FixBugs User Study Training Script

Hello, my name is .... . We are currently conducting research on improving the usability of static analysis tools. As part of our research, we are conducting a user study on the “Quick Fix”/”Quick Assist” tool features used to fix defects found by static analysis tools.. A “Quick Fix” is offered when the code is broken and a “Quick Assist” is a minor refactoring offered by the tool.

Today, we will be asking you to use and evaluate two different versions of Quick Fix. During this evaluation, you will be asked to complete a set of 3 tasks; each task is comprised of 6 subtasks. In each subtask, you will be given a segment of “buggy code” and asked to fix it. For each set of subtasks, you will be asked to fix the defects either using one of the versions of Quick Fix and/or manually. The first set of tasks should take approximately *4 minutes*. The remaining sets should take approximately *3 minutes*. During the study, we encourage thinking aloud as you are making your decisions. We hope you will share with us your thought process as you are completing the tasks. You will be able to ask minor questions if needed while completing the tasks, but in order to get reliable feedback we hope to avoid this by giving a thorough explanation now and throughout.

Here, you see we have some “buggy” code; there is a missing parameter in the **system.out.printf** method call (*point to show what method*). One way to fix this would be for you to manually go in and make the changes yourself; in this situation you can follow your own workflow and use any features available in the editor. Another option would be to use a quick fix. Now, the first thing to understand about using the quick fixes is how to invoke the tools. There are two ways to invoke a ‘quick fix’: 1) use the keyboard shortcut (Ctrl + 1) or 2) put your cursor on the line where the bug is and wait for the pop up.

Do both methods then ask participant to go ahead and do this.

You can also see that when you hover over the bug marker, a description comes up; FindBugs also provides two views, located at the bottom of the IDE here, that provide more detailed information regarding the defect.
In this example, we are explaining what the problem is, but once we begin we will only give general context to the problem. Just imagine that you are working on some code and have gotten this bug that you need to fix. You can use the information provided by the tool to help you decide what warning means and what the best fix will be. Once you are done with each task, you can move onto the next; once you’ve moved on to the next task you will not be able to return to make any modifications so make sure you are done before moving on.

Now that we’ve explained how to invoke the two versions of quick fix, let’s get started with your tasks so you can get the chance to use them! For the first set of tasks, you will be given buggy code and asked to fix it using Quick Fix version 1. Using quick fix version 1 is fairly straightforward; you invoke the tool using the methods we discussed previously and then click the fix (version 1). The fix will automatically be applied, however, if you are not satisfied with what the quick fix did you can make manual modifications. *Go to Example code to show how it works*. Any questions before you begin?

Okay, now for the next set of tasks you will be asked to manually fix the buggy code you are given; this means neither version of quick fix can be used. We also ask that you attempt to fix the bug without using the Internet for help. Any questions before we begin this portion?

Now, for the last set of tasks we are going to ask you to fix the buggy code using quick fix version 2. Quick fix version 2 is invoked the same way as quick fix version 1. Once you click the fix, it is automatically applied, however, you will be able to make certain modifications. Let's look at the example code again to see how this version of quick fix works.

[Introduction to multi-catch and try-with-resources]

[try-with-resources]
The try-with-resources statement is a try statement that declares one or more resources. A resource is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement. Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource.

The following example reads the first line from a file. It uses an instance of BufferedReader to read data from the file. BufferedReader is a resource that must be closed after the program is finished with it:

```
static String readFirstLineFromFile(String path) throws IOException {
    try (BufferedReader br = new BufferedReader(new FileReader(path))) {
        return br.readLine();
    }
}
```

In this example, the resource declared in the try-with-resources statement is a BufferedReader. The declaration statement appears within parentheses immediately after the try keyword. The class BufferedReader, in Java SE 7 and later, implements the interface java.lang.AutoCloseable. Because the BufferedReader instance is declared in a try-with-resource statement, it will be closed regardless of whether the try statement completes normally or abruptly (as a result of the method BufferedReader.readLine throwing an IOException).

[multi-catch]
In Java SE 7 and later, a single catch block can handle more than one type of exception. This feature can reduce code duplication and lessen the temptation to catch an overly broad exception.

Consider the following example, which contains duplicate code in each of the catch blocks:

```
catch (IOException ex) {
     logger.log(ex);
     throw ex;
catch (SQLException ex) {
     logger.log(ex);
     throw ex;
}
```

In releases prior to Java SE 7, it is difficult to create a common method to eliminate the duplicated code because the variable ex has different types.

The following example, which is valid in Java SE 7 and later, eliminates the duplicated code:

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

The catch clause specifies the types of exceptions that the block can handle, and each exception type is separated with a vertical bar (|).

Go back to first example again and go through invoking the tool/applying the fix -- explain:
- highlighting
- default vs. original vs. last change
- drag and drop capabilities (dotted outline)

You can see here the now that Quick Fix version 2 has been applied, multiple exceptions have been caught in the catch blocks here. You can also see that there are dotted outlines around each exception. This means that you are able to drag and drop each into one of the various "drop zones" available. Drop zones look like this (*begin to drag an exception to show what a drop zone looks like*). You can see that you are able to re-organize the exceptions within the catch blocks as well as drag the exception to be thrown (*show how this is done*). Any item that contains this dotted border is drag-and-droppable into drop zones.

- movable popup box
- ways to exit/keep fix.

Before we begin, please be aware that once you complete a task, you will not be able to revisit that task.

Now, do you have any questions before completing these last tasks?