of menu items to four, a simple key-press scheme could be used for programmers who prefer not to use the mouse. For instance, pressing Alt+Shift in the editor could display the menu, then pressing an arrow key could invoke the desired menu item. Second, we claim that marking menus are an especially intuitive refactoring initiation mechanism, because many refactorings are intuitively directional (e.g., PULL UP), conceptually similar (e.g., INLINE LOCAL VARIABLE and INLINE METHOD), and have inverses (e.g., INLINE METHOD and EXTRACT METHOD). We are in the process of validating this claim experimentally.

To summarize, we believe that using marking menus to initiate refactorings will give the programmer the speed of hotkeys without the difficulty of remembering them.

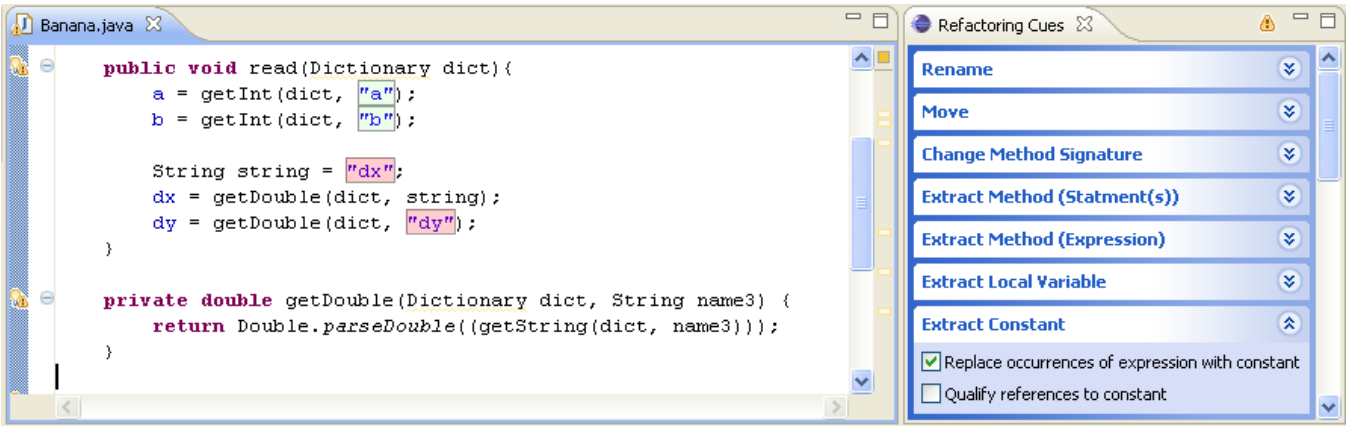


Figure 2. Refactoring cues, ready to perform the EXTRACT CONSTANT refactoring on two string literals, "dx" and "dy". Each refactoring is listed on an unexpanded *ExpandItem* in the Refactoring Cue view at right. Here, the EXTRACT CONSTANT *ExpandItem* is expanded, revealing configuration options for that refactoring.

3.1.2 Limitations of Marking Menus

While context and system menus may accommodate any number of refactorings, marking menus can accommodate only a limited number before submenus must be used. This is especially problematic in our implementation, when we limit the number of menu items to four. Submenus could be incorporated into our implementation, but to the detriment of memorability and speed. Furthermore, because users tend to rely more and more on menu positioning over time [7], disrupting old menus by adding new refactorings can decrease usability.

Using our design rationale of "left is specialize, right is generalize," some combinations of program elements and directions are ambiguous. For example, gesturing right on an expression might mean either EXTRACT METHOD or EXTRACT LOCAL VARIABLE. In such cases, we assign the smallest refactoring, reasoning that larger refactorings can be realized in multiple steps. For example, to EXTRACT METHOD from an expression, programmers can perform EXTRACT LOCAL VARIABLE, followed by EXTRACT METHOD on the resulting assignment statement, and then INLINE LOCAL VARIABLE. Such a multi-step refactoring can be performed quite quickly using marking menus.

Finally, a limitation of our design rationale is that some refactorings are not supported. For instance, two currently popular refactorings in Eclipse, RENAME and MOVE, are conspicuously absent. We are currently researching how these refactorings can be included in our design rationale, but we feel that an initiation mechanism supporting all refactorings is unnecessary, because programmers already use different initiation mechanisms for different refactorings [10].

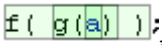
3.2 Refactoring Cues

Refactoring cues are editor highlights that indicate valid candidates for refactorings, and also provide a way to configure those refactorings. When using refactoring cues, a programmer first selects which refactoring that she wants to perform from several *ExpandItem* widgets displayed adjacent to an editor. The *ExpandItem* then expands to reveal the refactoring configuration options (Figure 2, right), and the code elements appropriate for the refactoring are highlighted in the editor (Figure 2, left). The programmer then indicates on which of the program elements she wishes to perform the refactoring by clicking somewhere in the corresponding highlight, turning that highlight from green to red. When satisfied, the programmer presses a button or hotkey (the same for all refactorings) to execute the desired refactorings. The highlights are then removed and the *ExpandItem* is collapsed.

3.2.1 Advantages of Refactoring Cues

There are several advantages to using refactoring cues over traditional methods of activating refactorings. First, the programmer can select multiple pieces of code as input to the refactoring, and can do so consistently for all refactorings. Second, the program elements that are appropriate as input to the refactoring engine are explicitly displayed, so that the programmer doesn't have to wonder what to select before initiating the refactoring. Third, configuration options are displayed non-modally and can be changed on demand, increasing the speed at which a refactoring can be performed and reducing visual disorientation. Fourth, performing the same refactoring on multiple program elements is handled consistently for all refactorings. In contrast, refactoring multiple program elements is supported in Eclipse, but only for certain refactorings (such as PULL UP) and when using certain views (such as the Outline).

3.2.2 Limitations of Refactoring Cues

Refactoring cues also have some limitations. In particular, the desired cue might be difficult to recognize when refactoring candidates are nested. For example, there are 3 nested EXTRACT METHOD candidates in  , each candidate darker than its parent. We have attempted to limit nesting by specializing refactorings (for example, by dividing EXTRACT METHOD into EXTRACT METHOD FROM STATEMENT(S) and EXTRACT METHOD FROM EXPRESSION, but this complicates the view. Also, because all refactorings must be shown in the Refactoring Cues view, it can take some time for a programmer to find a particular refactoring. This could possibly be alleviated by grouping the refactorings or by providing a search feature.

4. RELATED WORK

Callahan and colleagues have shown that in controlled experiments, pie menus are 15% faster than linear menus [1]. These authors also suggest that there are some domains for which

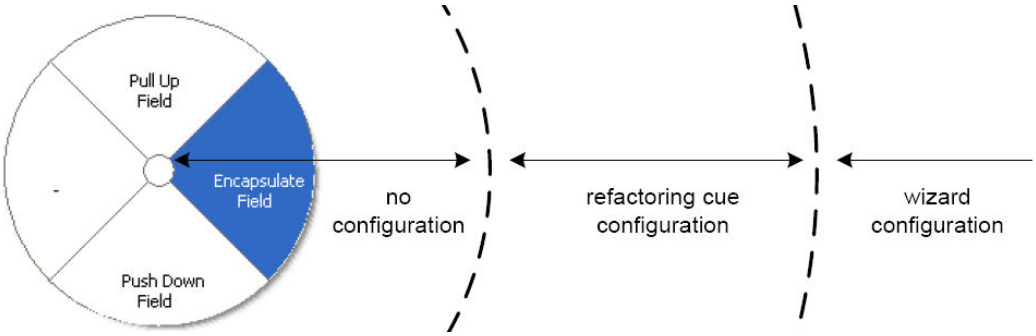


Figure 3. Refactoring marking menus with distance-from-center indicating configuration type.

marking menus are especially well suited; we believe that refactoring is one of these. When using marking menus, Kurtenbach and Buxton showed that users' selection time decreases over a prolonged period [7]; they have also shown that users of menus similar to ours have less than 3% error rates and take about 0.75 seconds to select a menu item [6].

Some other environments have touched on manipulation of code through gestures. Both IntelliJ IDEA [5] and Netbeans [9] have plugins that allow programmers to use gestures to initiate user-assignable commands, including refactoring. Ougi, a 3D environment for novice programmers, supports copying and moving program elements with virtual hand gestures [13].

Refactoring cues were inspired by several existing tools. First, the cues themselves are visually quite similar to Lommerse and colleagues' Visual Code Navigator, which colors different types of program elements to aid in program comprehension [8]. Refactoring cues also have a look and behavior similar to Box View, a tool that allows the programmer to select program statements, although Box View operates in a separate view rather than overlaid on program text [12]. Finally, refactoring cues are somewhat similar to domain/j: both tools provide a separate view containing a list of available refactorings and both assist the programmer in selecting code [14]. However, domain/j requires a code region to be selected first, after which the refactoring is applied to it; refactoring cues reverse this ordering.

While we have portrayed hotkeys, linear menus, and refactoring-wizards as typical refactoring tool interfaces, one exception is worth mentioning. In Eclipse, the MOVE CLASS refactoring can be activated by dragging an existing class to a different package. We feel that this is an especially memorable way to activate MOVE refactorings, but it is not obvious how to extend it to other refactorings.

5. EVALUATION

We are in the process of conducting evaluations on both of our tools. In a memory recall experiment, we are trying to determine how well programmers' mental models of refactoring map to our design rationale for refactoring marking menus. In a selection experiment, we are trying to determine how quickly and accurately programmers can select cues. In a survey and tool demonstration with professional programmers, we are trying to assess if, and how, refactoring marking menus and refactoring cues can be an effective part of a programmer's toolset.

6. FUTURE WORK

As we continue our research on marking menus, we plan on making both functional and interface improvements. Functionally, refactoring marking menus can be naturally extended to include other refactorings, such as left and right menu items on visibility modifiers to decrease or increase visibility. We are also investigating alternative design rationales to accommodate more refactorings on the marking menus, as well as to be more memorable for the programmer. In the marking menu literature, many alternatives and improvements to the standard marking menu interface have been proposed [17], and we will continue to investigate how these alternatives may improve usability when applied to refactoring. For example, several programmers have told us that they feel that marking menus are a waste of screen space, and so we are considering using a "labels only" display [17].

We plan on investigating user-interface modifications to refactoring cues as well. One area in need of improvement is nested refactoring cues, including determining the optimal contrast between parent and child cues, and how to accommodate color-blind programmers. We would also like to investigate how code smells, especially duplication, can be effectively displayed using cues. We are also looking into expanding refactoring cues to allow cue selection from multiple editors and different views.

We are also investigating how to integrate marking menus and refactoring cues. We would like to accomplish this by allowing marking menus to initiate a refactoring with no configuration (as in the current implementation), configuration through refactoring cues (allowing the programmer to select more code to refactor), or configuration through standard Eclipse wizards, depending on the mouse distance from the marking menu's center (Figure 3). Heuristically, the greater the distance from the marking menu's center, the more heavyweight the configuration. This approach makes our marking menus much more like control menus [15].

These two tools assist in three phases (selection, initiation, and configuration) of the larger refactoring process. We plan on investigating at least three other phases as well: smell detection, understanding precondition violations, and understanding refactoring results. For example, we will build alternatives to the standard Eclipse refactoring preview to help programmers understand what code was changed during refactoring(s). An alternative mechanism is especially important when programmers use fast refactoring activation mechanisms, such as marking

menus, because it is necessary to provide immediate feedback to quickly and accurately inform the programmer which program elements have changed, and how.

7. CONCLUSION

The availability of refactoring tools that promote continuous refactoring is a major selling point for IDEs like Eclipse: it is the bait that attracts programmers away from emacs and other editors. However, there is a danger that programmers will become disillusioned with Eclipse if they find that the tools that it provides are slow and cumbersome to activate, that is, if they find that instead of assisting in their workflow, the tools get in the way.

In this paper we presented two new mechanisms for activating refactorings. These mechanisms were designed to avoid introducing unnecessary modality and to have low conceptual and physical overhead. We hope that future user studies will show that they increase the usability of Eclipse's refactoring tools, and thus play a vital part in promoting the adoption of Eclipse by professional programmers, and thus in increasing their productivity and velocity.

8. ACKNOWLEDGMENTS

Thanks to our anonymous reviewers for their helpful comments and the National Science Foundation for partially funding this research under grant CCF-0520346.

9. REFERENCES

[1] Callahan, J., Hopkins, D., Weiser, M., and Shneiderman, B. 1988. An empirical comparison of pie vs. linear menus. In *Proc. of the SIGCHI Conference on Human Factors in Computing Systems* (Washington, D.C., United States, May 1988). J. J. O'Hare, Ed. CHI '88. ACM Press, New York, NY, 95-100.

[2] de Alwis, B. and Murphy, G. C. 2006. Using Visual Momentum to Explain Disorientation in the Eclipse IDE. In *Proc. of the Visual Languages and Human-Centric Computing* (September 2006). VLHCC '06. IEEE Computer Society, Washington, DC, 51–54.

[3] Fowler, M. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Publishing Co., Inc.

[4] Hong, J. I. and Landay, J. A. 2000. SATIN: a toolkit for informal ink-based applications. In *Proc. of the 13th Annual ACM Symposium on User interface Software and Technology* (San Diego, California, United States, November 2000). UIST '00. ACM Press, New York, NY, 63–72.

[5] IDEA Mouse Gestures. Accessed August 2007. http://www.smardec.com/products/idea.html.

[6] Kurtenbach, G. and Buxton, W. 1993. The limits of expert performance using hierarchic marking menus. In *Proc. of the INTERCHI '93 Conference on Human Factors in Computing Systems* (Amsterdam, The Netherlands). S. Ashlund, A. Henderson, E. Hollnagel, K. Mullet, and T. White, Eds. IOS Press, Amsterdam, The Netherlands, 482–487.

[7] Kurtenbach, G. and Buxton, W. 1994. User learning and performance with marking menus. In *Conference Companion on Human Factors in Computing Systems* (Boston, Massachusetts, United States, April 1994). C. Plaisant, Ed. CHI '94. ACM Press, New York, NY, 218.

[8] Lommerse, G., Nossin, F., Voinea, L., and Telea, A. 2005. The Visual Code Navigator: An Interactive Toolset for Source Code Investigation. In *Proc. of the 2005 IEEE Symposium on Information Visualization* (October 2005). INFOVIS '05. IEEE Computer Society, Washington, DC, 4.

[9] Mouse Gestures Plugin to Netbeans. Accessed August 2007. https://mousegestures.dev.java.net.

[10] Murphy, G. C., Kersten, M., and Findlater, L. 2006. How Are Java Software Developers Using the Eclipse IDE?. *IEEE Software.* 23, 4 (July 2006), 76–83.

[11] Murphy-Hill, E. 2006. Improving usability of refactoring tools. In *Companion To the 21st ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA, October 2006). OOPSLA '06. ACM Press, New York, NY, 746–747.

[12] Murphy-Hill, E. and Black, A. 2007. Why don't people use refactoring tools? In *Proc. of the 1st Workshop on Refactoring Tools*. ECOOP '07. TU Berlin, ISSN 1436-9915.

[13] Osawa, N., Asai, K., Sugimoto, Y. Y., and Saito, F. 2001. A Dancing Programmer in an Immersive Virtual Environment. In *Proc. of the IEEE 2001 Symposia on Human Centric Computing Languages and Environments (HCC'01)* (September 2001). HCC '01. IEEE Computer Society, Washington, DC, 348.

[14] Perera, R. 2004. Refactoring: to the rubicon… and beyond!. In *Companion to the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications* (Vancouver, BC, Canada, October 2004). OOPSLA '04. ACM Press, New York, NY, 2–3.

[15] Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. 2000. Control menus: execution and control in a single interactor. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems* (The Hague, The Netherlands, April 2000). CHI '00. ACM Press, New York, NY, 263–264.

[16] Roberts, D., Brant, J., and Johnson, R. 1997. A refactoring tool for Smalltalk. *Theory and. Practice of Object Systems* 3, 4 (October 1997), 253–263.

[17] Tapia, M. A. and Kurtenbach, G. 1995. Some design refinements and principles on the appearance and behavior of marking menus. In *Proc. of the 8th Annual ACM Symposium on User interface and Software Technology* (Pittsburgh, Pennsylvania, United States, November 1995). UIST '95. ACM Press, New York, NY, 189–195.

[18] Weißgerber, P. and Diehl, S. 2006. Are refactorings less error-prone than other changes?. In *Proc. of the 2006 International Workshop on Mining Software Repositories* (Shanghai, China, May 22–23, 2006). MSR '06. ACM Press, New York, NY, 112–118.

[19] Xing, Z. and Stroulia, E. 2006. Refactoring Practice: How it is and How it Should be Supported—An Eclipse Case Study. In *Proc. of the 22nd IEEE international Conference on Software Maintenance* (September 2006). ICSM '06. IEEE Computer Society, Washington, DC, 458–468.