Exploring How Developers Layout UML Class Diagrams

Bonita Sharif
School of Computing
University of Nebraska - Lincoln
Lincoln, Nebraska USA
Email: bsharif@unl.edu

Nathaniel Liess
Department of Electrical Engineering
University of Nebraska - Lincoln
Lincoln, Nebraska USA
Email: nliess2@huskers.unl.edu

Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent, Ohio, USA
Email: jmaletic@kent.edu

***Abstract*—The paper presents a video-based exploratory study that seeks to understand how developers modify UML class diagram layouts for better readability and comprehension of the system. Two diagram layouts showing a model subset from a Java open-source system are presented to six participants experienced in reading UML class diagrams. They are tasked to change the layout to make it easier for them to read and comprehend. The video is reviewed for major modifications to the layouts. Behaviors observed are presented. The eventual goal is to use this information to construct heuristics for automated layout algorithms based on semantics and architectural importance.**

***Index Terms*—UML class diagrams, layout, exploratory study**

I. Introduction

Several researchers have examined the problem of constructing different layouts for UML [1] class diagrams. These layouts mainly prioritize general (graph-based) aesthetics to produce less clutter. For example, they tend to minimize edge crossings, use 90-degree bends for edges, and use hyper-edges for the hierarchies. These layouts are generally better than those generated from state-of-the-art tools for UML modeling that typically do not prioritize a set of aesthetic criteria.

The orthogonal layout [2] for UML class diagrams is popular as it produces an aesthetically pleasing diagram via the minimizing of edge crossings, bends, and lengths while also maximizing symmetry. The layout scheme does not use any semantics from the UML model itself to drive the layout. To bridge this gap, we presented a pilot study to test if class stereotypes [3] of entity, boundary, and control classes, can help improve the layout of diagrams and found that the clustered layouts performed better in comprehension tasks. In particular, the multi-cluster layout, where each cluster represents a concept in the system, is compared to the orthogonal and three-cluster layout, where each class stereotype is placed in one of three clusters of entity, boundary, and control. We presented several other studies [4] after our initial pilot that continue to show that clustered layouts perform better than orthogonal ones for different tasks.

The above studies are tested on existing layouts produced by researchers. We are not aware of any studies that explore how a developer modifies an existing UML class diagram layout. To bridge this gap, we conduct an exploratory study to understand strategies developers use to modify a UML class diagram layout with the goal of improving comprehension. Our research question is: *What strategies do developers use to modify layouts of UML class diagrams to better comprehend the software system?* A small pilot study observing six developers produces some guidelines observed. The eventual goal is to conduct a larger study to articulate behaviors that can be built into a layout modification algorithm in existing tools.

II. Study Overview

The goal of this study is to understand how software engineers modify an existing UML class diagram layout. The diagram is shown in a random layout generated by the default setting in the modeling tool. They are expected to produce a more readable and comprehensible layout.

The participants are six graduate students and faculty recruited from an advanced software engineering course at Kent State University and the University of Akron in Ohio, USA. The participants volunteered for the study and use UML for design in graduate classes and research during their Ph.D. program.

The study consists of two layout modification tasks based on the JEdit open-source Java system. The system is first reverse engineered to produce a UML model. A subset of the model is then imported into MS Visio and the default orthogonal layout is used to generate a starting layout for participants to begin modifying. See Figure 1 for the two initial layouts in the JEdit system. There is relatively little overlap between the two sub models to avoid learning effects or biases. The diagrams have class stereotype information via textual annotations above the class name and via color (boundary classes are green, control classes are red, and entity classes are blue).

The study was approved by the Institutional Review Board at Kent State University. The participants are seated in front of a monitor where the diagrams are presented in MS Visio. Their screen is recorded via an external camcorder. Audio is also recorded as they are encouraged to think aloud. They are prompted to modify the layout to make it more readable and comprehensible to them based on how they understood the model. They are asked to move and rearrange the diagram as they liked. They are asked not to delete any classes or relationships. The study took approximately 45 minutes on average for both tasks. The videos are then reviewed for

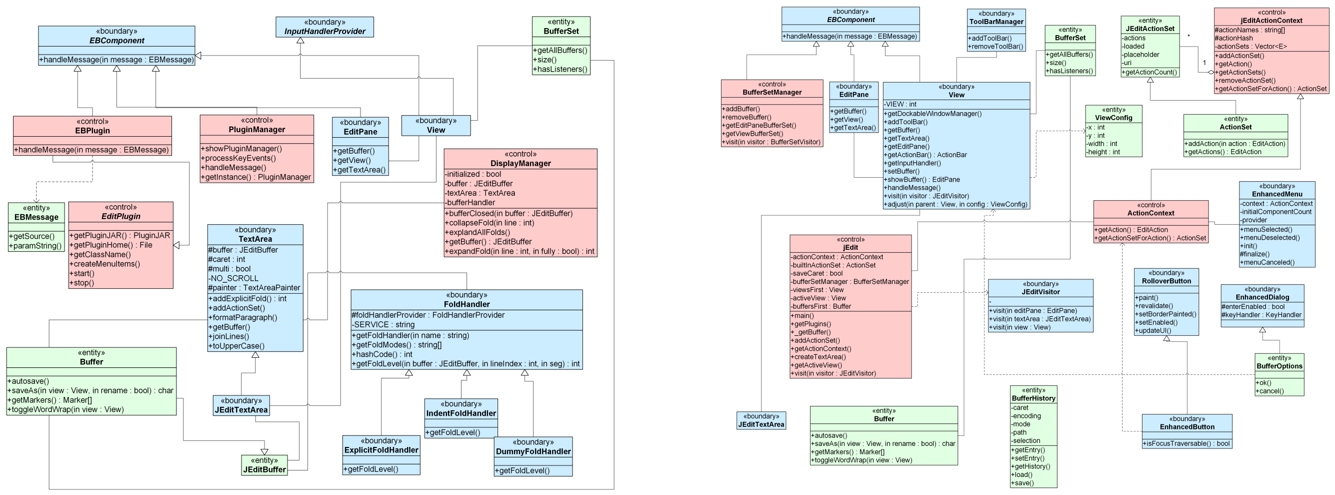

Fig. 1. The JEdit UML class diagrams presented in random layout for Task 1 (left) and Task 2 (right).

major modifications made by the participants to the diagrams to determine any trends in behavior. The main behaviors of interest are moving a class, changing the shape of a relationship, joining of inheritance arrows, making inheritance read in the typical top down way, moving related classes closer, to name a few.

III. Common Observations

After reviewing the videos, we found some common behaviors in each of the two tasks.

For the first task, the control class `DisplayManager` is moved around at different locations in the layout. Participants also add extra space around the `FoldHandler` boundary class. In the second task, participants focused on making the lines straight instead of bent. The class `JEditTextArea` is moved closer to the class `View`. Participants also moved pairs of classes close to each other - in particular `Buffer` and `BufferSet`. A few participants switch the order of the `Enhanced Button` class and the `RolloverButton` class. Finally, we noticed that the `JEdit` class is moved to the right to make space for the `Buffer` and `BufferSet` classes.

Finally, even though we ask participants to think out loud while they are performing the layout modification, they very rarely spoke and verbalized their thoughts.

IV. Discussion and Conclusions

This exploratory study gave us some indication on how developers go about modifying a layout. First, it signaled to us that the task of arranging a layout is inherently difficult for most people. Many of the developers just read the diagram for a while before making any modifications. This indicates to us that we should probably attach a comprehension task to each diagram instead of asking the developer to just rearrange it for overall readability and comprehension.

We did not uncover many insights on specific patterns but point out the major changes for each layout. In both tasks, the control classes of `DisplayManager` and `JEdit` are at least touched and moved. This indicates to us that the participants recognized these are important classes that contain a lot of the functionality. Classes that have associations but that are placed farther in the initial layout are brought closer together during the modification process indicating that if classes have associations, they should have a priority to be placed closer to each other. The state-of-the-art tools do not always prioritize this closer placement.

In hindsight, providing a specific task of comprehension or use of the diagram to solve a specific problem might have been a better design decision instead of a general comprehension and readability task. This is our hypotheses that can be tested in a followup study. If this indeed is the case, then it indicates that the layout should change based on the type of task being done. A task-dependent layout for class diagrams might make more sense where the developer is able to switch seamlessly between different layouts that are more conducive to certain tasks.

Acknowledgment

This work is supported in part by the US National Science Foundation under Grant Number CCF 18-55756.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*, 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2005.

[2] M. Eiglsperger, C. Gutwenger, M. Kaufmann, J. Kupke, M. Jünger, S. Leipert, K. Klein, P. Mutzel, and M. Siebenhaller, "Automatic layout of uml class diagrams in orthogonal style," *Information Visualization*, vol. 3, no. 3, pp. 189–208, 2004. [Online]. Available: https://doi.org/10.1057/palgrave.ivs.9500078

[3] O. Andriyevska, N. Dragan, B. Simoes, and J. Maletic, "Evaluating uml class diagram layout based on architectural importance," in *3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2005, pp. 1–6.

[4] B. Sharif, "Empirical assessment of UML class diagram layouts based on architectural importance," in *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*. IEEE Computer Society, 2011, pp. 544–549. [Online]. Available: https://doi.org/10.1109/ICSM.2011.6080828