Software Maintenance Autumn 2023

Frederik Busch

December 7, 2023

Student Mail: frbus21@student.sdu.dk
Date: mm/dd/yyyy

Contents

1 Change Request **3**

2 Concept Location **5**
2.1 Methodology 5
2.2 Table Content Overview 5

3 Impact Analysis **7**
3.1 Brief Introduction 7
3.2 Featureous Feature-code Characterization 8
3.3 Featureous Feature Relations Characterization 9
3.4 Feature-code correlation graph and feature-code correlation grid 10
3.5 Table - Impact Analysis 12

4 Refactoring Patterns and Code smells **13**

5 Refactoring Implementation **14**

6 Verification **15**

7 Continuous Integration **16**

8 Conclusion **17**

9 Source Code **18**

1 Change Request

For this Software Maintenance report document, i have chosen to work with the feature called *Tool Palette.* The refactoring of the code will be done in a group consisting of 5 students total, including myself. We have each choosen af feacture to reactore doing the course of this project.

The infomation we have gotten on the different features are only a short descriptive text, with the name of the feature. For my chosen feature the text is the following: *Tool Palette - Display, Drag and Drop.* With this feature name i can with some analysis and implementation of the given code I can figure out what i have to reactore within my feature. As I am working with the feacture called *Tool Palette,* I will assume the whole of the tool palette is within my feacture. The *Tool Palette* after inspection looks to contain tool sections, where it has different tools that one can select within these sections.

As part of the project we have made individual User Stories for our chosen feature. My User Story are the following:

Drag and Drop

The *Drag/Drop* user story outlines a feature, that is designed to enable users of the program to customize their workspace within the program itself. It allows the user to drag and drop different sections of the toolbar, to a location of their choosing. By allowing the user to customize their workspace, it can impove their work efficiency, but have their most used tools and options within easy reach.

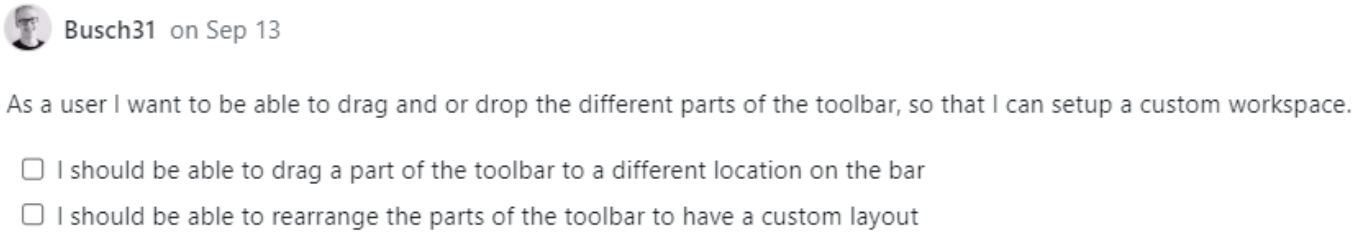

Figure 1: User Story for Drag and Drop

Display

The *Display* user story outlines a feature, that is designed to enable the users to show or hide different sections of the toolbar to their liking. Thereby allowing the users to hide or show only the tool section, that are relevant to their current task. It will also give the user less clutter on their screen doing their work.

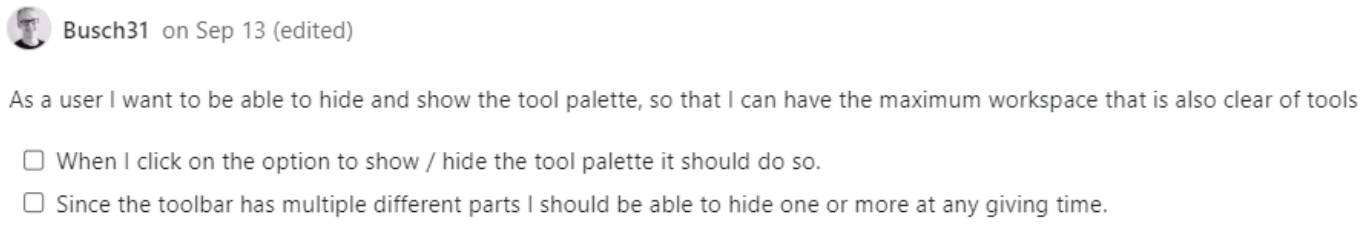

Figure 2: User Story for Display

To successfully complete the refactoring, the following steps should be undertaken by us as a group:

- Learn the feature scope of our different features within the codebase by doing a concept location to identify the relevant classes and tools.
- Evaluate the estimated impact of the refactoring on each developers features to anticipate any potential overlaps or conflicts our different features might have or could have.
- Understand the sections of code that require refactoring by identifying it with code smells.
- Carry out the refactoring while trying to minimize any unintended cascading changes that could happen with refactoring.
- Verify the changes after refactoring to ensure they achieve the desired outcome and that the primary function of the code is still maintained.

Besides having to do this refactoring, we as a group also have to setup continuous integration, thereby ensuring that any code is tested and verified before it is merged into the main branch.

2 Concept Location

2.1 Methodology

The location of the classes that I identified was based on the following tools, which I used to locate the different classes and the different methods that I found was relevant for the feature I had chosen *Tools Palette*.

- Different search methods such as:
  - Find all references.
  - Go to definition.
  - Quick search.
  - Global search.
  - Keyword search
- Tree scaling both up and down with Extension reference.
- Removing code to see what functionality it would affect, thereby better understand what the different pieces of code did what and affects.

2.2 Table Content Overview

The table below provides an easy overview of the different tools and processes I used to locate the different classes that I found relevant for my chosen feature.

#	**Domain classes**	**Tools used**	**Comments**
1	*AbstractToolbar*	Quick Search Find all references Code removal	I started by looking at the different abstract classes for the whole project. Here I found the *AbstractToolbar* class which looked like the right abstract class I was looking for when my features name is *Tool Palette.* I then tested with *code removal* to see what it would impact in the toolbar, but I just not see any changes to the behavior of the program itself when running, so I started looking at what the *AbstractToolbar* was extended from.
2	*JDisclosureToolbar*	Code removal Extension reference Go to definition	When I look at what *AbstractToolbar* was extended from, I found the abstract class named *JDisclosureToolbar*, I again tried code removal, this time giving my first result. The abstract class *JDisclosureToolbar* is responsible for the *show/hide* feature of the *tool palette,* which I needed for my user story *Display.*
3	*ToolsToolbar*	Code removal Extension reference	I then went back down the reference tree to see where in what class it would end. I ended up in the class *ToolsToolbar*. Here again with *code removal* I tried to see what the class was responsible for. I found that it was not the full toolbar as my first thought had been, but it was only a part of the whole *tool palette.*
4	*PaletteToolbarUI*	Code removal Keyword Search	After I hit a dead end with the *Extension reference* tool method, I tried to do a *global search* on different keywords such as *Tool, UI, Palette, Bar, ToolBar* and other keywords that could be assimilated with my feature. With this tool method I found the *PaletteToolbarUI* class which contained handlers. I tried to remove some of these handlers to see what it would affect.

Table 1: Overview of Domain Classes and Tools Used

3 Impact Analysis

3.1 Brief Introduction

The impact analysis is used to understand the implications of changing a specific feature within the *JHotDraw* project. The project itself will receive the different feature changes at different times, as the development team, consisting of 5 members, they are all working on different features of the application at their own pace, and some members might be further ahead than others at any given time. After inserting the feature entry points into the *JHotDraw* project, I was able to get an output of different relevant figures: *Feature-code Characterization, Feature-code Correlation Grid,* and *Feature-Package Correlation Graph.*

The feature entry points I used in this project are the following:

- *Tools-display*
- *Drag-drop*
  - *Pressed*
  - *Dragged*
  - *Released*

The Tools-display, as stated in the Concept Location chapter, is a class that references the *JDisclosureToolbar* class. The *Tools-display* is a handler callback that has been added to a button; when pressed, it will change the visibility of the chosen toolbar section from visible to hidden or vice versa.

The *Drag-drop*, which consists of *Pressed*, *Dragged*, and *Released*, are handlers that are connected with the *PaletteToolbarUI*. They are activated in the following order:

- *Pressed*: when a tool is pressed on with the click of a mouse.
- *Dragged*: when a tool has been pressed and the mouse moves while the button is still being held down by the user.
- *Released*: when the user releases the pressed mouse button again.

3.2 Featureous Feature-code Characterization

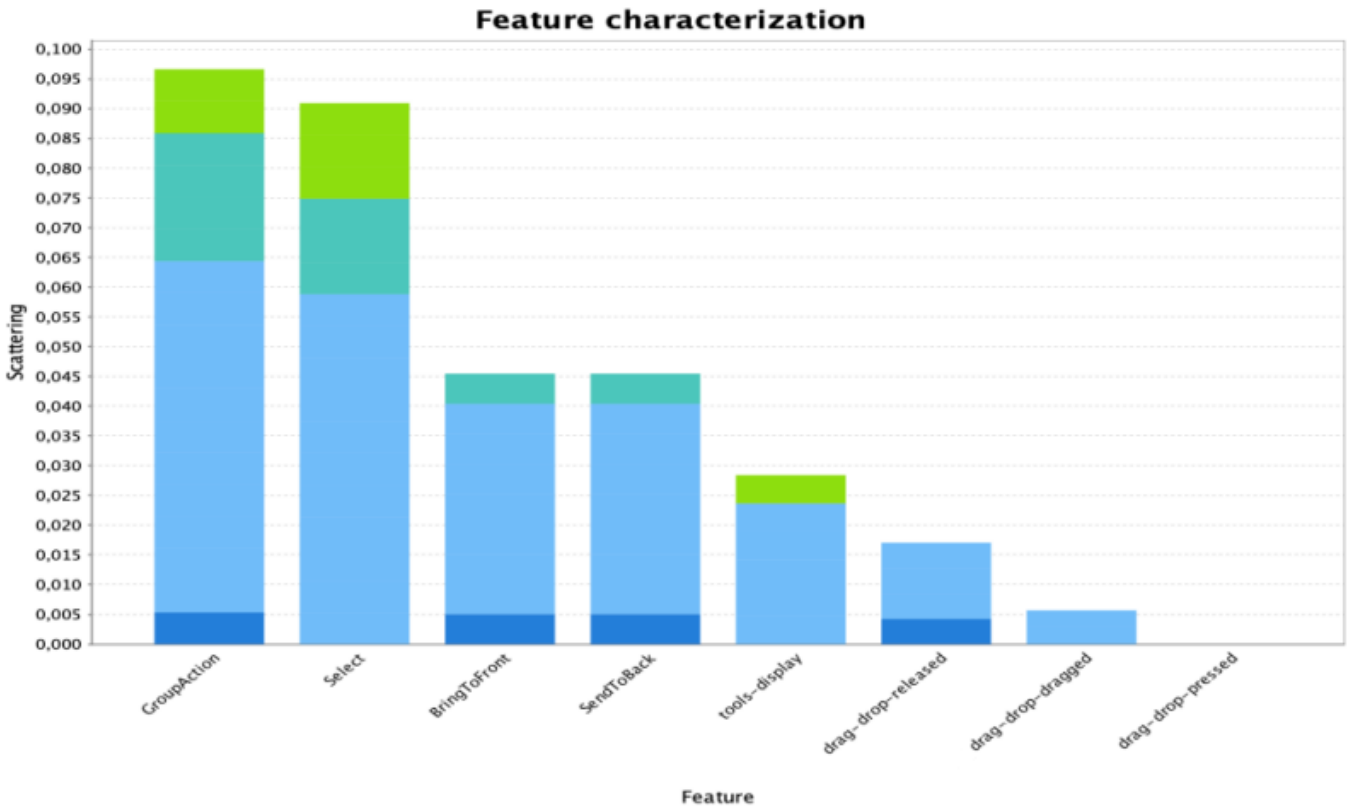

Figure 3: Featureous Feature-Code Characterization

As we can see, the units do contain alot of inter-group units, this can if one is not careful, end in entanglement with the other units that other developers are using in their part of the project, but it does offer us some insights for future refactorings.

3.3 Featureous Feature Relations Characterization

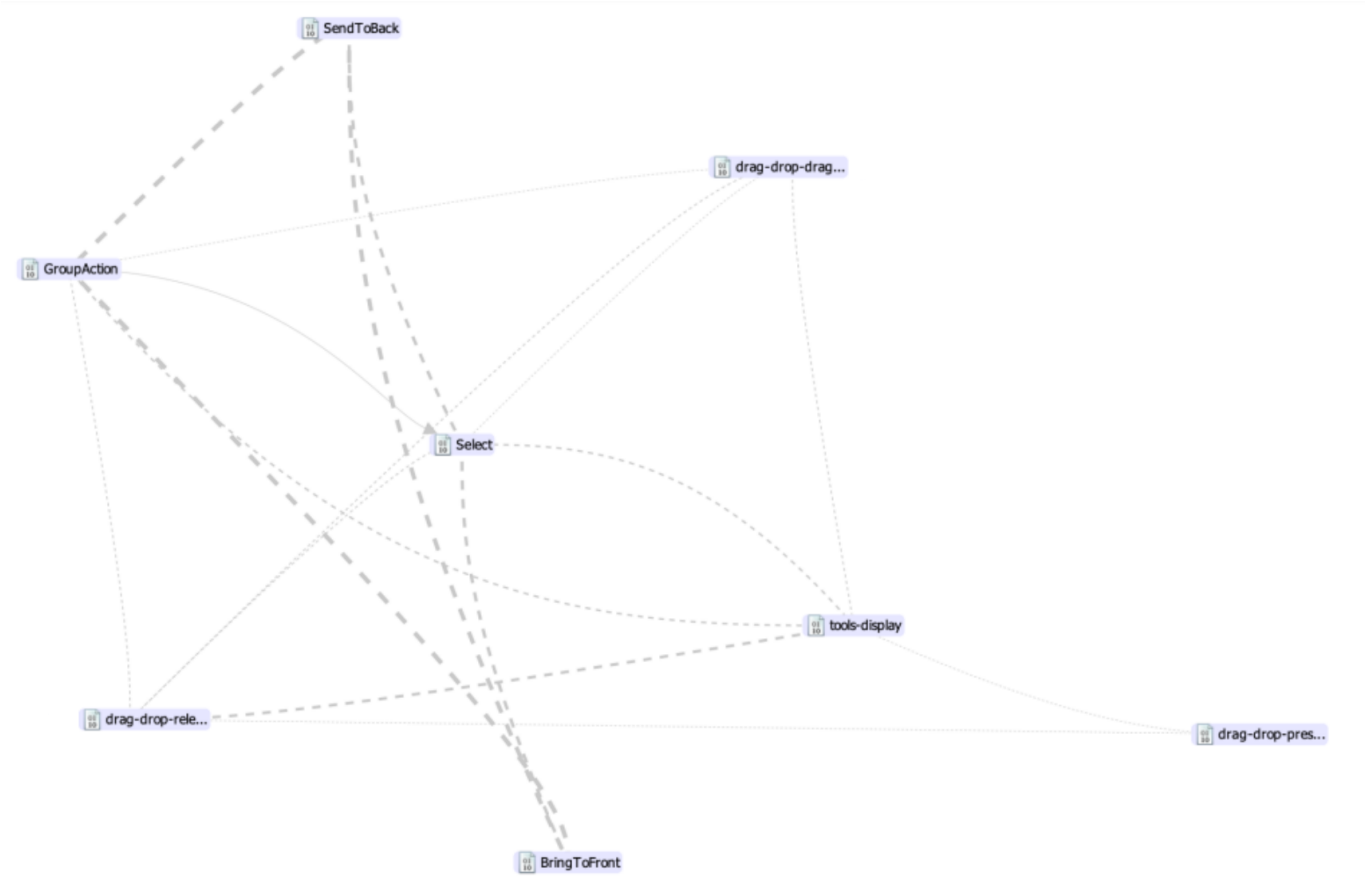

Figure 4: Featureous Feature Relations Characterization

As one can see, the connections which the entry points have made, does not contain strong connections with the other features that are also in this project. it does not look like they even engage in any consumer/producer connections.

3.4 Feature-code correlation graph and feature-code correlation grid

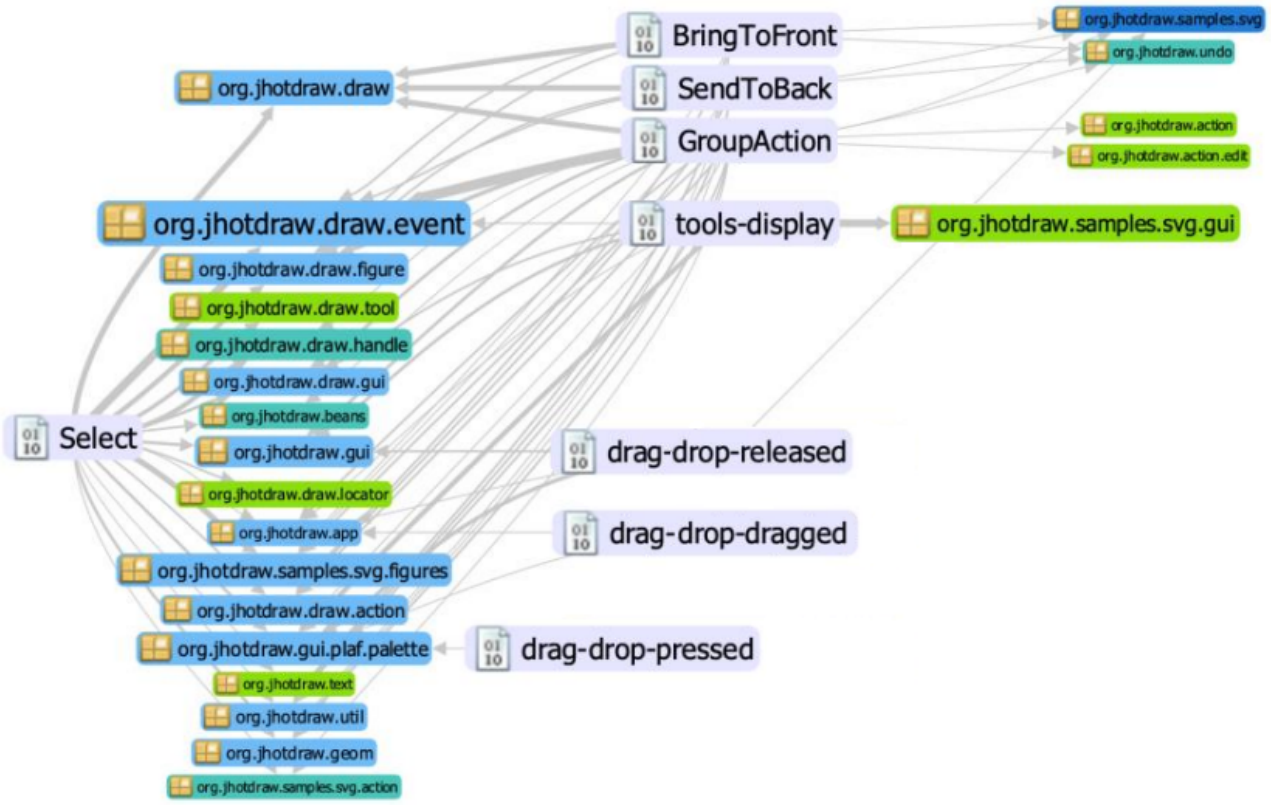

Figure 5: Feature-Package Correlation Graph

These tools can provide a deeper look into the connections between source code units and features, highlighting relationships and dependencies important for understanding the overall impact of the change request.

But however the *Feature-Package Correlation Graph* does not give the best overview of the connections between the features and the packages. What it can provide however, is a visual confirmation that there is no connection between the *PaletteToolbarUI* and the other entry points in the project.

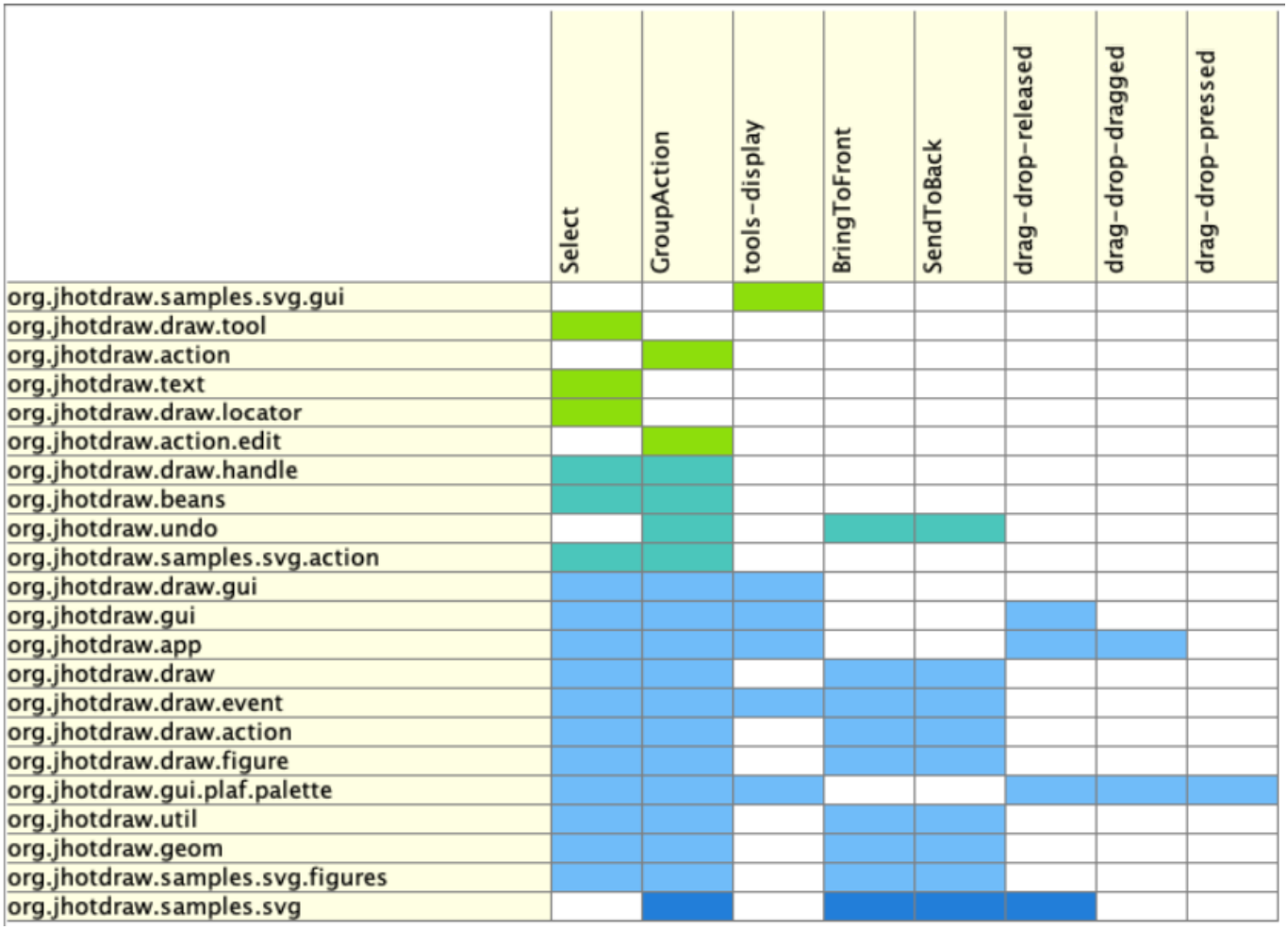

Figure 6: Feature-Code Correlation Grid

The analysis reveals only a small entanglement between my own features and those being used by other developers in the project.

The *drag-drop-released* feature shares the core package *org.jhotdraw.samples.svg* with *GroupAction*, *BringToFront*, and *SendToBack*. Also, all three functions of *PaletteToolbarUI* cross with *Select* and *GroupAction*. The *tools-display* feature also cross with several intergroup packages, but heavily with *Select*, *GroupAction*, and one package each from *BringToFront* and *SendToBack*. This interlinked nature of features requires very careful mindsets of other developers' work before proceeding with development and refactoring of their parts of the project.

3.5 Table - Impact Analysis

Package name	**# of classes**	**Tool used**	**Comments**
.gui	76	Correlation grid	changed
.gui.plaf.pallete	38	Correlation grid	changed
.samples.svg	65	Correlation grid	unchanged
.draw.event	20	Correlation grid	unchanged
.app	39	Correlation grid	unchanged
.draw.gui	6	Correlation grid	unchanged

Table 2: Feature-code correlation data

The *PaletteToolbarUI* class displays a high level of independence, with no connections into *JHotDraw* outside of the *gui.plaf.palette* package. Its use is restricted to *PaletteToolbarBorder* within the same package and *JDisclosureToolbar*, related to another primary feature. Also the *JDisclosureToolbar* is connected only with the *gui.plaf.palette* package. given that no changes are made to its public methods, any refactoring is not likely to disturb other parts of the project.

4 Refactoring Patterns and Code smells

hej

5 Refactoring Implementation

hej

6 Verification

hej

7 Continuous Integration

hej

8 Conclusion

hej

9 Source Code

hej