A Comparison of Environmental Support for Computer Program Restructuring

Dear Participant:

My name is Emerson Murphy-Hill, and I am a student at Portland State University. I am beginning a study on program refactoring tools, and would like to invite you to participate.
You are being asked to take part because of your experience in object-oriented programming.

As part of the study, I am interested in your opinions and attitudes about refactoring tools, and hope that the information I collect will help us to better understand these tools. If you decide to participate, you will be asked to select pieces of code and attempt to find errors, as well as provide some information about your education, programming experience, and opinion of the tools we will see. You may not receive any direct benefit from taking part in this study, but the study may help to increase knowledge that may help others in the future.

Any information that is obtained in connection with this study and that can be linked to you or identify you will be kept confidential. Subject identities will be kept confidential
by not keeping participant names with data collected about them.

Participation is entirely voluntary. Your decision to participate or not will not affect your relationship with the researcher or with Portland State University in any way. If you decide to take part in the study, you may choose to withdraw at any time without penalty. Please keep a copy of this letter for your records.

If you have concerns or problems about your participation in this study or your rights as a research subject, please contact the Human Subjects Research Review Committee, Office of Research and Sponsored Projects, 111 Cramer Hall, Portland State University, (503) 725-4288. If you have questions about the study itself, contact Emerson Murphy-Hill at
Department of Computer Science, P.O. Box 751, Portland State University, Portland, Oregon 97207-0751, (503) 725-4036.

Sincerely,
Emerson Murphy-Hill
Graduate Student, Portland State University
A Comparison of Environmental Support for Computer Program Restructuring

Pretest Questionnaire

Time: _____

(Read aloud by test administrator) The following questionnaire is intended for us to get an idea of what your background is. Your answers in no way effect the rest of the experiment, it simply gives us context for interpreting the results of the rest of the test.

EDUCATION

What is your current year in school or highest grade level achieved:

College: *1 2 3 4* Graduate: *1 2 3 4*
Working on: *Bachelor's* *Masters* *Doctorate* *None*

What is your major?

Computer Science Other: _____

EXPERIENCE

Do you have an outside job relating to your field? Job title: _____

How many years have you been programming? _____

Over the last year, about how many hours do per week would you say you spend programming, on average? _____

How proficient, on a scale from 0 to 4, where 0 means "not at all" and 4 means "expert"?
Java 0 *1* *2* *3* *4*
C/C++ 0 *1* *2* *3* *4*

When programming, do you typically use an Integrated Development Environment? *Y / N*
If so, which one(s) and for what % time? _____
If not, what editors do you use instead? _____

On a scale form 0 to 4, how familiar are you with the practice of refactoring? (0 = not at all to 4 = I refactor every time I program)
0 *1* *2* *3* *4*

Do you use any refactoring tools? *Y / N* If so, which ones? Proportion tools to
_____

Ok, that's it for the questionnaire.

Code Selection

Time: _____

Set-up:

- ☐ *Open "Select (Sandbox)"(TRBlockingServerProcessor.java) w/ "Show Selected Only"*
- ☐ *Assure only package manager and editor are open*
- ☐ *Search for all instances of "if" in project*

This task will help us determine how easy it is to select a piece of code. Here we are testing how the tools support you in this task, *not* your individual ability.

In front of you, there is an Eclipse development environment. Your task is to select every **if** block, including the **then**, **else**, and **else if** clauses that might go along with it. Once you are done making a selection, press the escape key ESC, to mark your selection as complete, then move on to the next **if** block. Once you have selected and marked each **if** block in a method, I will open the next method for you.

A few ground rules:

- When you finish selecting if blocks in a method, say "I'm Done."
- If an **if** block is inside of another **if** block, always select the outer one, not the inner one. *[show]*
- It doesn't matter how much white space you select
- You may ask any questions you like, but try to ask them before the actual testing begins, because we are testing for accuracy first, and then speed.
- If anything is particularly difficult or surprises you, don't hesitate to say something after you get to the end of the method.
- If realize you made a mistake after marking a selection with ESC, just move on.

We are going to use three tools for this task, and you will be able to try each one before the tests begin. For each tool, you will work on 3 methods.

[Disable old tool. Open Sample (Sandbox)]

- ☐ [A] For the next three methods, you may use the mouse and the keyboard. You can simply select the **if** blocks using either one; your preference. *[show]*
- ☐ [B] For the next three methods, you will use a tool that shows where the end of a statement is. You can simply select the **if** blocks using a mouse or keyboard, but the tool will tell you where the end of the current block is. *[show]*. Note that if you simply move the cursor, you'll have to wait a second before the highlight shows up, but if you start dragging the mouse, the highlight changes almost immediately. *[show]*
- ☐ [C] For the next three methods, you will use a tool that shows an outline of the source code. If you select a full statement in the code, a corresponding rectangle lights up in green *[show]*. If you select a partial block, a corresponding rectangle lights up in orange *[show]*. If you click a block in the view, the corresponding code is selected *[show]*. For this test, please use the mouse the tool to perform the actual selection.

Now you give it a try. *[Let them try it out, encourage questions][Close Sample (Sandbox)]* In a moment we will begin the actual test. Again, the task is selecting each outer **if-else** block, then pressing ESC. Ready*? [give a few seconds for questions, then open editor]* Please begin.

☐ S+A1 – SimpleDate	☐ S+C2 – StartStop	☐ S+B3 – GanttXML
S+B1 – StartStop	S+A2 – StreamToken	S+C3 – DHTTracker
S+C1 – PyAutoIndent	S+B2 – GanttGraphic	S+A3 – Proxy

% Mouse:

[Disable old tool. Open Sample (Sandbox)]

- ☐ [A] For the next three methods, you may use the mouse and the keyboard. You can simply select the **if** blocks using either one; your preference. *[show]*
- ☐ [B] For the next three methods, you will use a tool that shows where the end of a statement is. You can simply select the **if** blocks using a mouse or keyboard, but the tool will tell you where the end of the current block is. *[show]*. Note that if you simply move the cursor, you'll have to wait a second before the highlight shows up, but if you start dragging the mouse, the highlight changes almost immediately. *[show]*
- ☐ [C] For the next three methods, you will use a tool that shows an outline of the source code. If you select a full statement in the code, a corresponding rectangle lights up in green *[show]*. If you select a partial block, a corresponding rectangle lights up in orange *[show]*. If you click a block in the view, the corresponding code is selected *[show]*. For this test, please use the mouse the tool to perform the actual selection.

Now you give it a try. *[Let them try it out, encourage questions][Close Sample (Sandbox)]* In a moment we will begin the actual test. Again, the task is selecting each outer **if-else** block, then pressing ESC. Ready*? [give a few seconds for questions, then open editor]* Please begin.

☐ S+A1 – SimpleDate	☐ S+C2 – StartStop	☐ S+B3 – GanttXML
S+B1 – StartStop	S+A2 – StreamToken	S+C3 – DHTTracker
S+C1 – PyAutoIndent	S+B2 – GanttGraphic	S+A3 – Proxy

% Mouse:

[Disable old tool. Open Sample (Sandbox)]

- ☐ [A] For the next three methods, you may use the mouse and the keyboard. You can simply select the **if** blocks using either one; your preference. *[show]*
- ☐ [B] For the next three methods, you will use a tool that shows where the end of a statement is. You can simply select the **if** blocks using a mouse or keyboard, but the tool will tell you where the end of the current block is. *[show]*. Note that if you simply move the cursor, you'll have to wait a second before the highlight shows up, but if you start dragging the mouse, the highlight changes almost immediately. *[show]*
- ☐ [C] For the next three methods, you will use a tool that shows an outline of the source code. If you select a full statement in the code, a corresponding rectangle lights up in green *[show]*. If you select a partial block, a corresponding rectangle lights up in orange *[show]*. If you click a block in the view, the corresponding code is selected *[show]*. For this test, please use the mouse the tool to perform the actual selection.

Now you give it a try. *[Let them try it out, encourage questions][Close Sample (Sandbox)]* In a moment we will begin the actual test. Again, the task is selecting each outer **if-else** block, then pressing ESC. Ready*? [give a few seconds for questions, then open editor]* Please begin.

☐ S+A1 – SimpleDate	☐ S+C2 – StartStop	☐ S+B3 – GanttXML
S+B1 – StartStop	S+A2 – StreamToken	S+C3 – DHTTracker
S+C1 – PyAutoIndent	S+B2 – GanttGraphic	S+A3 – Proxy

Method Extraction

Time: _____

Setup:

- ☐ *Open EXTRACT (Sandbox) and Select (Sandbox) (behind)*
- ☐ *Assure "Show Selected Elements is off"*
- ☐ *Remove all search results*

This task will help us determine how well the tools assist you in finding problems with the extract method refactoring.

The extract method refactoring should be applied whenever you have a duplicated piece of code or a method that is too long. You can take the piece of code in which you are interested, put it in a new method with the appropriate parameters and return value, and replace all instances of that code with a call to the new method. I'll show you some examples in a moment.

For this test, the code that I want you to attempt to extract will be clearly marked. We will look at a total of 8 pieces of code, one piece at a time, and you will be asked to try to extract one method for each piece. However, there will be one or more things that make the code impossible to extract. So your job is to figure out why the code cannot be extracted and then select and mark the problem points.

The ground rules for this task are:

- When you are done marking the all problem points, say "I'm Done."
- Your two goals, ordered by importance are:
  1. Firstly, mark each and every problem within the selection (I'll define what you should mark in a minute)
  2. Secondly, try to do it as quickly as possible.

For this task, we will use two tools. For the first few refactorings, we will use a wizard-based tool that will perform the extraction for you automatically. To use the tool to try to perform an extraction, select the desired code and press **Alt-Shift-M** *[Show]*. If the selection is extractable, you will get this wizard. *[Select. Use keys]*. Since this wizard showed up, everything is OK to go ahead with this extraction, although for this test, you'll never need to go farther than this point.

Now that one wasn't problematic. Fortunately, the wizard will give you an idea of what is wrong with the code if there is a problem. During this test, you will see three reasons that you might not be able to extract.

[*Open next method*] The first occurs when the selection makes assignments to two or more variables that are used outside of the selection. You can only assign to one, because

you only have one return value for the extracted method. Here is an example *[try to perform the refactoring, read message]*. In this case, you would select and mark each variable that is assigned which would need to be returned in the extracted code. Note that not every assigned variable needs to be returned; it doesn't if it is not used again later in the code, or if it is an instance variable. For instance, assignments to these variables do not need to be returned *[point to **instanceVar** and **power**]*. In this case, you would just mark each assignment to **sum** and **product**, and the declaration of sum, then say "I'm Done." *[Show]*

[*Open next method*] The second reason you may not be able to extract code is because it contains a **break** or **continue** statement, but the place where it breaks or continues to is outside of the selection. Here is an example of that *[Try to perform the refactoring, read message]*. Here you would mark each offending **break** or **continue** *[show]* and then say "I'm Done." Not every break and continue may be a problem; those whose targets are within the selection are just fine. *[Select whole loop – attempt to extract]*

[*Open next method*] The third and final reason is that there may be a return statement inside of a conditional, so that the piece of code returns sometimes, but not others. Here is an example *[Try to perform the refactoring, read message]*. So you would select every place that returns early *[Show]*. Note that not every return statement is a problem – if the selection always ends in return, it's not a problem. *[Reduce selection – attempt to extract]*

So those are the three reasons that you won't be able to extract a piece of code. When we begin the test, you'll see all of these – some code will exhibit only one problem, some will exhibit multiple problems. Note that this tool will only tell you about the first one it finds, but you will need mark *every* one.

Now you can try the tool. *[Open Select (Sandbox)]*. Again, you'll use the wizard based tool to help you find the problems, then you mark them with ESC. Any questions? *[Pause]* Ok, let's begin.

---

*[Repeat * 5][Open an editor. Wait for them to finish and close the editor.][/Repeat]*

- ☐ E+Y1 – VirtualChannelSelectorImpl
  E+X1 – DownloadManagerImpl
  E+Z4 – ResourceLoadGraphicArea
  E+W1– PyLintVisitor
- ☐ E+Z2 – AZMessageDecoder
  E+W2– PyEdit
  E+X2 – MemoryMappedFile
  E+Y2 – BigInteger

Now we are going to do the same thing, but you will use another tool that performs some analysis, specifically for the extract method refactoring. *[Reopen EXTRACT (Sandbox) and Select (Sandbox)]*

Let's try it with the tool *[Open method]*. Now, if you press **Alt+Shift+N**, the tool will show which variables are involved when you extract a method. Each involved variable is assigned a distinct color. Assignments have black boxes around them *[point]*, while regular read values don't *[point]*. The lines coming in the top of the selection represent the parameters that will be passed in *[point]*. A line coming out from the bottom of the method will be the return value *[point]*. The arrow near the gutter indicates that the piece of code flows from top to bottom *[point]* (more on this in a minute).

Let's look at our first example of a failed refactoring again, and see what the analysis tool shows us. *[Next method, enable tool]*. Now we can see that there are two lines coming out the bottom, representing two outputs. Since we couldn't extract this, these lines have Xs over them. Here's an important point – this tool will only draw an arrow from one assignment per variable. If the selection has multiple assignments that are conflicting, you will need to mark each one. *[Add assignment, show tool again, then how to mark]*.

Now for the second example *[select, enable tool]*. What we notice is there is a line going from the offending **continue** to the target, outside of the selection. Again, the X tells us that this cannot be extracted. Also, notice that there is a little hook coming from one variable *[Show]*. This means that not only is the variable used below the selection, it is also used above within the loop. *[Mark continue]*

Finally, our third example *[select, enable tool]*. Notice now how the gutter arrow comes into play; it is crossed by one or more return statement, again with X's. This tells you that the piece of code returns sometimes, but doesn't at other times. *[Mark return]*

So, now you give it a try. [*Close and reopen editor*]. Please use this graphical tool to analyze the code for this test, then select and mark the problem points. Do not use the extract method wizard for this test. Any questions? *[Pause]*. Please begin.

---

*[Repeat * 5][Open an editor. Wait for them to finish and close the editor.][/Repeat]*

- ☐ E+Y1 – VirtualChannelSelectorImpl
  E+X1 – DownloadManagerImpl
  E+Z4 – ResourceLoadGraphicArea
  E+W1– PyLintVisitor
- ☐ E+Z2 – AZMessageDecoder
  E+W2– PyEdit
  E+X2 – MemoryMappedFile
  E+Y2 – BigInteger

Ok, that’s the end of that test.

Posttest Questionnaire

Time: _____

Please answer the following questions based on your experience during this test. If you feel it necessary, feel free to write notes in the margin to explain your answers.

Helpfulness

1. I found the **mouse/keyboard alone** a helpful mechanism for selecting *if* blocks.
   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree

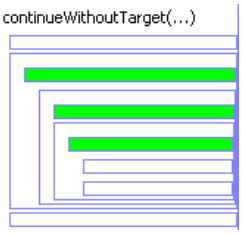
2. I found the **selection assist tool** (green statement highlighter) a helpful mechanism for selecting *if* blocks.

```
public int continueWithoutTarget(){
    int a = 1, b = 2, sum = 0;
    for(int i = 0; sum<10; i++){
        sum = a*i + a*b;
        if(i<5)
            continue;
        else
            System.out.println(sum);
    }

    return sum;
}
```

   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree

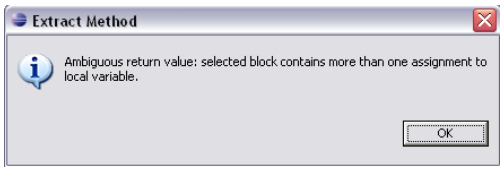
3. I found the **statement box view** (nested boxes off to the side of the code) a helpful mechanism for selecting *if* blocks.



   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree

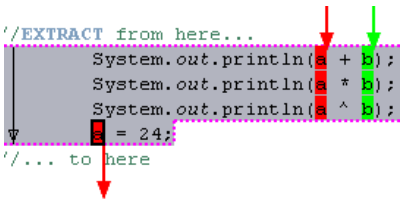
4. I found the **extract method wizard** (popups that contained an error message) a helpful indication of what went wrong when I tried to extract a method.

- ☐ Strongly Disagree
- ☐ Disagree
- ☐ Neutral
- ☐ Agree
- ☐ Strongly Agree



5. I found the **extract method annotations** (colors/lines on top of code) a helpful indication of what went wrong when I tried to extract a method

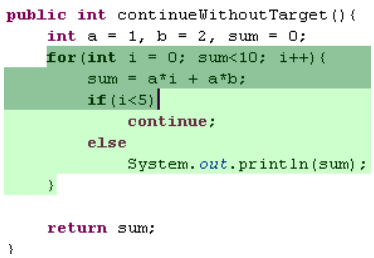
- ☐ Strongly Disagree
- ☐ Disagree
- ☐ Neutral
- ☐ Agree
- ☐ Strongly Agree



Likely to Use Again

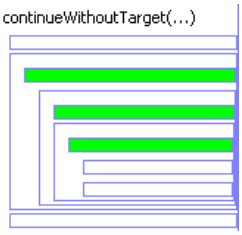
6. If the **selection assist tool** (green statement highlighter) were available for my usual development environment, I would be likely to use it again.

   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree



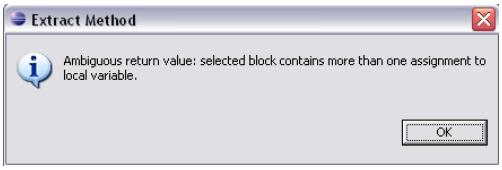
7. If the **statement box view** (nested boxes off to the side of the code) were available for my usual development environment, I would be likely to use it again.

   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree



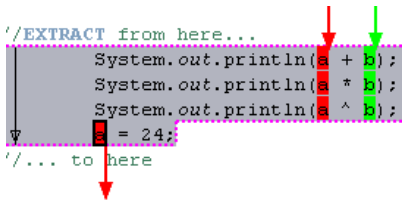
8. If the **extract method wizard** (popups that contained an error message) were available for my usual development environment, I would be likely to use them again during the extract method refactoring.

   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree



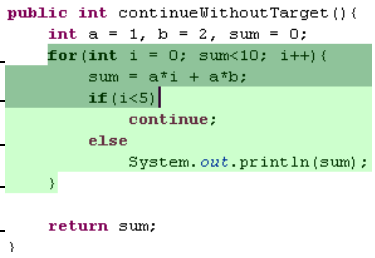
9. If the **extract method annotations** (colors/lines on top of code) were available for my usual development environment, I would be likely to use them again during the extract method refactoring.

   - ☐ Strongly Disagree
   - ☐ Disagree
   - ☐ Neutral
   - ☐ Agree
   - ☐ Strongly Agree

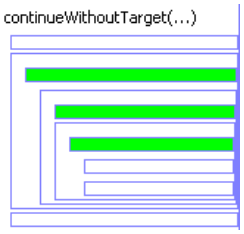


Comments / Suggestions

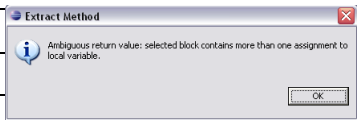
Selection Assist Tool (green statement highlighter)



Statement Box View (nested boxes off to the side of the code)



Extract Method Wizard (popups that contained an error message)



Extract Method Annotations (colors/lines on top of code)