Introduction

Information about software architecture concerns the important elements of a design, the relationships between them, and important designs made about them. While much work has been done to help architects document, analyze, or reverse engineer architecture, less is known about the use of architectural information during coding tasks. While it seems such information should be helpful, which information is most helpful, how it should be presented, and exactly how it would help remain less clear. To understand the requirements for future diagram or visualization systems, specifications, or coding conventions to better present architectural information during coding tasks, we are studying how developers currently interact with architectural information. We hope that a careful analysis of the questions developers ask, the information they seek, and the difficulties they experience will lead to the design of more useful future systems for presenting architectural information.

You will be asked to perform two change tasks. You will have 1.5 hours (beginning after you finish reading the actual task description) to work on each task. The tasks have been designed to be challenging, so you will likely not have time to investigate or design as much as you could with unlimited time constraints. Your goal is to complete each task with a well designed implementation that respects the existing architecture of the system as much as possible. You will be investigating and trying to change two complicated pieces of code in a codebase that you will have little time to explore. So do not feel discouraged if you are not able to accomplish as much as you might hope you could accomplish.

You will be provided with a copy of Eclipse 3.2. You may use any feature of Eclipse – including running the program – and may use Explorer to open any files created by jEdit – such as log files or files edited by jEdit – in Notepad or jEdit. But you may not use any other application (including a web browser). At the end of the first task, you will be given a fresh copy of the source to work on the second task. After each task, the experimenter will ask some questions about your understanding of the system, how you were working, and decisions you made in creating your implementation.

To understand how you are working, you will be asked to "think aloud" – talk about what you are thinking while you work. You should describe what you're trying to accomplish and what you're considering doing to accomplish your goals. If you find information that you've been seeking for a while, you should make sure that you say something about what you found. If you are distracted of forget to think aloud, the experimenter will prompt you to continue to think aloud by asking you about what you're doing. To allow us to analyze how you work, you will be recorded using a laptop audio recorder, a screen capture program, an Eclipse event logger, and two video cameras. Your identity will not be revealed to anyone other than researchers in the research group. To ensure anonymity, only the researchers in the research group will have access to the audio and video recordings. However, transcripts of the audio recording with any personally identifying information removed and the screen recording may be used in public presentation of this work.

Eclipse Code Navigation Tutorial

Eclipse features a number of sophisticated features for navigating through source code. In this brief tutorial, you will try out several of the most useful features.

1. Press CONTROL-SHIFT-T and type “jedit” to open up the jedit class.
2. Press CONTROL-O to open the method outline and type getProperties to navigate to the method getProperties().
3. Hold down control and click on getProperties() (the method call on propMgr, not the method declaration). This moves you to the method declaration.
4. Go back to the previous method you were looking at by clicking on the left arrow secondmost from the right edge of the toolbar.
5. Place the cursor over the getProperties() method declaration, right click, and select “Open Call Hierarchy”. At the bottom of the screen, a tree view shows either the Callee Hierarchy or the Caller Hierarchy. Switch between them by clicking a button to the right of the Call Hiearchy tab.
6. Place the cursor on propMgr. All references to this field in the current Java file are now highlighted with a grey highlight both in the text editor and next to the scrollbar, allowing you to quickly find all of the references in the file.
7. Find all of the methods that assign the field propMgr by placing the cursor on top of it and selecting from the “Search” menu at the top of the screen “Write Access” and then “Project”.
8. Find all references to the jEdit type by selecting Java from the Search menus, typing in jEdit, and selecting “Type” in Search For.

jEdit Background
For your two tasks, you will be working with jEdit. jEdit is an open source text editor mainly intended for editing code written in a little over 50,000 LOC of Java. It was originally written by mostly one developer (Slava Pestov) but has more recently been edited by several other developers. Start jEdit by clicking the debug button the on the Eclipse toolbar (the fourth icon from the left).

Several important classes in jEdit can be considered components:

jEdit
jEdit is the main class in the application and contains the application entry point. It is mostly responsible for orchestrating interactions between other components such as opening and closing buffers, creating and switching views, managing plugins, and setting and retrieving global application properties.

View
The top level window for a jEdit application toolbars, user input, creating and deleting EditPanes, and visual look and feel. There can be more than one open view for a particular application – to create a new View simply select View.New View on the menu bar.

EditPane
The containing visual element for a JEditTextArea text editing control. There may be more than one when a view is split into multiple panes. EditPanes are responsible for switching buffers and managing markers (bookmarks).

JEditTextArea
jEdit's text control for editing text. Responsibilities include scrolling, selection, painting, buffer access, caret position, text input, and text transformations.

JEditBuffer
jEdit's representation of a file containing functionality for manipulating the text in a buffer.

Buffer (inherits from JEditBuffer)
jEdit's representation of a file containing rules about how the text file in memory is loaded and stored to disk.

EditBus
Registered components receive messages reflecting changes in the application's state, including changes in buffers, views and edit panes, changes in the set of properties maintained by the application, and the closing of the application.

StatusBar
StatusBars are a small message bar at the bottom of a View that convey information about the state of the application such as the scroll position and caret position.

The next two pages give a component and connector diagram of these components split into two diagrams based on the type of communication occurring.

Try out jEdit by performing the following actions:
-Create a new file
-Type some text in the file and note that the file's icon changes.
-Press return enough times so that there is more text than can fit on the screen. Scroll the active window. Note that the left portion of the StatusBar displays two pieces of information. The leftmost piece is the line and column of the caret (cursor's) position. Next to this is a percentage describing how far down the text are is scrolled. Other messages (such as a buffer being autosaved) are displayed in the center portion of the status bar. On the right edge is information on the current character set and mode and current memory use.
-Create a second new file. Use the list box with the file's name above the text area to switch back to the first file.
-JEdit has a feature called "folds" which hierarchically hides lines of text. Type in the following text using tabs for spacing:

```
The
	Quick
		Brown
			Fox
	Jumped
		Over
	The
		Lazy
			Dog.
```

Note that JEdit displays triangles next to each of these lines. Clicking on the triangles causes JEdit to hide or show the lines of lower fold level than the line selected.

Task 2 – Fold Level Updates
Consider the following code fragment:

BufferHandler.doDelayedUpdate() : BufferHandler.java:363

```
            // force the fold levels to be
            // updated.

            // when painting the last line of
            // a buffer, Buffer.isFoldStart()
            // doesn't call getFoldLevel(),
            // hence the foldLevelChanged()
            // event might not be sent for the
            // previous line.

            buffer.getFoldLevel(delayedUpdateEnd);
```

BufferHandler is a subcomponent of JEditTextArea responsible for responding to events on the BufferListener event bus provided by Buffers. It does this by implementing the BufferListener interface and being subscribed to the bus. Unfortunately, getFoldLevel() is both architecturally questionable and clearly bad design. Architecturally, it is intended to change the buffer's state from within a different component (JEditTextArea), which may not be architecturally ideal. At a design level, it is changing the state by calling a getter method, which most developers might reasonably assume to have no effects (does not change any of the object's fields). Indeed, it is using a getter method solely to change the state of the buffer and ignoring the information the getter method is supposed to be used to obtain. This is clearly poor design.

Your task is to investigate why this is the case and implement a better design. You should carefully budget your time to make your improved design as ideal as possible while carefully scoping your changes to what you can implement within your allotted time. As you work, you should produce a design diagram or diagrams illustrating your new design. You may use any notation you wish (e.g. UML class diagrams), including your own, but are encouraged to use a notation that captures as much of your design as possible. You may change as much or as little code as you'd like. Your goal is only to make the design as ideal as possible by the criteria of performance, understandability, and reusability. Once you've completed your task, the experimenter will ask you several questions about your design and design process including what alternatives you considered and what criteria you used to select a design alternative. You may use the file "Task 2 Notes.txt" for any notes you wish to record, and the piece of paper on the desk for any diagrams or notes you do not wish to type. You have 1.5 hours beginning now to work.

Task 1 – StatusBar caret updates

Start jEdit from the debug command, type a couple of characters of text, and create a second buffer. Place a breakpoint at the beginning of StatusBar.updateCaretStatus() at StatusBar.java:368 and switch buffers. updateCaretStatus() should be being called a number of times, resulting in the breakpoint also being hit a number of times. This is bad, at the very least from a performance perspective, because the status bar's caret message should only be changing once – from the value for the old buffer to the value for the new buffer. But it likely reflects deeper problems in the semantics of what the events that trigger these updates mean.

Your task is to investigate why this is the case and implement a better design. You should carefully budget your time to make your improved design as ideal as possible while carefully scoping your changes to what you can implement within your allotted time. As you work, you should produce a design diagram or diagrams illustrating your new design. You may use any notation you wish (e.g. UML class diagrams), including your own, but are encouraged to use a notation that captures as much of your design as possible. You may change as much or as little code as you'd like. Your goal is only to make the design as ideal as possible by the criteria of performance, understandability, and reusability. Once you've completed your task, the experimenter will ask you several questions about your design and design process including what alternatives you considered and what criteria you used to select a design alternative. You may use the file "Task 1 Notes.txt" for any notes you wish to record, and the piece of paper on the desk for any diagrams or notes you do not wish to type. You have 1.5 hours beginning now to work.