Senseo Experiment

Thank you for participating in this experiment!

Please answer the following questions about the JEdit application. In total you have 120 minutes to answer all questions. Please refrain from answering questions in a rush, rather skip the last question(s) if there is not enough time left.

For each question we will record the time it takes you to answer it and we will also rate the correctness of your answer. At max you can achieve four points for each question if your answer is fully correct. Please do not go back to a previous question which you have already answered to not perturb the recorded answering time for this previous question.

The questions are concerened with typical maintenance tasks you would perform in an application you have not developed yourself and about which you have no or only limited knowledge. You are not asked to actually perform these maintenance tasks, but to just acquire the necessary knowledge and insights into the system to be able to perform these tasks.

The system used in the experiment is JEdit, a fully-fledged text editor written in Java.

We ask you to work on five different tasks, each consists of two subtasks. These tasks are ordered in a top-down approach, which means that you first gather general knowledge about JEdit and later go down on a more detailled level to study the inner structure and behavior of JEdit.

Before beginning with the tasks, we provide you with a short written description and manual for the tools provided to you (Senseo and its features, tools). You can use this description as a reference during the experiment. No other tools are permitted during the experiment, in particular no manuals or other documentation about JEdit.

At the end of the experiment we kindly ask you to answer a short questionnaire asking you for your experience with the experiment and the tools used therein.

The experiment and the allocated 120 minutes start when you tell us that you are ready to start.

Task 1:

Question 1.1:

In JEdit you find the feature 'indent selected lines' in the menu at 'Edit -> Indent -> Indent selected lines'. Please study where and how this feature is implemented in the code on a coarse-grained level: Which packages, classes implement the feature? In which part or layer of the application is this feature mostly implemented? (UI, model, text processing, etc.)

Question 1.2:

Compare the package in which the 'indent selected lines' feature is mainly implemented with the package 'org.gjt.sp.jedi.gui' in terms of collaboration between them. Do these packages collaborate with each other and if so, which classes communicate?

Task 2:

We want to assess the quality of JEdit, for instance whether its classes are organized in the right packages or whether we should move classes to other packages. For this we need to analyze how certain classes are coupled to each other and how they communicate to the rest of the system. If a class of package C heavily communicates with classes of package B, it should maybe moved to package B. Or if a class is heavily used by package B (high fan-in from B) but has low fan-out to other packages, it should probably also belong to package B.

Fan-in of a class is defined as the number of methods or constructors of this class that are invoked from other classes. Fan-out of a class is the number of methods or constructors of other classes this class is invoking.

In the following we ask you to compare the quality of different artifacts of JEdit.

Question 2.1:

Compare the quality of class org.gjt.sp.jedit.buffer.FoldHandler, org.gjt.sp.jedit.msg.ViewUpdate and org.gjt.sp.jedit.Mode to each other in terms of fan-in, fan-out. Please report on fan-in and fan-out of for each of the three classes. Order the classes according to their quality, if we consider low fan-in and low fan-out as a sign for tight cohesion, that is, good inner quality.

Question 2.2:

Which classes of package org.gjt.sp.jedit.msg are heavily coupled to package org.gjt.sp.jedit? Are there classes in org.gjt.sp.jedit.msg heavily coupled to other packages than org.gjt.sp.jedit?

Task 3:

Question 3.1:

Analyze class org.gjt.sp.jedit.buffer.FoldHandler. To which objects (ie. instances of which classes) does FoldHandler send messages in which order when folding a paragraph in the text editor?

Question 3.2:

What kind of client objects communicate to instances of FoldHandler, that is, invoke methods of FoldHandler? Also consider subclasses of FoldHandler. How does the communication go from the FoldHandler to subclasses implementing particular folding features?

Task 4:

We look again at the 'indent selected lines' (Edit -> Indent -> Indent selected lines) feature. This feature works on selected lines of text, but after executing (eg. indenting two lines of text), the selection is often lost, no text is selected anymore sometimes. We want to correct this problem by keeping the selection after having indented a text.

Question 4.1:

Compare this feature to the implementation of other related features such as 'remove trailing whitespaces' or 'shift indent left'. What is implemented differently in these features than in 'indent selected lines'? Consider particularly the method that executes the rules for locating the desired piece of text in the editor.

Question 4.2:

We really want 'indent selected lines' to keep the selected text. How can you achieve this? What will you change in the code (which methods to adapt, what statements to add, change or remove)?

Task 5:

Feature 'remove whitespace' (Edit -> Indent -> Remove trailing whitespaces) has been identified by users to be very slow; the lead developer of JEdit thinks class org.gjt.sp.jedit.indent.WhitespaceRule is the bottleneck. Can you confirm this assumption? Answer the following two questions to decide whether we should optimize this class.

Question 5.1:

What are the methods and the control structures in class WhitespaceRule that get exectuted the most? Are there repeated patterns of code that get executed when this rule is applied to a document?

Question 5.2:

Compare class WhitespaceRule to other classes in the same hierarchy of Rules (classes implementing IndentRule). Is it really used more often or more heavily (eg. does it create more objects or execute more complex code) than the other classes? Explain why or why not.