

Software Design 2DV608

Assignment 3

Retake 1

Author: Nima Safavi (ns222tv@student.lnu.se)

Teacher: Mauro Caporuscio

Introduction

This assignment is focusing on the architectural design. The target to mitigate the discovered issues in classes, methods, architecture, and re-design the implementation. This is done by applying the Design Principles and architectural pattern.

Task 1 – Code analysis by Sonargraph

According to the instructions for this assignment the following process has been done for the analysis task:

- The code was imported in the Sonargraph Explorer application.
- The code analysis was done by Java quality model provided by the application.

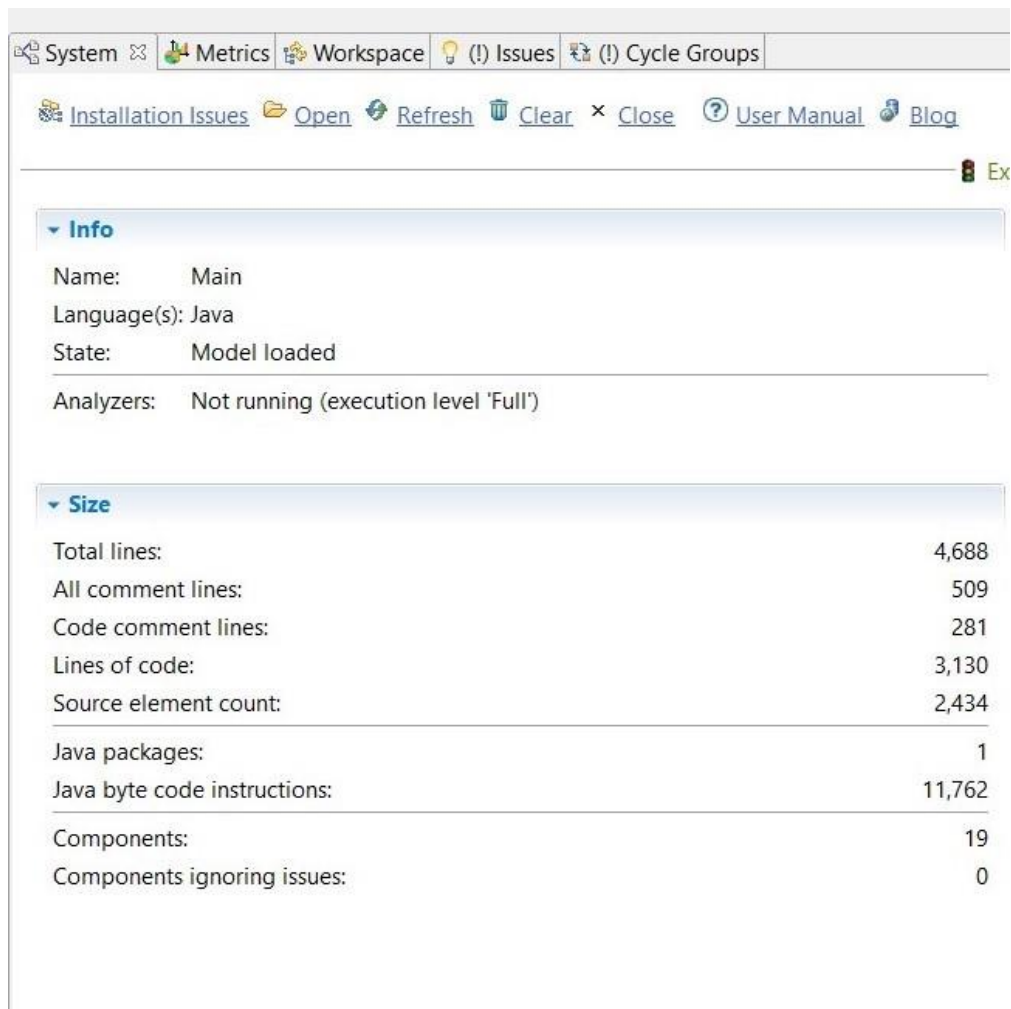


Figure 1

Figure 1 depicts the summarize the result of code analysis for the BrickMosaic project. As shown the project language is in java and there are 4688 lines of code including 509 lines for all comments and 281 lines for code comments (documentation comment) and 3130 lines of actual code. This project owns only one package with 19 classes (components).

lorer

▼ Issues	
Total:	8
Configuration:	0
Workspace:	0
Threshold violations:	7
Cycle groups:	1
▼ Structure	
System Maintainability Level:	90.33
System ACD:	5.63
Highest module ACD:	5.63
Cyclic Java packages:	0
Structural debt index (Java packages):	0
Component dependencies to remove (Java packages):	0
Parser dependencies to remove (Java packages):	0
Cyclic components:	4
Structural debt index (components):	46
Component dependencies to remove (components):	3
Parser dependencies to remove (components):	16

Figure 2

Figure 2 illustrate other information regarding general system analysis. The important point to mention is the **Issues** section, 8 issues were identified including 7 for threshold violation and 1 for component cycle group which we will discuss and resolve later in this report. Next, below that in the **structure** section, we can observe that the system maintainability is 90.33 which is quite high. The average component dependency is 5.63 which means each class is depended on the average to 5.63 of the other classes. The number of cyclic java packages is 0, as a result the

number of structural debt index and component are 0 in this case. There are 4 cyclic components which leads to 46 structural debt indices. Also, 3 of 4 cyclic component dependency should be removed. The number of parser dependencies to be removed is 16.

To increase the usability, I must increase cohesion, abstraction and reduce coupling.

Issues

The figure 2 shows information for the detected issues in the project.

Element	Affected Eleme...	Error	War...	Info
Installation	1	1	0	0
Main	9	0	8	0
Workspace	9	0	8	0
Project	9	0	8	0

Issue [9]	Description	Severity	Category	Element	To Element
Component Cycle Group	Java Module 'Project' contains 4 cyclic components	Warni...	Cycle Group	Component cycle group 1.1	n/a
Threshold Violation	Number of Statements = 115 (allowed range: 0 to 100)	Warni...	Threshold Violati...	createDialog() : void	n/a
Threshold Violation	Number of Statements = 174 (allowed range: 0 to 100)	Warni...	Threshold Violati...	createDialog() : void	n/a
Threshold Violation	Number of Statements = 120 (allowed range: 0 to 100)	Warni...	Threshold Violati...	actionPerformed(ActionEvent) ...	n/a
Threshold Violation	Modified Cyclomatic Complexity = 24 (allowed range: 0 to 15)	Warni...	Threshold Violati...	actionPerformed(ActionEvent) ...	n/a
Threshold Violation	Number of Statements = 192 (allowed range: 0 to 100)	Warni...	Threshold Violati...	initialize() : void	n/a
Threshold Violation	Number of Statements = 227 (allowed range: 0 to 100)	Warni...	Threshold Violati...	dolnBackground() : Integer	n/a
Threshold Violation	Modified Cyclomatic Complexity = 20 (allowed range: 0 to 15)	Warni...	Threshold Violati...	dolnBackground() : Integer	n/a
Python Interpreter Not Fo...	Python interpreter could not be determined from environment...	Error	Installation Confi...	Installation configuration for P...	n/a

Figure 3

As mentioned, there are 8 issues in the project but here there is one added error due to installation of Python (which is not related to our task and will not be solved in this report). We can see that there are 7 threshold violations with the warning severity and 1 for cycle group component (with 4 cyclic elements) again with warning severity.

The picture below demonstrates the exploration view for the source code provided.

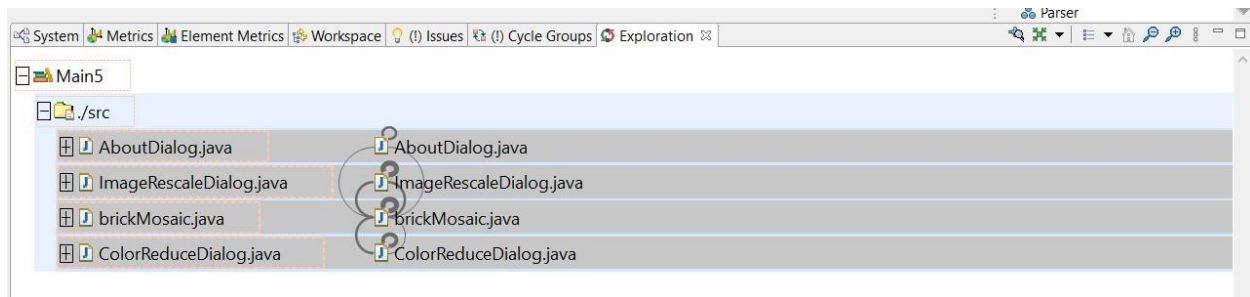


Figure 4

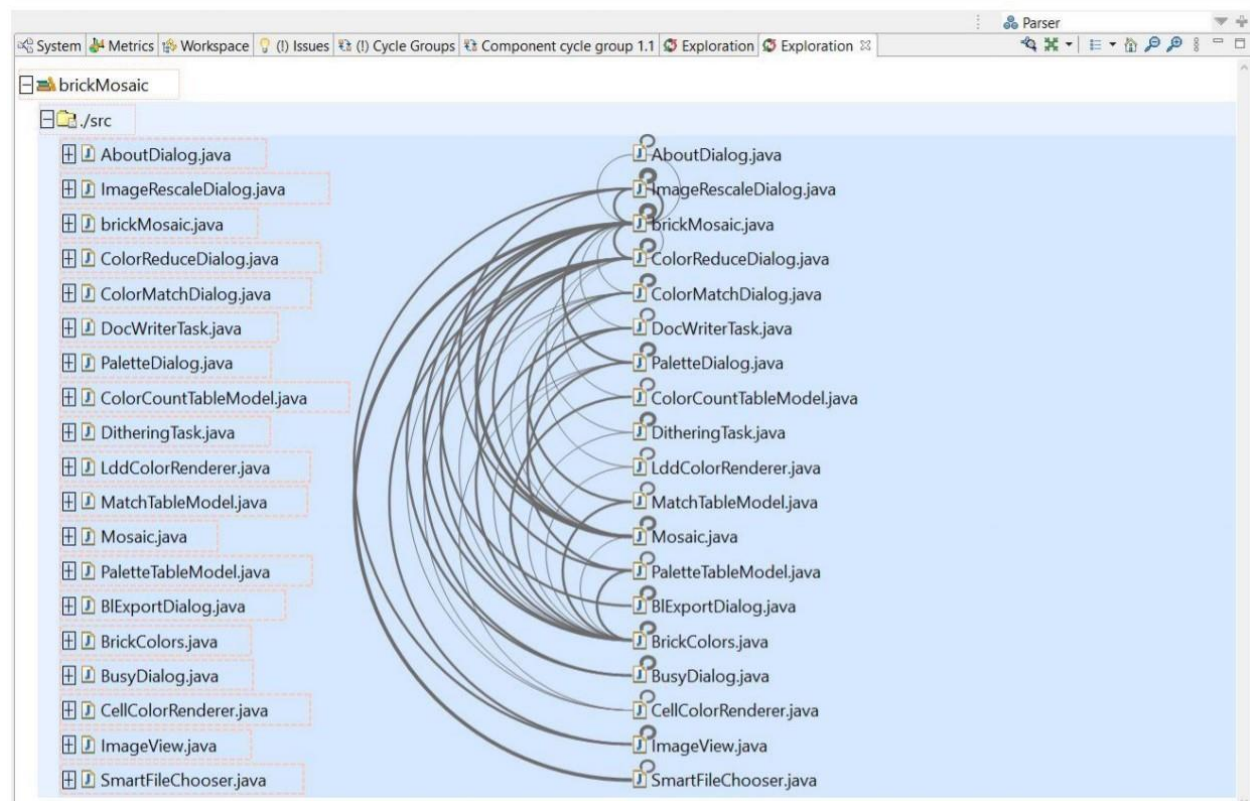


Figure 5

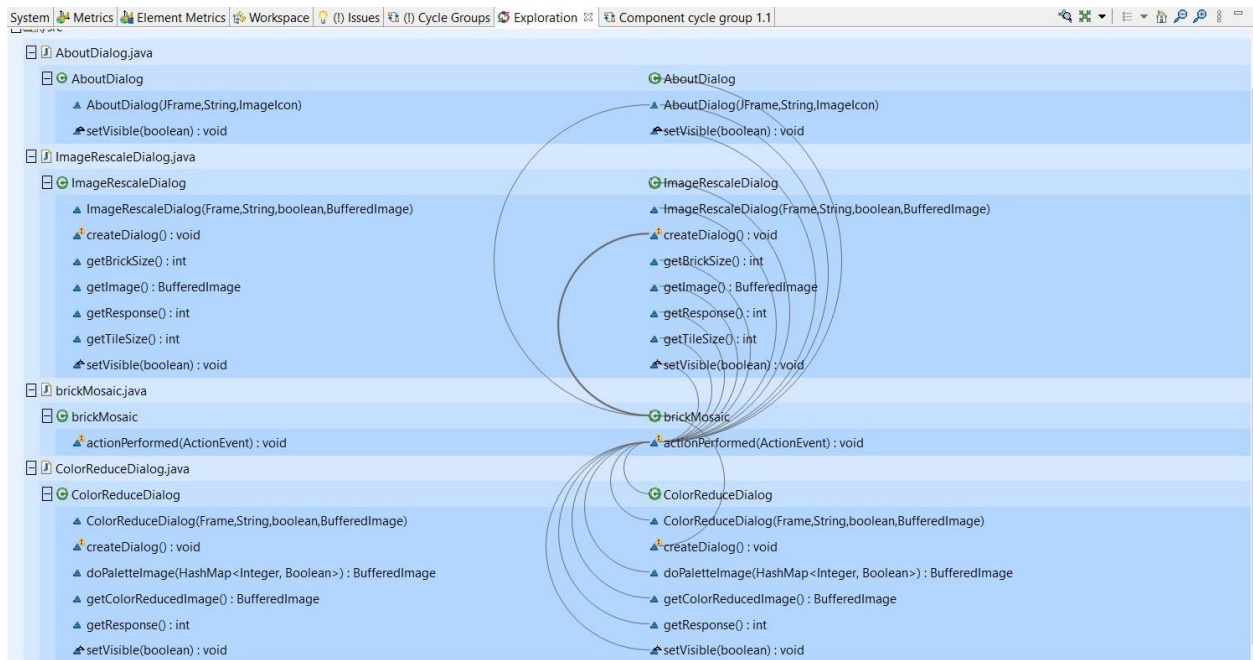


Figure 6

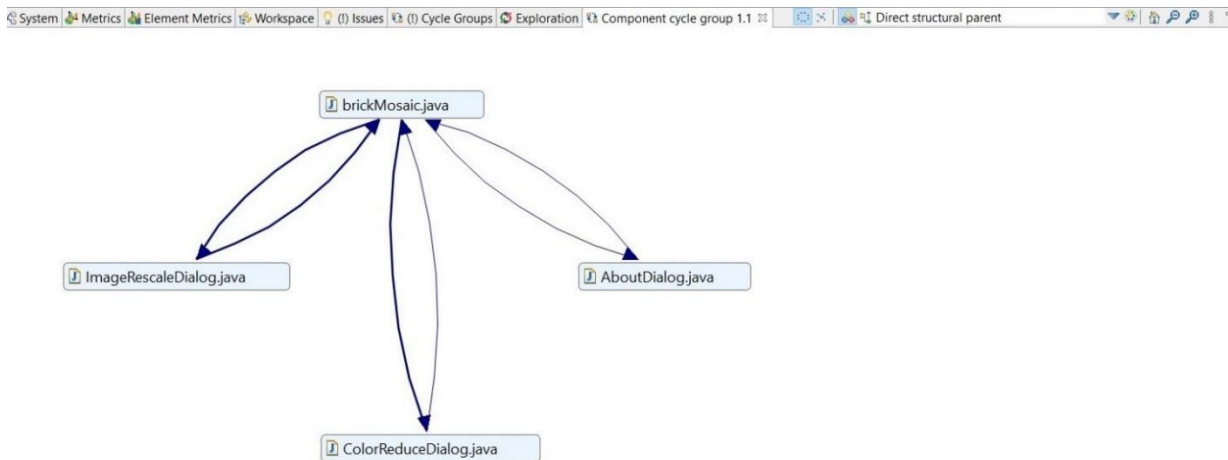
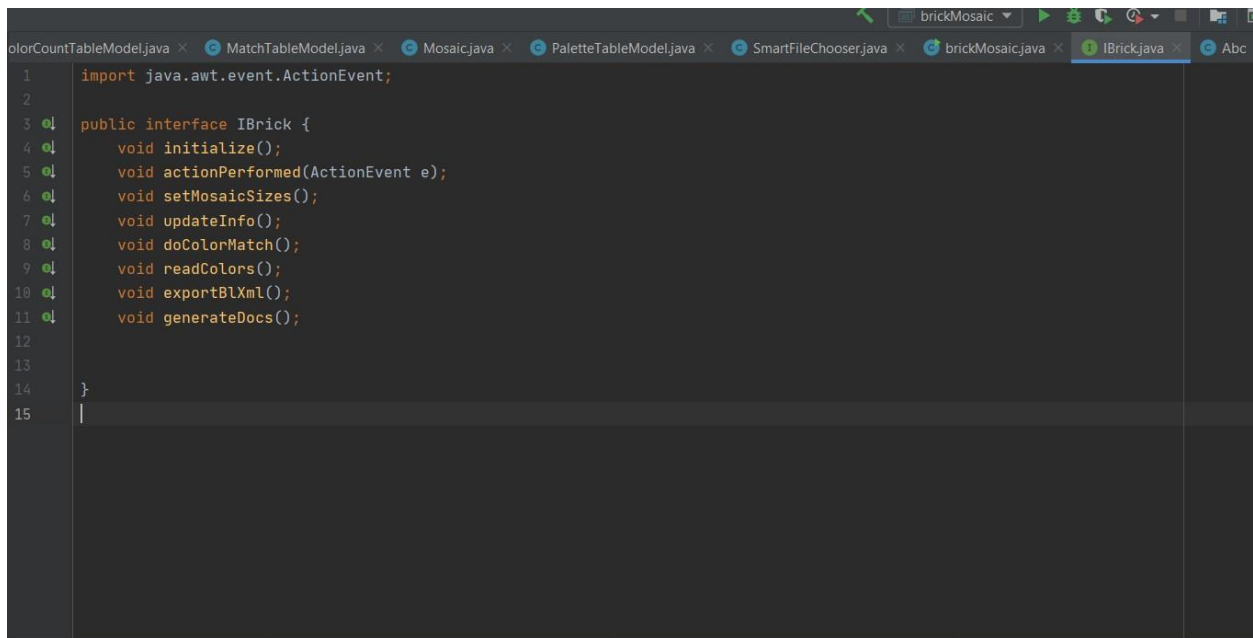


Figure 7

Figure 7 Shows that the issues related to component cycle group originate from brickMosaic class. In other words, the shown classes are relying on brickMosaic classes due to bad implementation. To deal with this problem I decided to take advantage of design pattern dependency inversion so that all implementations of the interface class depend on it. This can be done by creating interface and extract the methods from brickMosaic class that were in use so instead of being depended on one class (brickMosaic as shown in figure 7), the methods are called from the interface, as a result the coupling will be removed and there will be no need for other classes to rely on this class.

To be more accurate, an interface must be created containing the brickMosaic methods and the brickMosaic class must implement the created interface named IBrick.

The IBrick interface class



```

1  import java.awt.event.ActionEvent;
2
3  public interface IBrick {
4      void initialize();
5      void actionPerformed(ActionEvent e);
6      void setMosaicSizes();
7      void updateInfo();
8      void doColorMatch();
9      void readColors();
10     void exportBLxml();
11     void generateDocs();
12
13 }
14
15

```

Figure 8

The logic behind this operation is that instead of invoking brickMosaic.class.getResource in the dependent classes AboutDialog, ColorReduceDialog, ImageRescaleDialog , So I will apply the following changes by implementing the interface in the code as for example :

```

prog = new JLabel(new
ImageIcon(brickMosaic.class.getResource("images/BrickMosaic.jpg")));

```

To


```
prog = new JLabel(new  
ImageIcon(IBrick.class.getResource("images/BrickMosaic.jpg")));
```

Therefore, the `brickMosaic.class.getResource` will be replaced by `IBrick.class.getResource` in mentioned dependent classes.

The extraction of the methods used by other classes and implementing them in the interface will result in more flexibility of the code.

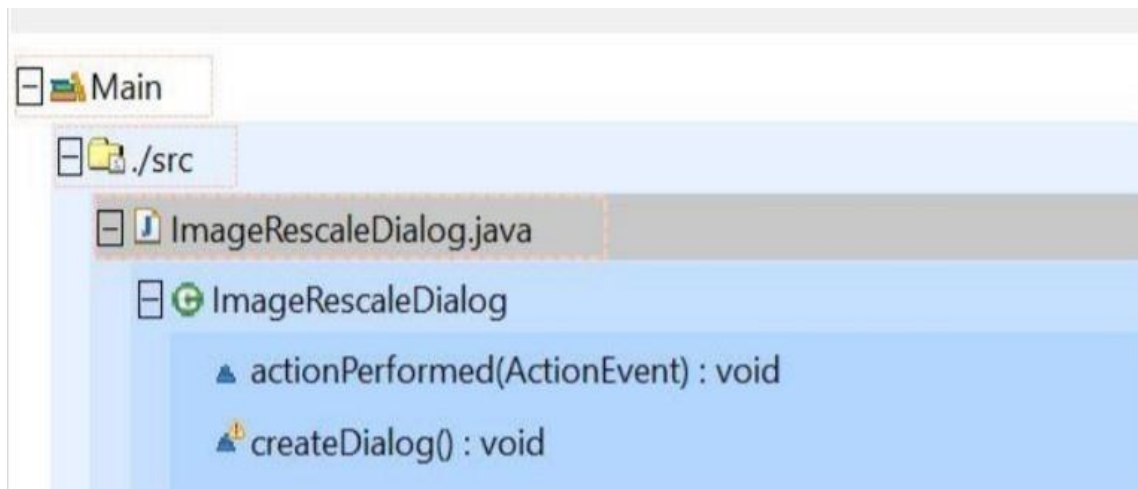


Figure 9

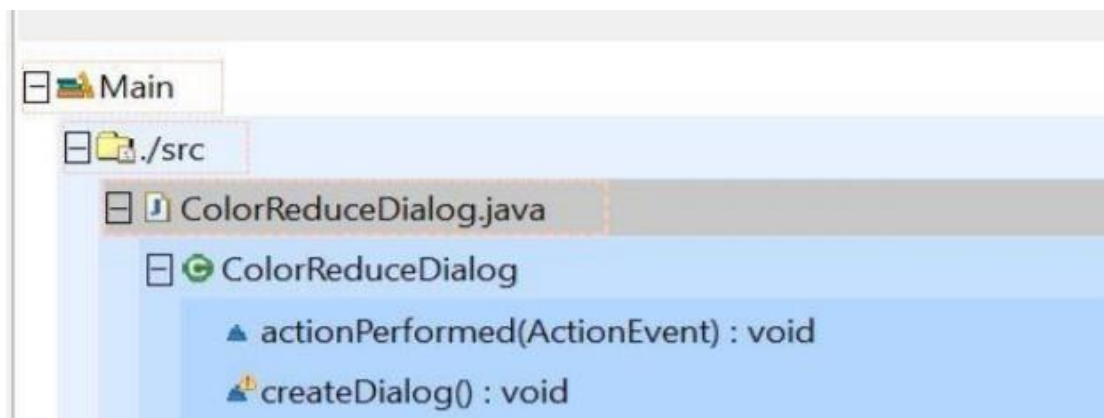


Figure 10

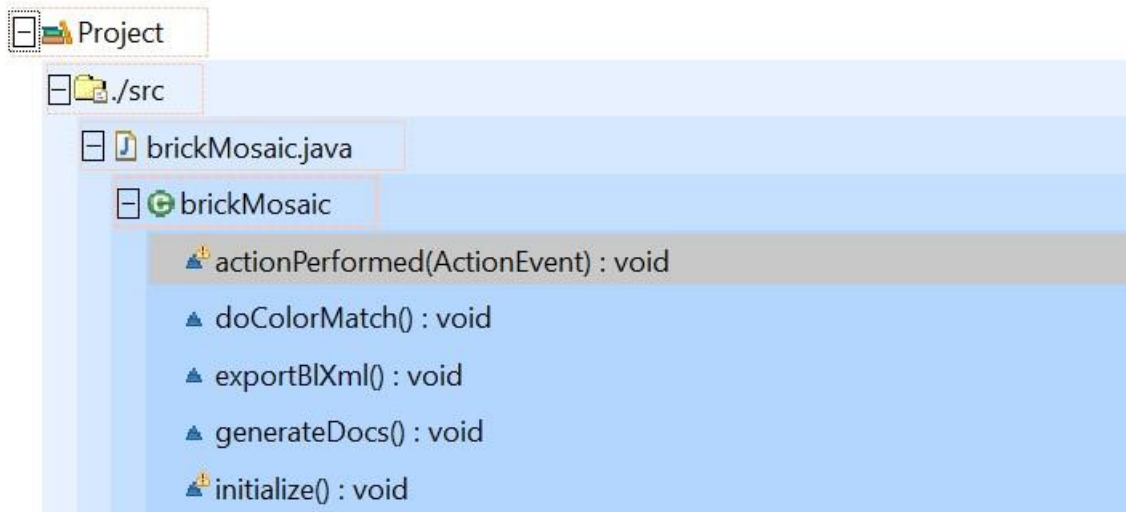


Figure 11

Next, we move to threshold violation issue. The threshold violation occurs due to violation in the number of allowed statements. Figure 9, 10, 11 depicted such violations in several classes.

In details we can observe in figure 9 that createDialog method in the imageRescaleDialog has more than the number of statements allowed. We can refer to the number of allowed statements in figure 3 which is 0 to 100 but while it is 174 in this method. The reason behind this is having abundant lines of code causing a severity of warning in the Sonargraph issues analysis. I decided to simplify the code by converting the above-mentioned method to smaller methods.

Figure 10 shows createDialog method in ColorReduceDialog class has exceeded number of statements. The number of statements must be in the interval of 0 to 100 but it is 115 due to exceeding line of codes and this has led to having an error with warning severity. So, I restructured the method by splitting it to smaller parts to make it simpler and follow the rules.

Moving on to figure 11, we notice that initialize method in brickMosaic class has outnumbered the allowed statement range (0 to 100). By referring to figure 3 it is noticeable that the number of statements is 192 that is above the range. This has caused a warning in Sonargraph issues analysis, Therefore, to mitigate this error I divide the large method into the smaller methods like previous methods to make the code simpler. After that I must deal with another warning caused by actionPerformed method in this class. As mentioned, the number of allowed statements must be ranged from 0 to 100 while it is 120 and it not following the defined rules for this. So as a solution the method should be divided to smaller methods to remove the threshold violation along with improving the code simplicity and reusability.

Furthermore, As can be seen in figure 11 and figure 3, There is another threshold violation (modified cyclomatic complexity) for actionPerformed method in BrickMosaic class. The permitted number of statements is from 0 to 15 while it has violated this rule with 24 so it requires 24 different tests to be done to check the implementation for testers. This happens for the reason of having too many numbers of If-statements which will increase the complexity and difficulty of testing the code. The severity of the case is warning, as a result I converted the mentioned method to smaller methods to remove this issue along with the complexity.

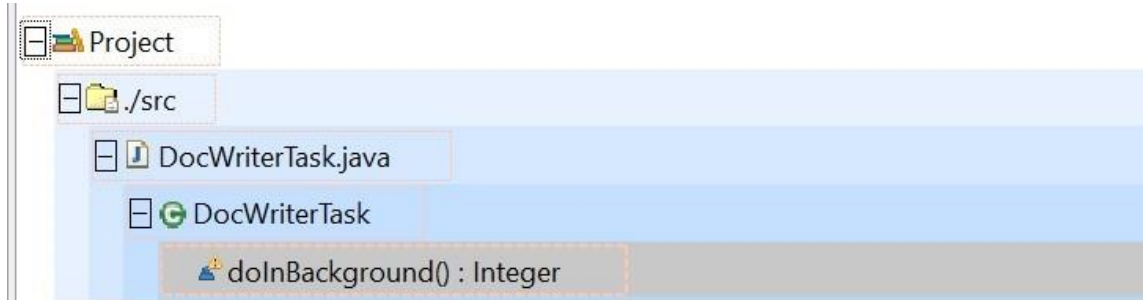


Figure 12

According to figure 3 and figure 12, It can be observed that there a threshold violation in doInBackground method in DocWriterTask class. It is caused by exceeding the number of allowed statements defined from 0 to 100. In this method, the violation number is 227 due the high number of codes in this method. This has result in warning severity in Sonargraph issue analysis. I resolve this issue by splitted the method into smaller methods to have simpler code and increase the reusability.

More to add, the other threshold violation for this class refers to modified cyclomatic complexity. The doInBackground method in this class has violated the allowed range (0 to 15) and the violated number is 20. This error with warning severity has occurred because of the high number of If-statement in above-mentioned method which makes the testing complex. So, this necessitates to apply 20 different tests by testers to evaluate and check this method. To solve this issue and for the sake of simplicity I convert the method to smaller methods. By solving this issue, I increased the clarity and the simplicity of the code.

Metric view on routine level:

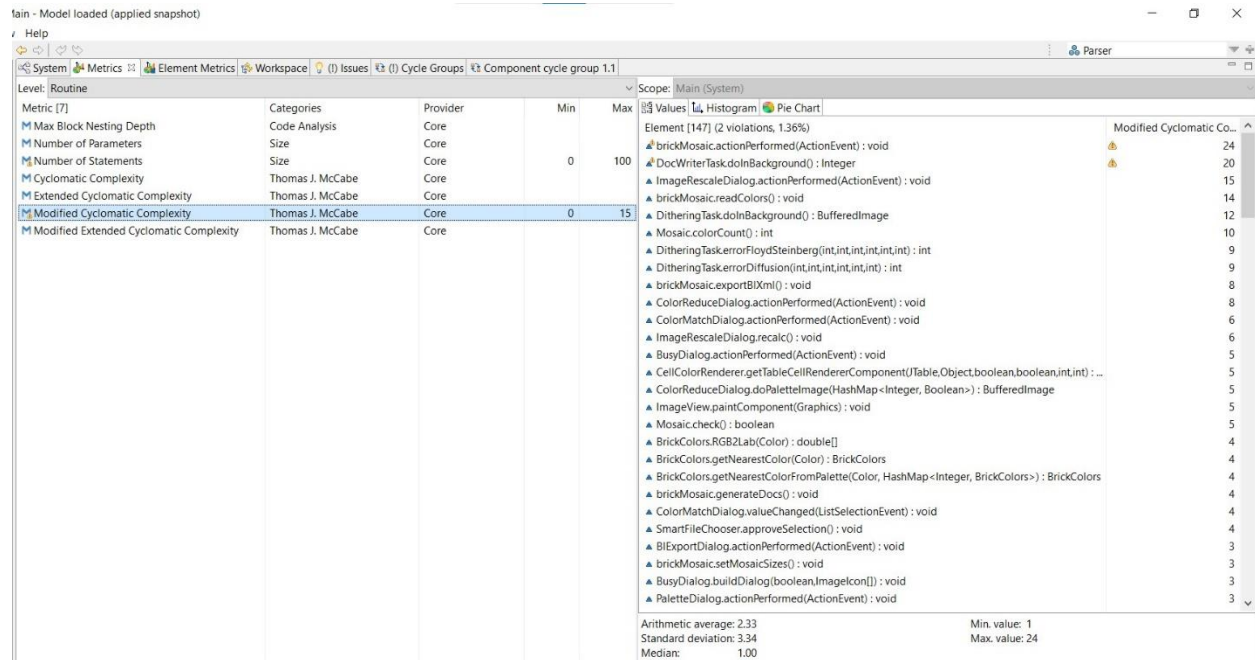


Figure 13

Number of statements in metric view:

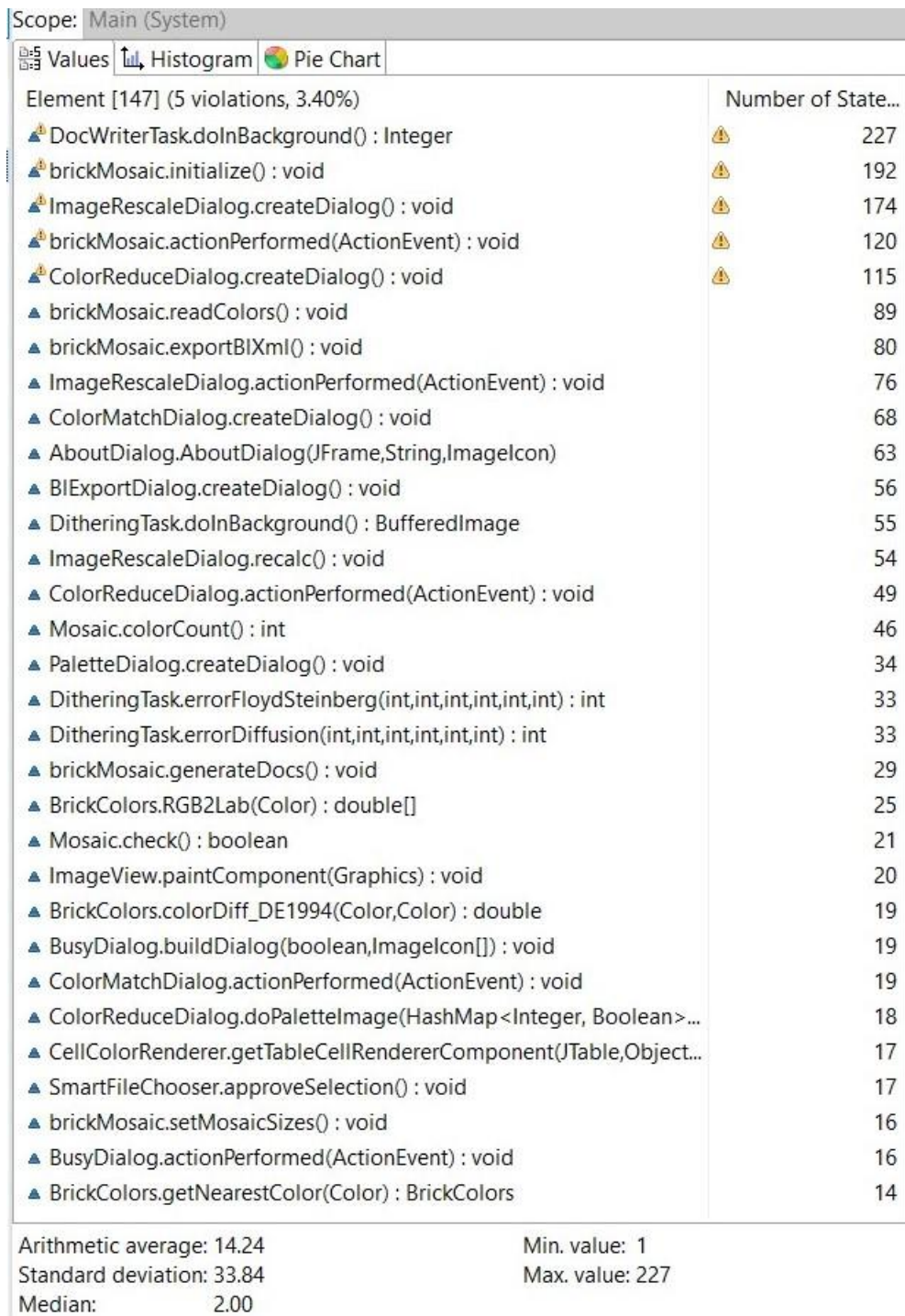


Figure 14

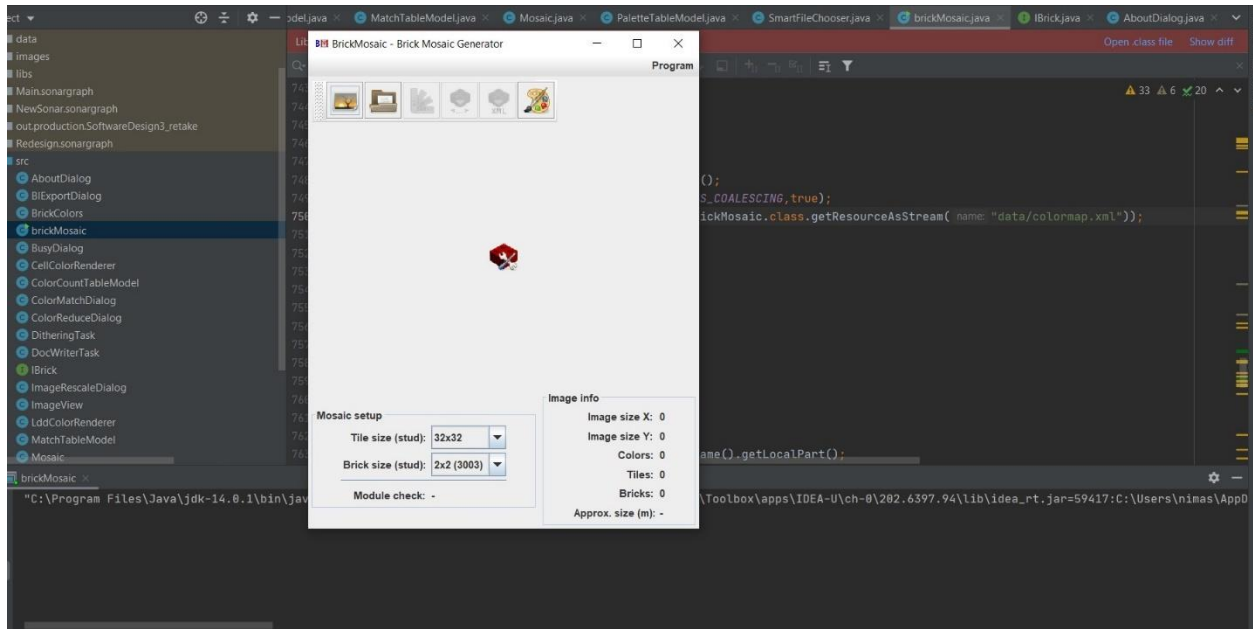


Figure 15

Task 2

Resolve all issues found in Task 1

As mentioned in the previous task the issues were due to 1 cyclic relationship and 7 threshold violations.

Removing the cyclic component issue:

According to lectures, Robert C Martin suggested to create on abstractions rather than on implementations because the key it is the to key to have a flexible architecture is low coupling and high cohesion. Building on abstraction will result in financial benefits and budget control.

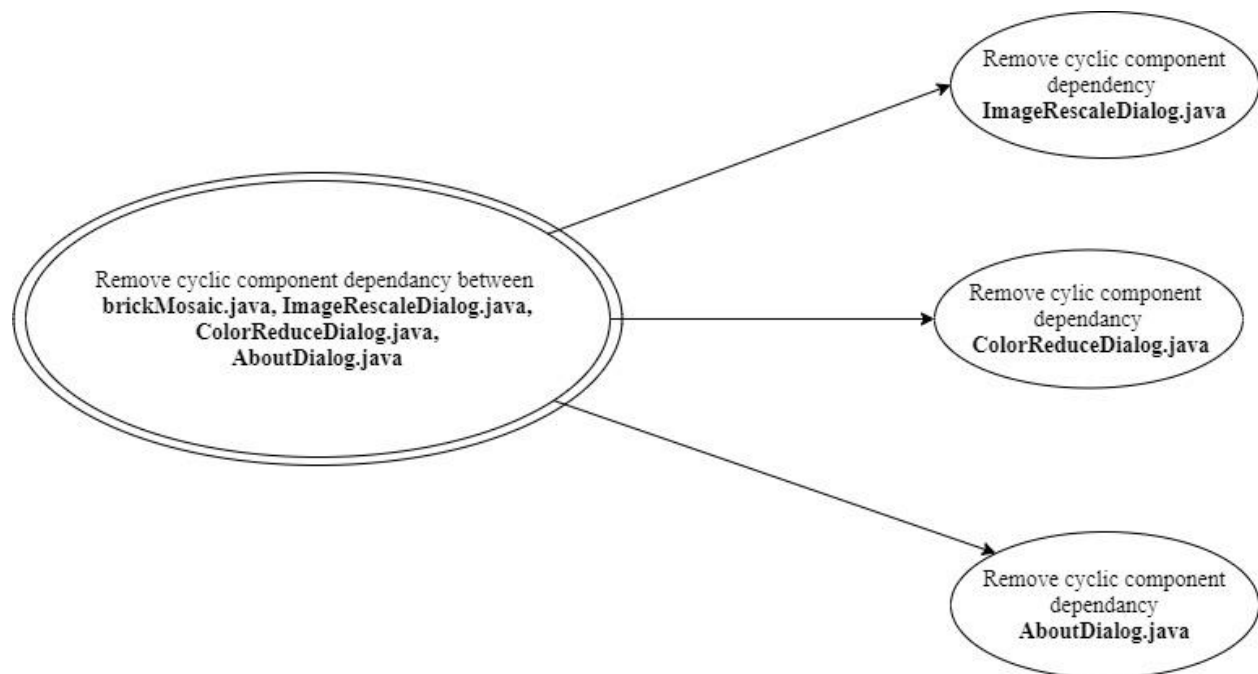


Figure 16

Figure 16 illustrates the 3 classes detected by Sonargraph that have cyclic dependency. The classes involved are ImageRescaleDialog, ColorReduceDialog, AboutDialog. I mainly must remove class dependency for the shown classes.

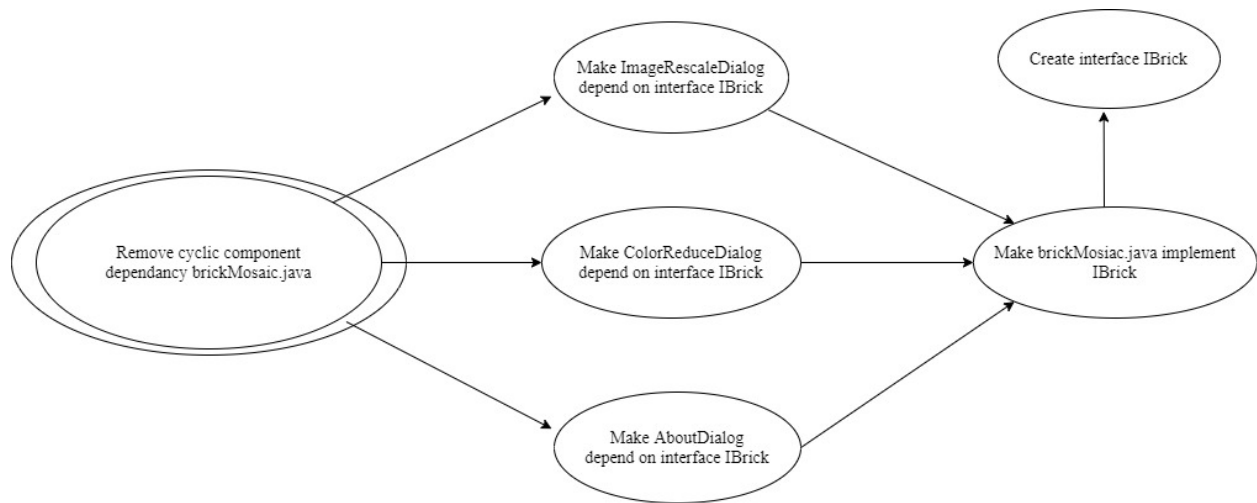


figure 17 – The plan for removing cyclic dependency

I have taken advantage of Mikado method to devise a plan to overcome cyclic component dependency issue. In figure 17, the assumed plan to remove the cyclic component dependency from the involved classes is depicted step by step. The 3 classes namely ImageRescaleDialog.java, ColorReduceDialog.java and AboutDialog.java are depended on brickMosaic.java. The way to mitigate this issue is to modify the classes to depend on the created interface of brickMosaic.java called IBrick which consequently will lead to improvement in system ACD (shown in Figure 2, valued 5.63 before applying Mikado plan). In the next step, I make brickMosaic depend on the IBrick interface by implementing it. To clarify, I extract the necessary methods out of brickMosaic and move them to the IBrick). As a result, the cyclic dependency will be removed and system ACD will be decreased to 3.45 (see figure 19).

Removing threshold violation error:

As illustrated in figure 3 and table 1, there are 7 threshold violation errors that caused by different methods in several classes. These violations occur due to exceeding number of statement or large number of If-statements. The number of allowed statements for all the issues is from 0 to 100.

Class name	Method	Issue type
ColorReduceDailog	createDialog ()	Number of statements
ImageRescaleDialog	createDialog ()	Number of statements
brickMosaic	actionPerformed(ActionEvent e)	Number of statements
brickMosaic	actionPerformed(ActionEvent e)	Modified cyclomatic complexity
brickMosaic	Initialize ()	Number of statements
DocWriteTask	doInBackground ()	Number of statements
DocWriteTask	doInBackground ()	Modified cyclomatic complexity

Table 1

- The first 1 error is located ColorReduceDialog.java, createDialog() method. This method has 115 statements which are higher than allowed criteria. So, I created a void method called colorSetting () to solve this error.
- The second error is from the ImageRescaleDialog.java class in createDialog() method. This method 174 statements. To solve it, I created two methods: ImageSetter() and ButtonAndGbc().
- The next 3 threshold violations are related to brickMosaic.java class in actionPerformed (ActionEvent e) method. The first one is for violation in number of statements in this method which 120 statements. Therefore I created a method called imgOrganizer() method to mitigate this problem.

The second issue is for modified cyclomatic complexity with a value of 24 (allowed value is 0-15). I created the checker () method to decrease the complexity.

The third threshold violation in brickMosaic.java is in initialize () method with violation value 192. Therefore, I created two methods called fileSetter() and frameSetting() to fix this violation.

- 2 Threshold violations in DocWriteTask : The first threshold violation for this class is in doInBackground() method due to number of statements(227) which is the highest among all violations. So, I created two methods namely genralOutlook() and colorOfTable(). The next issue in this class is modified cyclomatic complexity with value of 20 (allowed value is 0-15) in the same method that is solved by creating insTable() method.

The reason for modified cyclomatic complexity issues is mainly excessive number of if-conditions so I traced down the issues by using Sonargraph application and in practice I designed new methods and moved the if-condition to these methods to decrease the complexity level also make them easier to test. Therefore, the threshold violations are fixed by creating smaller methods but with the same function so it will increase the cohesion and decrease the complexity which makes the implementation easier to understand.

2.2. Re-architect the application according to the well know Multi-layer Architectural Pattern

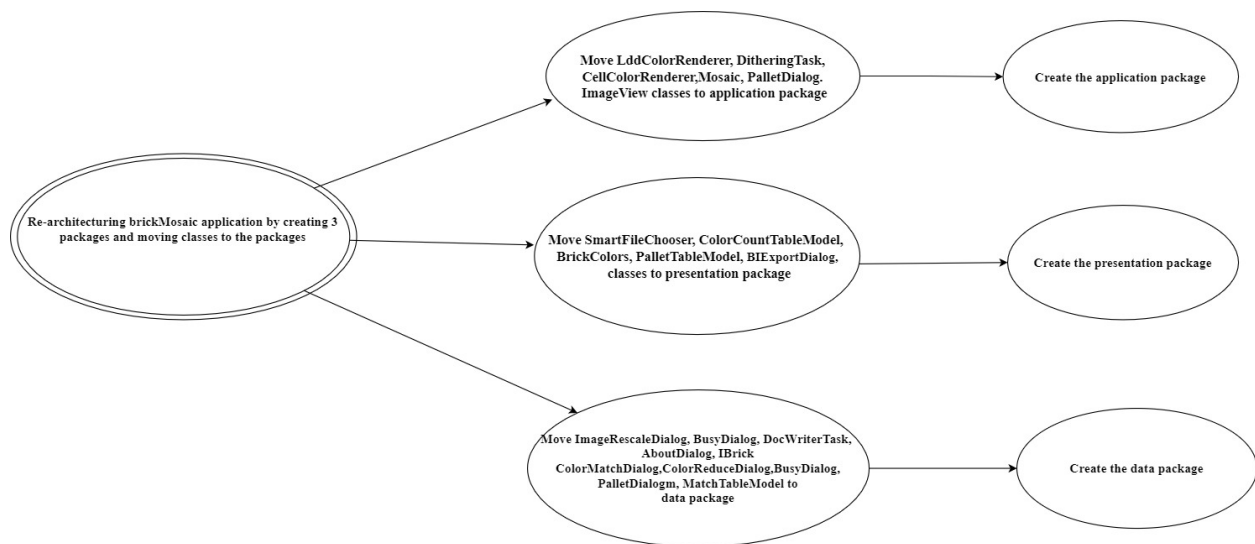


Figure 18

Figure 18 shows the process of applying multi-layer architectural pattern to brickMosaic application. As you can see, 3 packages were created namely application, presentation, data, and the classes were moved to these packages. This classification of classes for moving into the relative packages are based on their responsibility and functionality. However, I decided to keep birckMosaic class which is a main class out of the packages. Applying this pattern will result in

increased maintainability, and browsability. Besides, it also makes it easier to update the code if necessary – when the architecture is broken up into multiple layers, the changes that need to be made are simpler and less extensive and at the same time each layer can communicate better with layer beneath it.

Task 3 Re-engineering

Sonargraph system view:

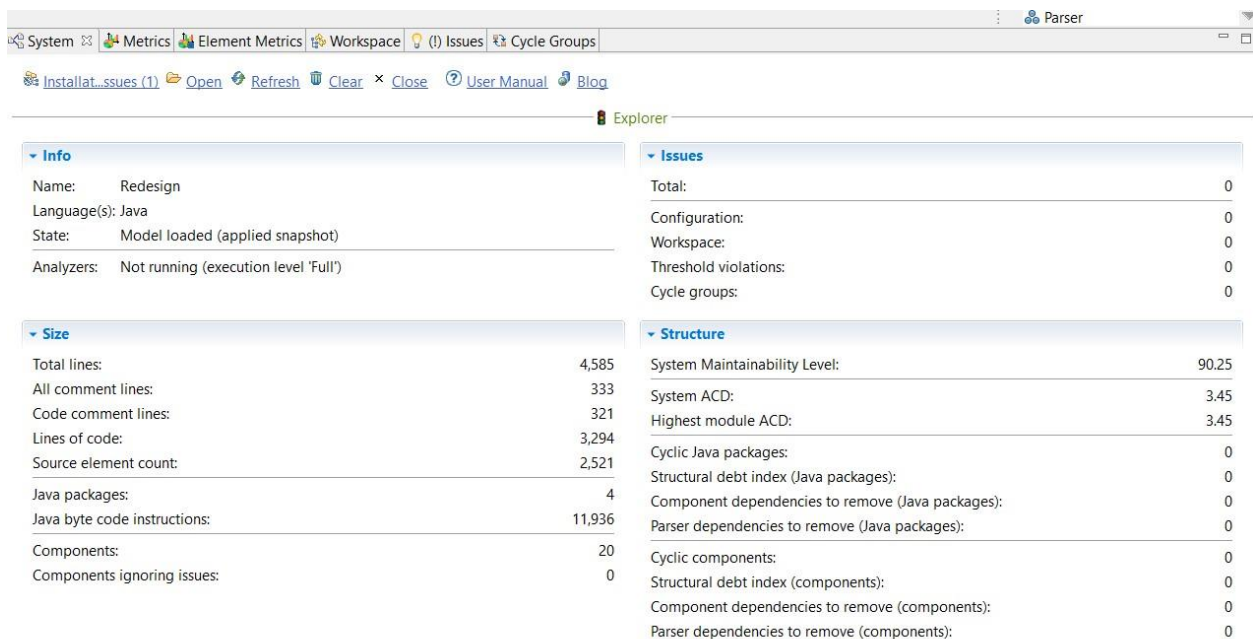
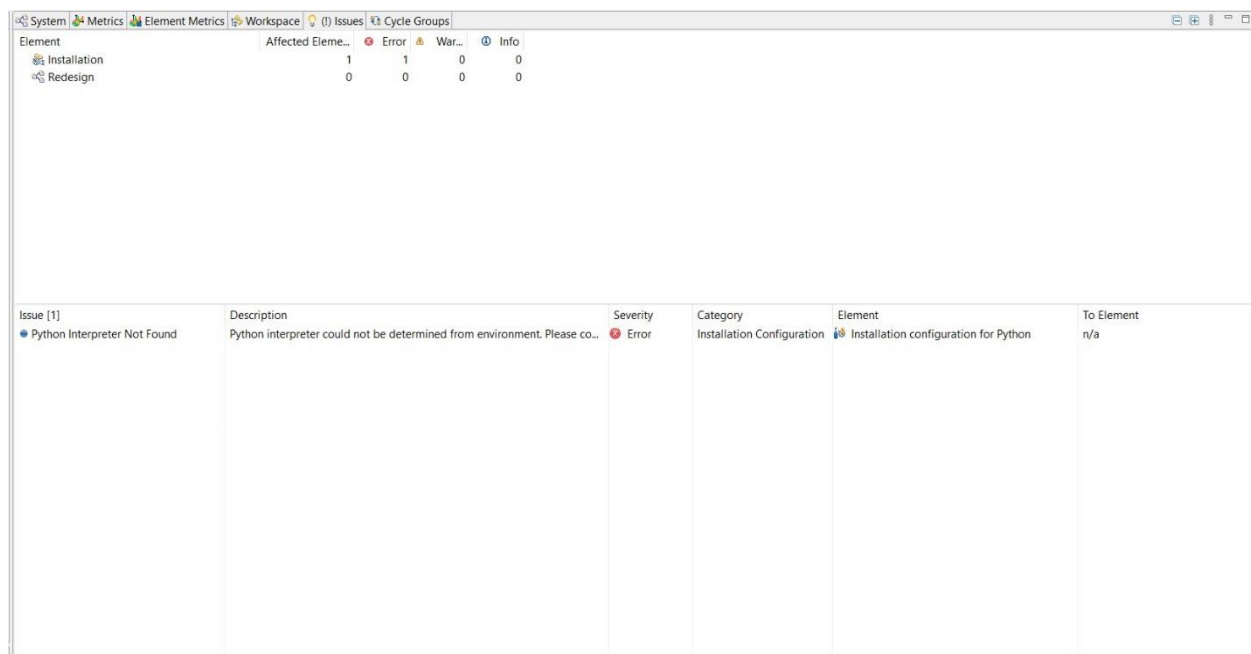


Figure 19

As shown in figure 11 the changed are applied to the information comparing to figure 1.

- Total lines of code are increased due to adding methods to the application.
- The number of code comment is decrease since the commented lines were removed.
- An Important point to notice is that the number of issues has changed to zero.
- The number of packages is now 4 since the application is re-architected.

- The number of components is increased because of adding an interface to the implementation.
- ACD is decreased to 3.45 that means classes depend on other classes with the average of 3.45 after re-designing the software so it means the dependency is decreased.
- The number of cyclic java packages and structural debt index are 0 even though I added packages to the implementation and there is no dependency for this part.



Issue	Description	Severity	Category	Element	To Element
Python Interpreter Not Found	Python interpreter could not be determined from environment. Please co...	Error	Installation Configuration	Installation configuration for Python	n/a

figure 20

As illustrated in figure 20, after the re-designing of software the number of issued related to threshold violation, cyclomatic complexity and cycle group is now 0 (The only issue exist is related to installation that is not the target of this assignment). As a result, no threshold violation and cycle component group warning can be observed.

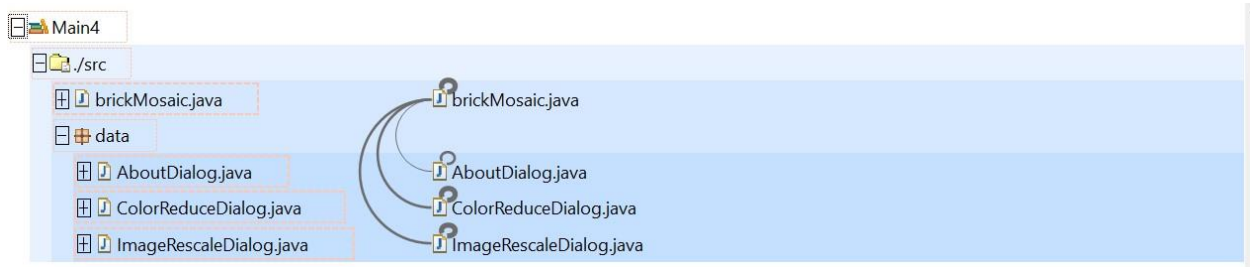


Figure 21 New exploration view after re-design

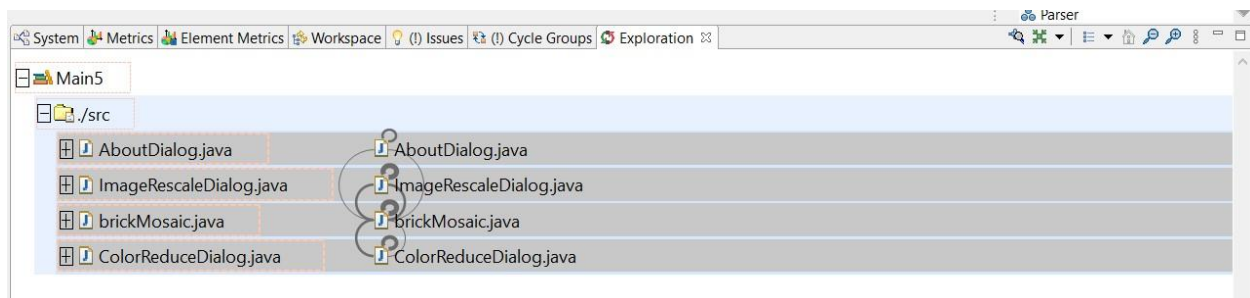


Figure 22 before Re-design

As can be seen by comparing figure 21 & 22, the dependencies are now changed there are no upward going dependencies and the right arc are removed.

Exploration view for the whole application:

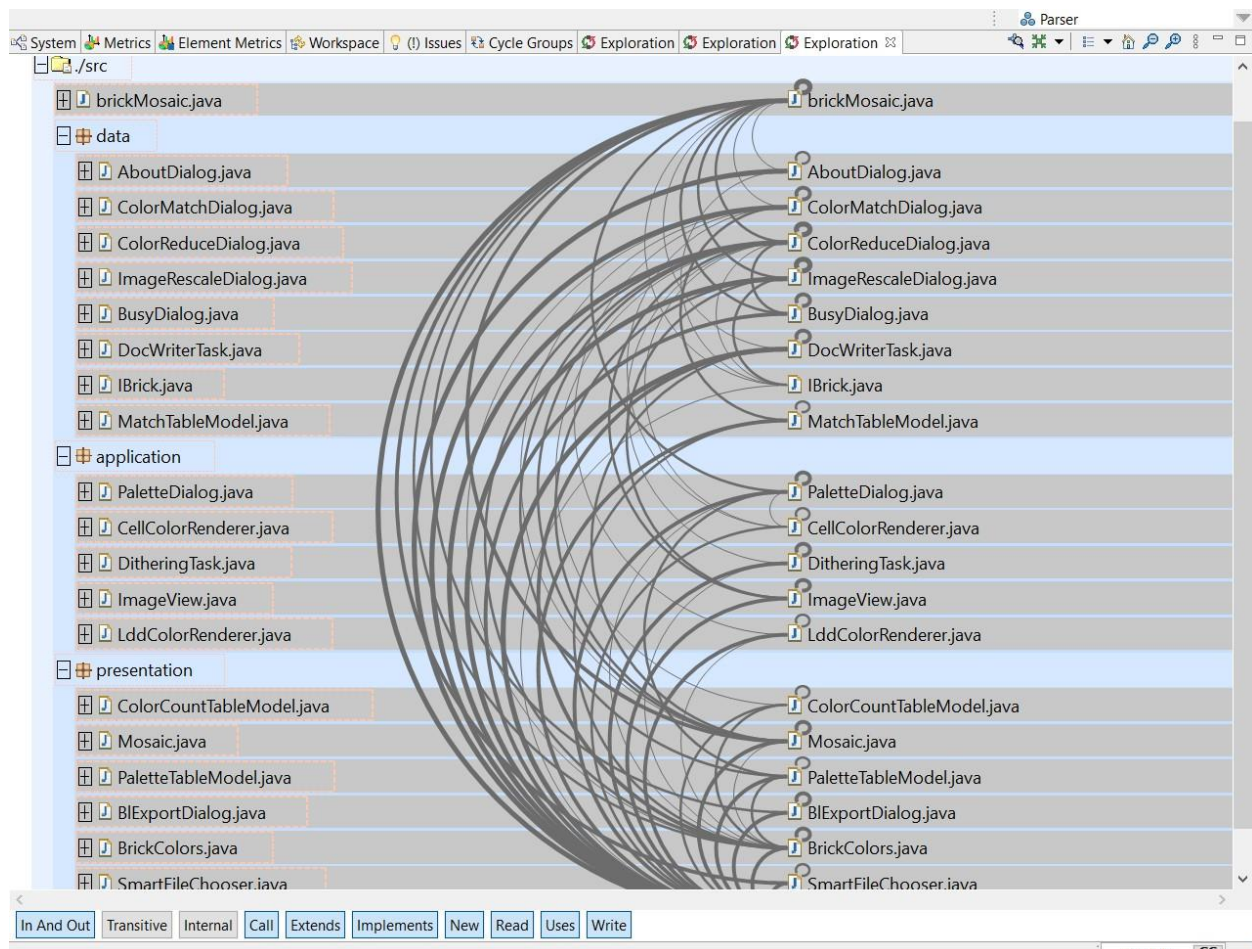


Figure 23

Old metric view Routine Vs new metric view

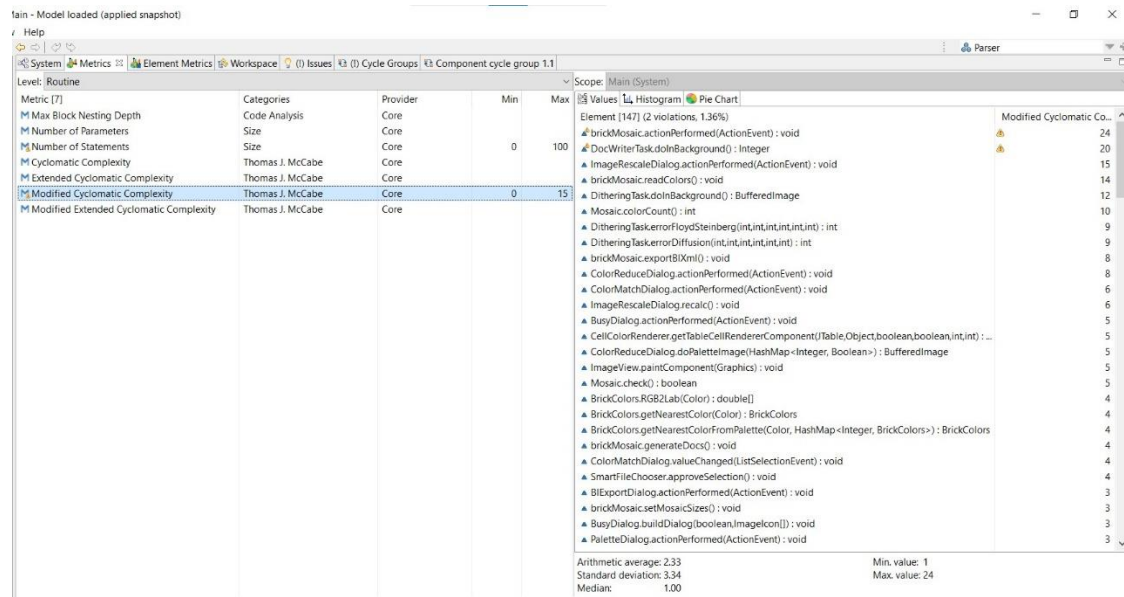


Figure 24 old metric view

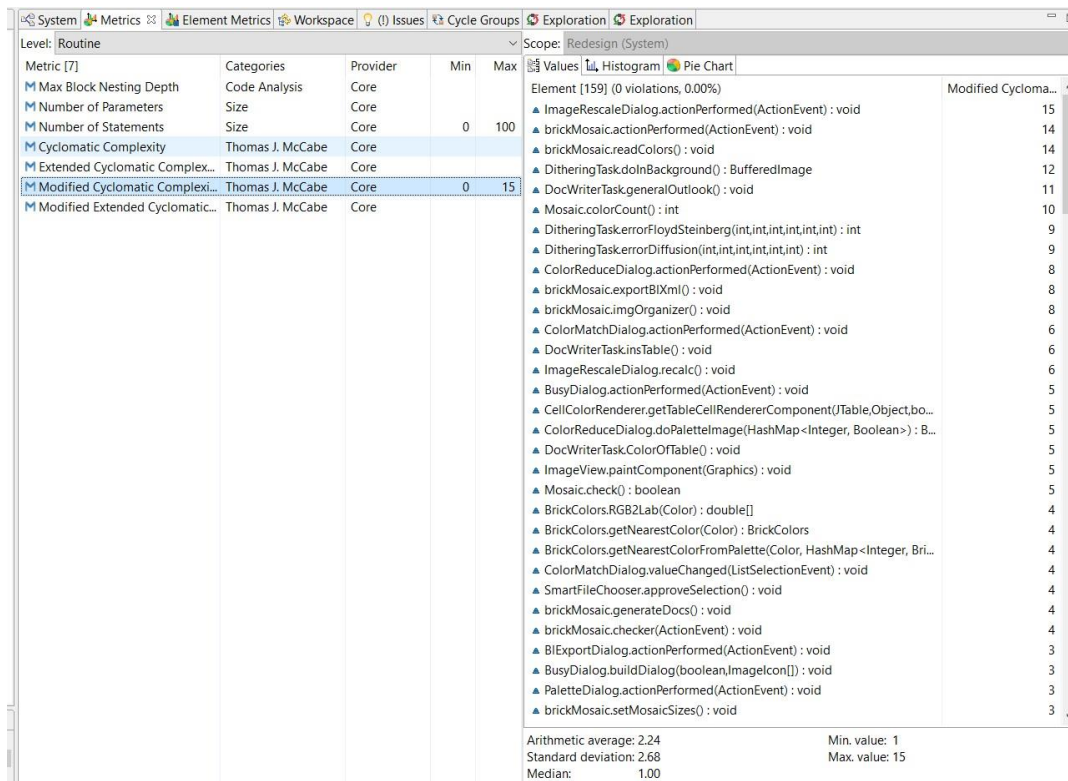


Figure 25 new metric view