Usability of Refactoring Tools for Java Development

Jeffrey Mahmood
Department of Software Engineering
Rochester Institute of Technology
38 Lomb Memorial Drive
Rochester, NY 14623, USA
ph: 001-585-7295385
jim9679@rit.edu

Y. Raghu Reddy
Department of Software Engineering
Rochester Institute of Technology
38 Lomb Memorial Drive
Rochester, NY 14623, USA
ph: 001-585-4757609
raghu@se.rit.edu

ABSTRACT
Refactoring tools can be used by software developers to prevent human error, and maintain current behavior of the software system. However surveys have found that refactoring tools are not used by a majority of developers, rather they prefer to refactor by hand. Various reasons mentioned by developers for not using refactoring tools were the lack of user control, confusing error messages, and the number of steps necessary to perform a single refactoring. Other studies have noted the lack of automation and absence of multi-stage integration of refactoring tools as barriers to their usage and adoption. In this paper, we present the results of usability evaluation for Java based refactoring tools IntelliJ IDEA, JBuilder, and RefactorIT against 34 usability guidelines. The results found that the refactoring tools have not shown any major improvement in recent years. We recommend some improvements to existing refactoring tools.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]**:** *Distribution, Maintenance, and Enhancement* - Restructuring, reverse engineering, and reengineering

General Terms
Design, Measurement, Reliability

Keywords
Refactoring, Usability, IDE

1. INTRODUCTION
Refactoring is a technique for changing existing source code to improve the design without changing the external system behavior [7]. At times developers/maintainers may inject compile time or runtime errors due to carelessness, lack of understanding, or overly complex code while refactoring. Refactoring can be a time consuming task and various factors contribute to the complexity of refactoring. The amount of time consumed in refactoring mainly depends on the size of the system, the amount of refactoring, availability of tools, and the developer's understanding of the system.

Refactoring tools can be used by developers to prevent errors and to perform refactoring tasks faster. However, a survey of programmers at Agile Open Northwest 2007 revealed that although 90% of the developers had access to refactoring tools, only 40% used the tools available [4]. The developers cited being able to refactor by hand faster, too many steps or screens, complex key sequences, linear menus, code selection, and no general purpose mechanism for refactoring several pieces of code at once as some of the reasons for not using any software refactoring tools. In addition some of the other reasons for not using software refactoring tools was the lack of automated support and multi-stage integration [1,2].

Developer's understanding of the system also played a key role in using refactoring tools. Sometimes developers are unsure of a specific refactoring that may need to be applied and lack of good tool support enhances this problem. Providing tool support for identification and selection of the refactoring type can reduce the amount of time a developer spends on refactoring. At the same time, it is important to characterize the complexity of the refactoring to be undertaken, and automated tools can aid developers in prioritizing their efforts [9].

The goal of this study is to evaluate the usability of three different software refactoring tools, compare the results to other studies of similar nature, and suggest improvements for tools that can help reduce the perceived ineffectiveness of software refactoring tools.

The rest of the paper is organized as follows Section 2 provides background information on usability and refactoring. Section 3 presents the methodology used for the study and Section 4 presents the results of the study. Section 5 provides suggestions for improvements. Section 6 gives a brief overview of some related work. Section 7 presents conclusions and provides some insights into future work.

2. BACKGROUND
According to Henry [8], a large software usability gap can lead users getting confused, frustrated, or panicked, and can result in the software system being misused or not used at all. A software system should be easy to use, quick, and pleasant in order to promote learning and recall for end-user supported tasks. The consistency of software applications usability has been shown to reduce user training times by 25% to 50% [8]. The lack of developers/maintainers using software refactoring tools suggests that they suffer from a large software usability gap.

Refactoring manually requires developers to validate their refactorings by updating the affected modules to compensate for the changes. Li et al. [6] agree that compensating for a refactoring is a flexible but fault-prone method for validating a refactoring. The other types of validation for refactoring are preservation of pre and post-condition. Precondition check is the method used by most software refactoring tools, and is defined as preserving all the interactions of the code involved before allowing the refactoring to take place. A post-condition check relies on testing the code after a refactoring and does not apply to refactoring tools since it requires a test suite to verify the changes [6]. For most refactoring tools, when a pre-condition is violated the user is notified via an error message.

IWRE'10, Feb 25, 2010, Mysuru, Karnataka, India

TABLE 1: USABILITY GUIDELINES COMPLIANCE SIGNIFICANT RESULTS

	IntelliJ 7.0.4	Jbuilder 2008	RefactorIT 2.7beta
Errors			
- Strategy for error recovery	1	1	1
- Permit reversal of actions	5	5	5
- Automate error-prone tasks	1	1	1
Ease of use			
- Automate tedious/time consuming tasks	1	1	1
User Control			
- Adapt to the Users, customization	3	3	2
- Allow Users to control detail, error messages, style	1	1	1
Usability Totals	118	114	94
Percentage Usability Guidelines Compliance	69%	67%	55%

TABLE 2: USABILTY COMPLIANCE OF MEALY ET AL. ANALYSIS

Refactoring Tool	Consistency	Errors	Information Processing	User Experience	Define for User	User Control	Goal Assessment	Total	%
Eclipse 3.2	4	8.5	9	7	7.5	6	3.5	45.5	56%
Condenser 1.05	0.5	5	1	3	2.5	1.5	3	16.5	20%
RefactorIT 2.5.1	4	10.5	7	9	7	4	6	47.5	59%
Eclipse 3.2 with Simian 2.2.12	4	12	9	9	7.5	7	6	54	67%
Total Requirements by Category	4	15	10	10	9	25	8	81	100%

This error message allows the user to identify where the error was made in order to fix it. It is the failure to produce and properly display these error messages that have deterred developers from using software refactoring tools [4, 5].

Automation of refactoring can reduce some of the errors caused by manual refactoring. The benefit of automated tools lies in their ability to be customized. For example, users can set their preferred automation level there by selecting specific refactorings that can be automated. Three suggested levels of automation are Assisted, Global, and Severity based [9]. At the Assisted level code-smells are identified by the IDE and suggested resolutions are provided to the user. The Global level implies full-automation where the IDE automatically resolves any issues found. The Severity based level detects issues the same as the other levels, but only automates a solution based on a complexity threshold specified by the user.

The Agile Open Northwest 2007 survey results [4] imply that the lack of sufficient refactoring tools leads to developers performing manual refactorings. One of the goals of the IDE and refactoring tools should be to facilitate the development of software by providing meaningful tools to minimize defects injected by the developer. Refactoring tools should allow developers to perform code refactorings with relative ease from the syntax level to the component design level to the package level. Providing refactoring tools with high usability has the potential to improve overall code quality and maintainability, and minimize future rework.

This study will use usability guidelines based on Mealy et al. [2] to evaluate three Java refactoring tools, and compare the results to that of their previous study to determine if any improvements have been made.

3. EXPERIMENTAL METHOD

The study used commercial as well as open source tools. The refactoring tools chosen were two commercial IDE's and one open source project which supported Java development. The commercial IDE's selected for this study are IntelliJ IDEA 7.0.4 and JBuilder 2008, and the open source refactoring tool is RefactorIT 2.7beta. In a previous study, Mealy et al. [2] used the RefactorIT 2.5 plugin version. In our study we use the standalone version. IntelliJ IDEA is available as a 30 day trial with all available functionality, and JBuilder 2008 is the free for download, limited functionality release of JBuilder. There are currently two commercially available versions of JBuilder that charge per seat or per license.

For the evaluation of the tools a small scale Master's level student project and a considerably larger open source project was used. The student project was an implementation of a bowling game simulator that had 24 classes and approximately 2000 lines of code. The open source project used was Google Web Toolkit, in particular only its /dev/src module was evaluated and it contained 421 classes and approximately 110,000 lines of code. The bowling alley simulation was used because of its relative ease in understanding, and the open source project Google Web Toolkit was chosen because of its maturity and considerably larger size.

Each software refactoring tool chosen supported more than 20 refactorings, and each one was applied to the two software systems selected. IntelliJ IDEA supported refactorings for Rename, Change Signature, Make Static, Convert to Instance Method, Move, Copy, Safe Delete, Extract Method, Replace Method Code Duplicates, Invert Boolean, Introduce Variable, Introduce Field, Introduce Constant, Introduce Parameter, Extract Interface, Extract Superclass, Use Interface Where Possible, Pull Members Up, Push Members Down, Replace Inheritance with Delegation, Inline, Convert Anonymous to Inner, Encapsulate Field, Replace Temp with Query, Replace Constructor with Factory Method, Generify, and Migrate.

JBuilder 2008 supported refactorings for Rename, Move, Change Method Signature, Extract Method, Extract Constant, Inline, Convert Anonymous Class to Nested, Convert Member Type to Local, Convert Local Variable to Field, Extract Superclass, Extract Interface, Use Supertype Where Possible, Push Down, Pull Up, Introduce Indirection, Introduce Factory, Introduce Parameter Object, Introduce Parameter, and Encapsulate Field.

RefactorIT supported refactorings for Undo, Redo, Add Delegate Methods, Change Method Signature, Clean Imports, Convert Temp to Field, Create Constructor, Create Factory Method , Extract Superclass/Interface, Inline, Introduce Explaining Variable, Minimize access rights, Move, Override/Implement Methods, Pull Up/Push Down, Rename, and Use Supertype Where Possible.

The usability guidelines used to evaluate the usability of the software refactoring tools was comprised of eight categories: Consistency, Errors, User Experience, Ease of Use, Design for the User, Information Processing, User Control, and Goal Assessment. The specific criteria for each category can be seen in *Appendix A*. Each criteria is rated on an integer scale from 1 to 5 based on the compliance agreement of the usability guideline where 1 is strongly disagree, 2 is disagree, 3 is neutral, 4 is agree, and 5 is strongly agree. Each tool evaluated is this study is assumed to provide the adequate set of refactorings necessary for refactoring both applications. So, tool adequacy has not been considered as a category in the usability guidelines.

In their work, Mealy et al. were able to conceive 81 usability requirements that they rated based on 34 Usability guidelines [2]. These usability requirements were not made available at the time of this particular study. For the basis of the comparison, the 1 to 5 scale was used in order to add a finer level of granularity to the assessment of the guidelines in the absence of the individual requirements.

4. RESULTS

The entire set of the compliance scores relative to the refactoring tool is tabulated in *Appendix B – Full Set of Compliance Scores*.

Table 1 shows the raw data for the most significant usability guidelines scores. From the Percentage Usability scores in Table 1 it can be seen that all three tools were similar in their compliance agreement of the usability guidelines. RefactorIT scored lower than IntelliJ IDEA and JBuilder because of its strict precondition validation rules that do not allow the user to modify the code by means of a built-in text editor. This was especially bad in instances where new variables needed to be created to continue a sequence of refactorings, or the introduction of a new parameter in a method signature could not be extracted. As a result, the Ease of Use category for RefactorIT brought down its overall score.

The "*strategy for error recovery*" row in Table 1 refers to the error handling capabilities of the refactoring tools. For each tool the most common way to handle improper use of the refactoring was an obtuse error message. Clearing the error message provided no further assistance about the refactoring and the user is left to empirically figure out what had caused the error. In more than one instance the refactoring was never performed and resulted in manual refactoring.

The tools provide a means of reversing actions when a refactoring is carried out improperly. This is represented in Table 1 by high marks in the "*permit reversal of action*s" row. The tools evaluated allowed reversal of actions by integrating the refactoring tools' undo and redo commands with the overall undo and redo commands of the IDE.

The low scores in the "*automate error prone tasks*" and "a*utomate tedious/time consuming tasks*" rows of Table 1 is a reflection of the lack of automation in the refactoring tools in general.

The User Control rows of Table 1 were derived from the ability to provide variable, parameter, and class names when performing refactorings such as Introduce Field, Introduce Variable, Convert Anonymous to Inner, and Encapsulate Field. The refactoring tools themselves did not allow customizations, but IntelliJ provided a set of templates for creating new files and classes that can be customized.

In Table 2 Mealy et al.'s previous scores can be seen. Again, the scores of the usability guidelines show that there is little difference in terms of usability of the software refactoring tools with the exception of Condenser. A calculation of the percentage of raw points earned in the Mealy et al. study shows that the minimum percentage for compliance was 55% and the maximum was 67%. This is almost the same exact range as the compliance percentages from the Table 1 which show a minimum compliance of 55% and maximum compliance of 69%. This shows that there is no discernable difference between the commercial tools and the open source tools evaluated by Mealy et al., and there is no discernable difference between the previous RefactorIT plug-ins and the RefactorIT standalone version. Since there is no difference between the previous tools evaluated and the current one, we can state that there has been no significant improvement in the usability of software refactoring tools since the previous evaluation. The raw score numbers confirm that the automation of the tools is still non-existent and that user control is still lacking.

Since the goal of the study was to evaluate the usability of the refactoring tools, there was no difference between the usability of the tools between the student project and open source project evaluated and therefore the results are presented as a single table. The two projects proved useful in providing ample opportunities to attempt all the refactorings offered by the tools.

5. DISCUSSION

A major issue not discussed in the previous section that consistently scored low according to the raw data in Table 1 was the Strategy for Error Recovery. In the current state of software refactoring tools whenever a pre-condition is violated an error message is displayed to the user. It was the ineffectiveness of this error message that lead Murphy-Hill and Black to develop their plug-ins for Eclipse to enhance Extract Method [3]. Once an error is encountered, the user is notified and the refactoring is either canceled or allowed to continue based on the user's discretion, and it is up to the user to compensate for any errors injected into the system.

In order to improve the usability of the tool, the tool should instead analyze the user's code selection, and based on the context of the refactoring suggest a set of corrective actions necessary to complete the refactoring. For instance, in the scope of an Extract Method, if the user has an incomplete selection then the refactoring tool should display a corrective actions such as "Did you mean:" where the display is the suggested block of highlighted text that would make the refactoring feasible. The user can then confirm the intended action and the tool can perform the refactoring as expected. In the instance of a Change Signature refactoring, if a method variable name clashes with a parameter name then the tool could suggest appending the name with "param" or "local" for the appropriate case where the suggested suffix is presented in an editable text field. Currently this scenario either brings the user back to the initial refactoring screen or allows the user to continue and leaves them to resolve the error post refactoring. By making the system more tolerant to errors and providing suggestions, the user will be able to more quickly refactor the code and relate it with a more enjoyable user experience.

6. RELATED WORK

Murphy-Hill and Black have developed a series of Eclipse plugins to alleviate problems when performing an Extract Method refactoring [3]. Their Refactoring Annotations tool

helps developers prevent violation of preconditions when selecting sections of code to extract using visual cues [3]. This also aids in minimizing the confusion associated with any error messages the users may see when performing a refactoring incorrectly.

Murphy-Hill and Black included two other tools that provide visual cues in the Eclipse plug-in to alleviate problems with code selection [3, 5]. The Selection Assist tool and Box View provides visual cues for selecting blocks of code when performing Extract Method refactorings. Studies done with students have shown that Selection Assist, Box View, and Annotated Refactoring have dramatically reduced the time it takes to perform an Extract Method as well as dramatically reduced the number of errors caused by improper code selection. Using Selection Assist and Box View correct code selections were shown to increase 16%, and selection time decreased between 25% and 50% [3]. Refactoring Annotations was shown to decrease the time spent on an Extract Method refactoring by almost 75%.

In a previous study Mealy et al. provide a set of usability guidelines of refactoring tools based on 11 different sources of usability evaluation criteria and the ISO 9241-11 software usability guide [2].

In a separate study, Mealy et al. conclude that the tools evaluated do not support the entire refactoring process, the differences between tools were not usability based, and the inspection tools were not yet industrial strength [1]. By not supporting the refactoring process they mean that the current set of tools only aid the developer in a single stage of refactoring. The stages of refactoring that Mealy and Strooper refer to are the identification of code stage, selection of the refactoring stage, and lastly the implementation of the refactoring stage [1]. They also note the lack of user control and customization of the refactoring tools.

7. CONCLUSION AND FUTURE WORK

The results from the comparison of the usability guidelines found that sufficient improvements have not been made to open source as well as commercially available refactoring tools available for Java development. However, in case of Extract Method, one of the refactorings singled out by Martin Fowler as being fundamental to refactoring, work has been done to aid developers in increasing their efficiency with the development of the Selection Assist, Box View, and Refactoring Annotations plug-ins available for Eclipse.

Work is currently being done to address the automation and integration of the entire refactoring process, and applying additional user control to the proposed automation improvements. Identification of problem code is an area of needed improvement, and better identification tools will allow further automation of refactoring either by mapping from a code-smell to a transformation or from a single transformation to numerous source code candidates.

Certain refactorings such as the introduction of constants, removal of unused code, and even extraction of method can be done automatically. These types of code-smells require a simple change and can be carried out automatically. In the case of an Extract Method, performing this refactoring automatically can be done by identifying repeated code and encapsulating the code block into a parameter-constrained method. However such refactoring requires an explicit pre post-condition check. Any type of refactoring that requires redesign, such as long methods, large classes, too many method parameters, or high coupling can be detected and brought to the user's attention, but any type of fix would be at the user's discretion. These types of refactorings can still be identified at the Assisted level, but are most likely to be categorized above the complexity threshold of a severity level automated refactoring.

Incorporating the use of refactoring tools into the educational process can give future developers more exposure to their possibilities. Since the refactorings themselves are universal to object-oriented systems, and the tools show a high correlation of the same refactorings being available this would not be an issue of IDE selection, but rather a lack of exposure to these types of tools.

Proper usage of software refactoring tools reduces the chance of compile time and run time errors that are inherent in manual refactorings. It is the belief of these researchers that a developer's time spent refactoring could be dramatically reduced by removing the barriers of usability for software refactoring tools.

Further evaluation of refactoring tools for Java needs to be carried out to provide a comprehensive analysis. This study focused on tools that had integrated refactoring into their IDE with the exception of RefactorIT. A study of available plug-ins for open source systems such as the Eclipse IDE needs to be conducted to further gauge the current state of the open source IDE. A plug-in for IntelliJ, RefactorJ, was not available at the time and could potentially address some issues pointed out in the study.

8. REFERENCES

[1] E. Mealy and P. Strooper, "Evaluating software refactoring tool support", Proceedings of the Australian Software Engineering Conference (ASWEC '06), Sydney, Australia, April 18-21, 2006, pp:331-340.

[2] E. Mealy et al., "Improving Usability of Software Refactoring Tools", Proceedings of the Australian Software Engineering Conference (ASWEC '07), Melbourne, Australia, April 10-13, 2007, pp. 307-318.

[3] E. Murphy-Hill, "Improving usability of refactoring tools", Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications, Portland, Oregon, USA: ACM, 2006, pp. 746-747.

[4] E. Murphy-Hill, "Activating refactorings faster", Companion to the 22nd ACM SIGPLAN conference on Object oriented programming systems and applications companion, Montreal, Quebec, Canada: ACM, 2007, pp. 925-926.

[5] E. Murphy-Hill and A.P. Black, "Breaking the barriers to successful refactoring: observations and tools for extract method", Proceedings of the 30th international conference on Software engineering, Leipzig, Germany: ACM, 2008, pp. 421-430.

[6] H. Li and S. Thompson, "Tool support for refactoring functional programs", Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation, San Francisco, California, USA: ACM, 2008, pp. 199-203.

[7] Martin Fowler, Refactoring: Improving the Design of Existing Code, The Addison-Wesley object technology series, Addison-Wesley Professional, 1999, ISBN:978-0201485677.

[8] P. Henry, User-Centered Information Design for Improved Software Usability, Norwood, Mass: Artech House, 1998, ISBN:978-0890069462.

[9] M. Z. Drozdz. "A critical analysis of two refactoring tools", Master's Dissertation, University of Pretoria, August 19, 2008.

Appendix A: USABILITY GUIDELINES

Consistency (C)

- (C1) Ensure that things that look the same act the same and things that look different act different.
- (C2) Be consistent with any interface standards (either explicit or implicit) for the domain/environment.

Errors (E)

- (E1) Assist the user to prevent errors (through feedback, constrained interface, use of redundancy).
- (E2) Be tolerant of others.
- (E3) Provide understandable, polite, meaningful, informative error messages.
- (E4) Provide a strategy to recover from errors.
- (E5) Permit reversal of actions/ability to restart.
- (E6) Allow the user to finish their entry/action before requiring errors to be fixed. Do not interrupt the task being completed.
- (E7) Automate error-prone tasks/sub-tasks.

User Experience (UX)

- (UX1) Make interface minimal, simple to understand, organized, without redundancy, socially relevant (especially for communication) and aesthetically pleasing.
- (UX2) Provide the information, or access to the information, needed for a decision when/where the decision is made.
- (UX3) Use the fewest number of steps/screens/actions to achieve the user's goals/

Ease of Use (EU)

- (EU1) Make the system flexible.
- (EU2) Make the system simple to use.
- (EU3) Make the system efficient to use.
- (EU4) Make the system enjoyable to use.
- (EU5) Automate tedious/repetitive/time-consuming tasks/sub-tasks.

Design for the User (DU)

- (DU1) Define the user and match the system to the user.
- (DU2) Use the user's mental model and language (avoid codes).
- (DU3) Automate mundane/computable tasks/sub-tasks.

Information Processing (IP)

- (IP1) Assist the user to understand the system.
- (IP2) Minimize memorization (i.e. reduce short-term memory load), through use of selection rather than entry, names and not numbers, predictable behavior and access to required data at decision points.
- (IP3) Make commands and system responses self-explanatory.
- (IP4) Use abstraction or layered approaches to assist understanding.
- (IP5) Provide help and documentation, including tutorials and diagnostic tools.
- (IP6) Assist the user to maintain a mental model of the structure of the application system/data/task.
- (IP7) Maximize the user's understanding of the application system/task/data at the required levels of detail.

User Control (UC)

- (UC1) Adapt to the user's ability, allow experienced users to use shortcuts/personalize the system, and use multiple entry formats or styles.
- (UC2) Put the user in control of the system, ensure that they feel in control and can achieve what they want to achieve. Allow users to control level of detail, error messages and the choice of system style.

Goal Assessment (GA)

- (GA1) Ensure the user always knows what is happening. Respond quickly, meaningfully, informatively, consistently and cleanly to user requests and actions.
- (GA2) Make it easy for the user to find out what to do next.
- (GA3) Make clear the cause of every system action or response.
- (GA4) Provide an action/response for every possible type of user input/action.
- (GA5) Provide feedback/assessment/diagnostics to allow the user to evaluate the application system/data/tasks.

APPENDIX B: FULL SET OF COMPLIANCE SCORES

	IntelliJ	Jbuilder	RefactorIT
C			
C1	4	4	4
C2	4	4	4
E			
E1	3	4	3
E2	4	3	3
E3	3	2	2
E4	1	1	1
E5	5	5	5
E6	4	4	4
E7	1	1	1
UX			
UX1	3	3	3
UX2	5	5	2
UX3	4	3	3
EU			
EU1	4	4	1
EU2	3	4	1
EU3	4	4	3
EU4	4	4	1
EU5	1	1	1
DU			
DU1	5	4	4
DU2	5	5	3
DU3	2	1	1
IP			
IP1	4	4	3
IP2	4	4	4
IP3	3	3	3
IP4	4	4	4
IP5	5	3	5
IP6	3	3	3
IP7	3	4	3
UC			
UC1	3	3	1
UC2	1	1	1
GA			
GA1	5	5	5
GA2	4	4	3
GA3	4	4	3
GA4	4	4	3
GA5	2	2	3
Total	118	114	94
%	69%	67%	55%