

Linnaeus University

Faculty of Technology – Department of Computer Science

2DV608 – Software Design

Student: Mohammadali Rashidfarokhi



Assignment 3- AD

1. Task 1 – Codebase Analysis

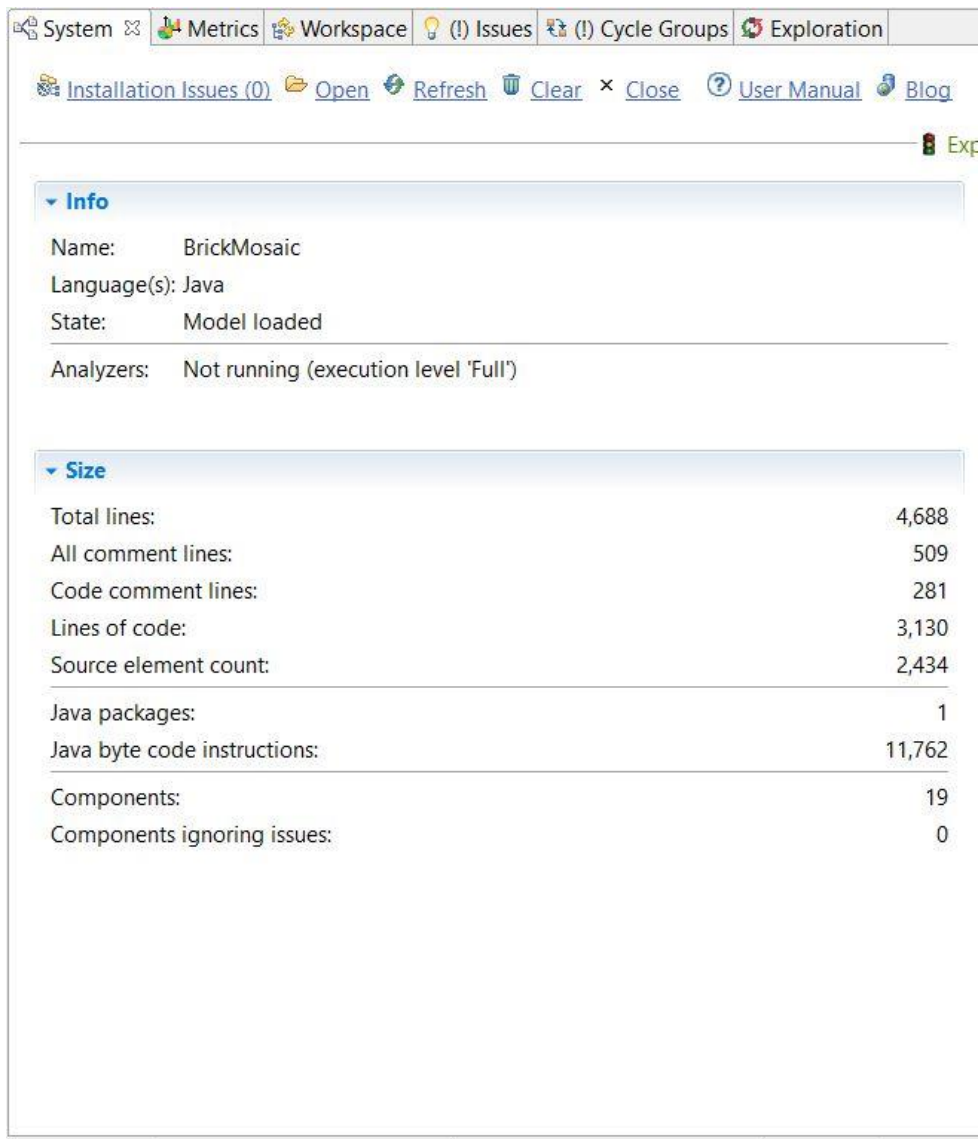


Figure 1

Figure 1 will indicate the first part of the system analysis. Consequently, a total number of 4,688 lines is existing. However, 509 lines are considered all comments lines and 281 code comment lines. Regarding the java packages, only 1 package has been detected. Lastly, the number of classes (components) is 19.

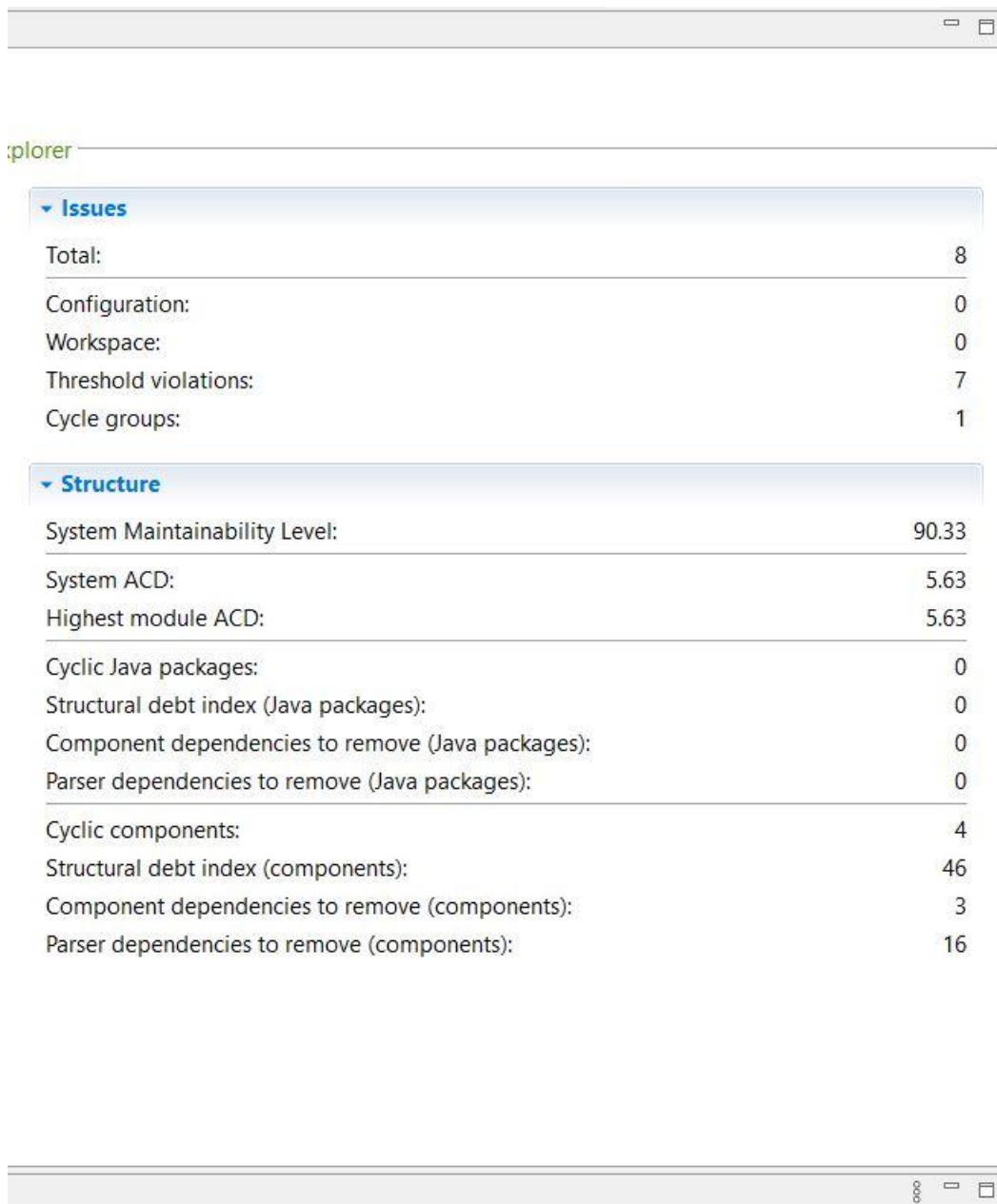


Figure 2

Figure 2 will demonstrate the rest of the information belong to the system analysis. To begin, a piece of noticeable information in this figure would be the number of the issues which is 8. Noticeable issues would be threshold violation and component cycle group. However, full detail about these issues and how to tackle them will be covered later.

Moving on, it is undeniably clear that the maintainability level is quite high based on the size of the project (90.33). The ACD for the system has been saved as 5.63. What I mean by this is that each class is dependent on other classes with an average of 5.63. Also, there are 4 cyclic java packages. Therefore, the debt index would be 0. Noticeably, the number of dependencies would be 0 as well.

Regarding the cyclic components, the structural debt index is 46. To be able to tackle this issue and eliminate the dependency from these 4 cyclic components, 3 dependencies should be eliminated.

ColorReduceDialog.java						
Parser						
Element	Affected Eleme...	Error	War...	Info		
BrickMosaic	9	0	8	0		
Workspace	9	0	8	0		
Main	9	0	8	0		
Installation	0	0	0	0		

Issue [8]	Description	Severity	Category	Element	To Element	Provider
Component Cycle Group	Java Module 'Main' contains 4 cyclic components	Warning	Cycle Group	Component cycl...	n/a	Core
Threshold Violation	Number of Statements = 115 (allowed range: 0 to 100)	Warning	Threshold Violati...	createDialog() : ...	n/a	Core
Threshold Violation	Number of Statements = 174 (allowed range: 0 to 100)	Warning	Threshold Violati...	createDialog() : ...	n/a	Core
Threshold Violation	Number of Statements = 192 (allowed range: 0 to 100)	Warning	Threshold Violati...	initialize() : void	n/a	Core
Threshold Violation	Number of Statements = 120 (allowed range: 0 to 100)	Warning	Threshold Violati...	actionPerformed...	n/a	Core
Threshold Violation	Modified Cyclomatic Complexity = 24 (allowed range: 0 to 15)	Warning	Threshold Violati...	actionPerformed...	n/a	Core
Threshold Violation	Number of Statements = 227 (allowed range: 0 to 100)	Warning	Threshold Violati...	dolnBackground...	n/a	Core
Threshold Violation	Modified Cyclomatic Complexity = 20 (allowed range: 0 to 15)	Warning	Threshold Violati...	dolnBackground...	n/a	Core

Figure 3

Figure 3 will portrait the issues in detail. That is, as it has been mentioned before, 8 issues are existing in this project. One of the issues is Component Cycle group and the rest (7) are Threshold violation. The severity of all issues is just a warning.

Since in the source code of this project there is no package, the below picture will show its exploration view in the namespaces section in the internal part.

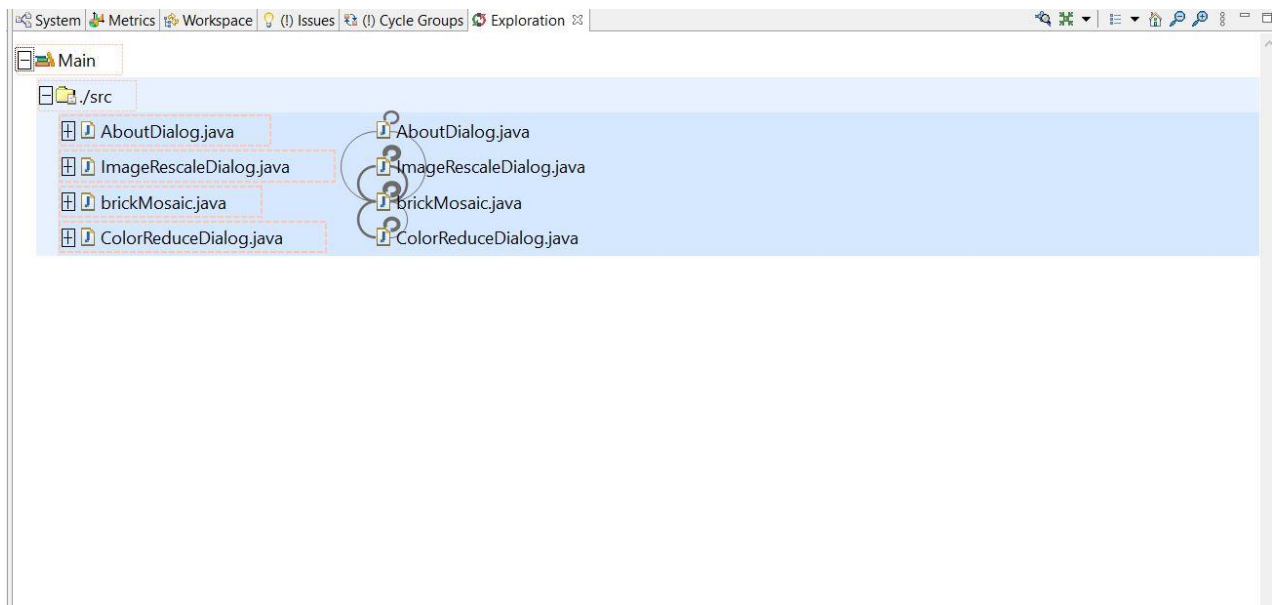


Figure 4

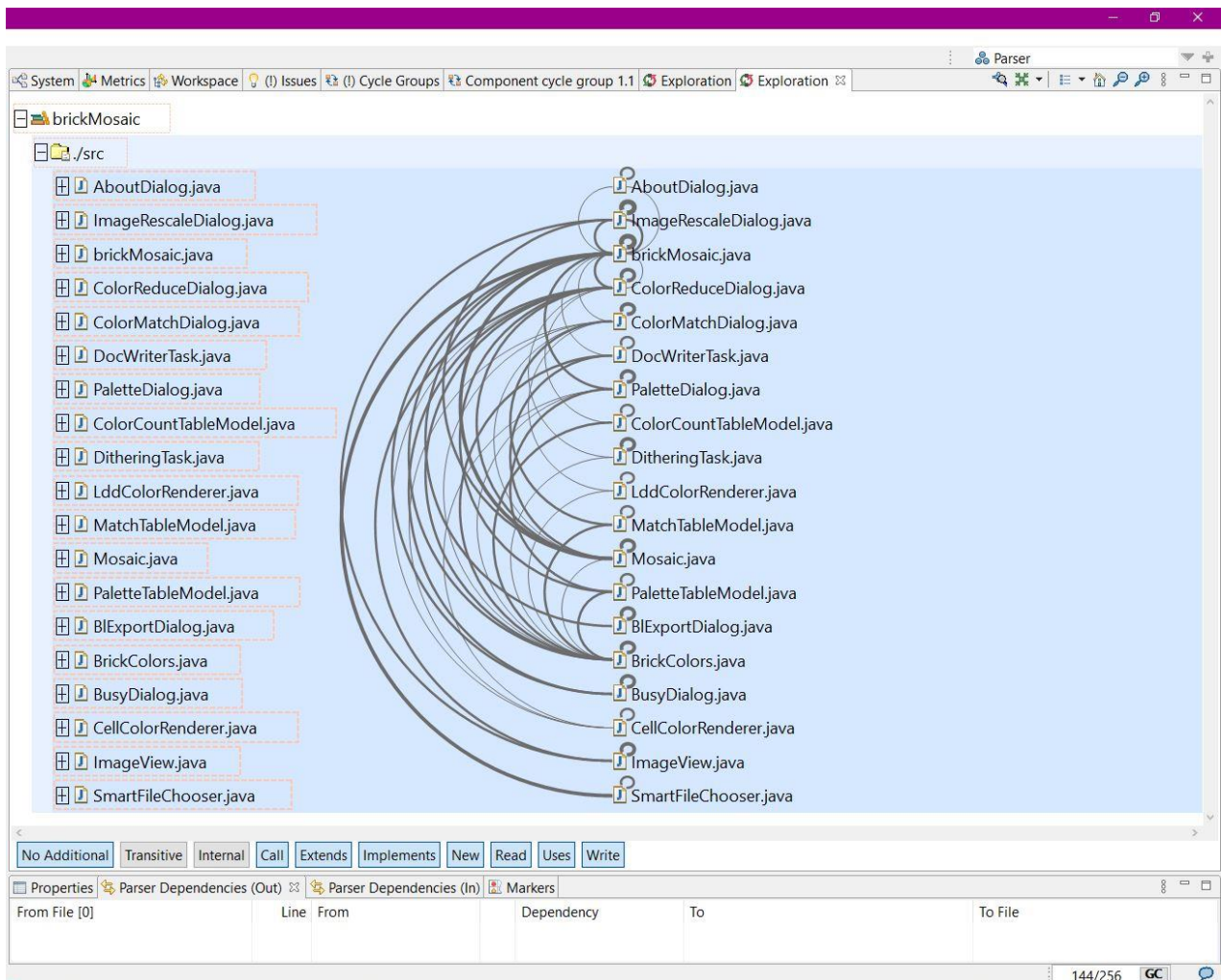


Figure5

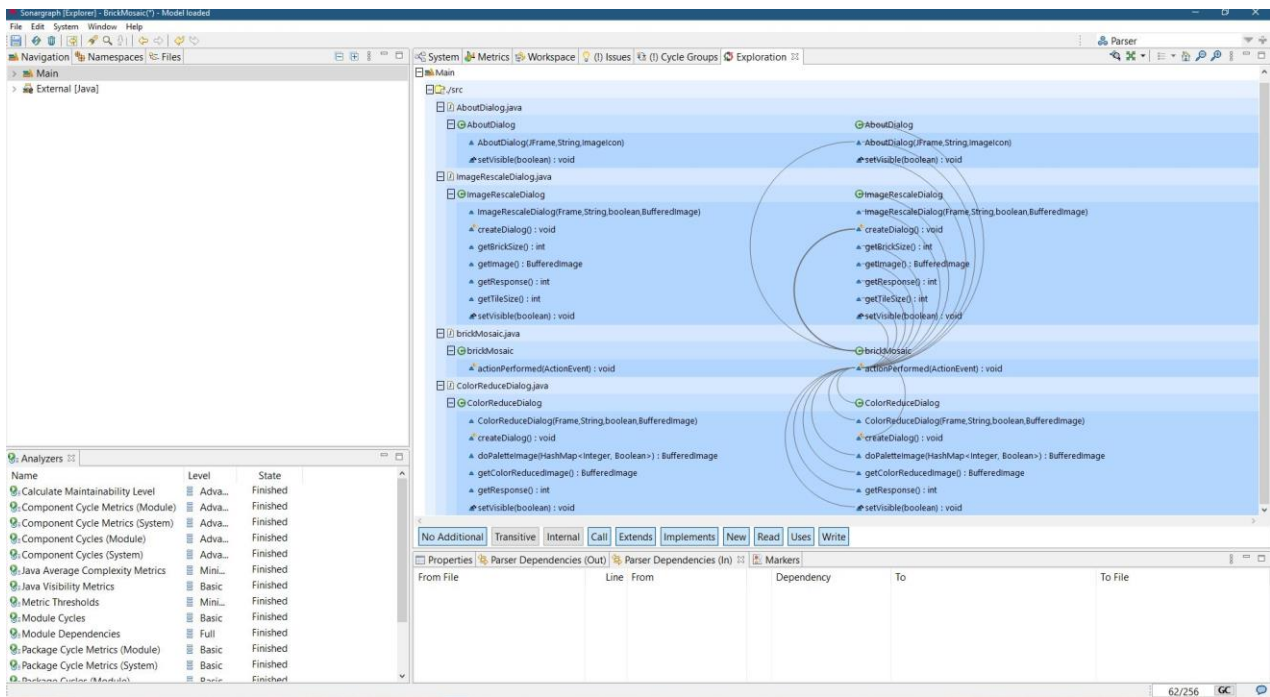


Figure 6

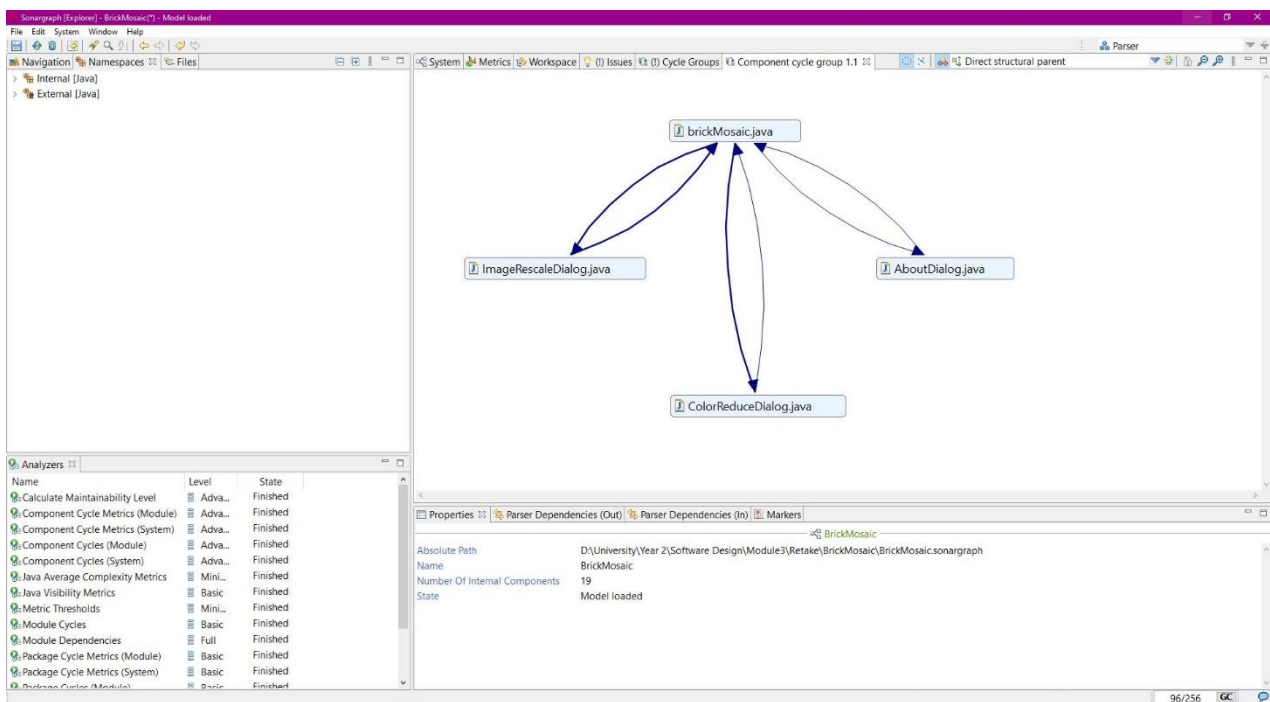


Figure 7

Regarding figure 7, an issue will rise from the BrickMosaic class. What I mean by this is that this class is dependent on its implementations which will result in 4 cyclic dependencies. Therefore, this would be considered bad practice. To tackle this issue, a solution would be inverting the design pattern dependency. What I mean by this is that an abstract class could be created containing the BrickMosaic's method signatures. Consequently, BrickMosaic class will extend the abstract class.


```

1  import java.awt.*;
2  import java.awt.event.ActionEvent;
3  import java.awt.image.BufferedImage;
4
5  public abstract class ABrickmosaic {
6
7      private void initialize(){
8
9      }
10     public void actionPerformed(ActionEvent e) {
11
12     }
13     private void setMosaicSizes() {
14
15     }
16     private void updateInfo() {
17
18     }
19     private void doColorMatch() {
20
21     }
22     private void readColors() {
23
24     }
25     public void exportBtXml() {
26
27     }
28     private void generateDocs() {
29
30     }
31 }

```

Figure 8

The reason for doing such action is that instead of calling `brickMosaic.class.getResource` each time in the classes `AboutDialog`, `ImageRescaleDialog`, and `ColorReduceDialog`, the abstract call will be used as follows:

`ABrickmosaic.class.getResource`.

The reason for making an abstract class of the `BrickMosaic` class is that this class will be used in several classes. Since the other classes are depending on the `BrickMosaic` class, this class could be exchange with an abstract class and result in the flexibility of the code.

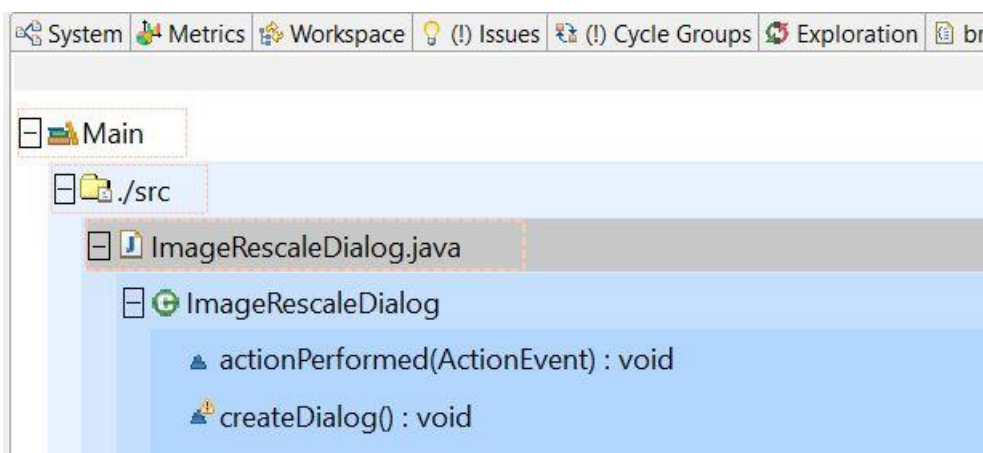


Figure 9

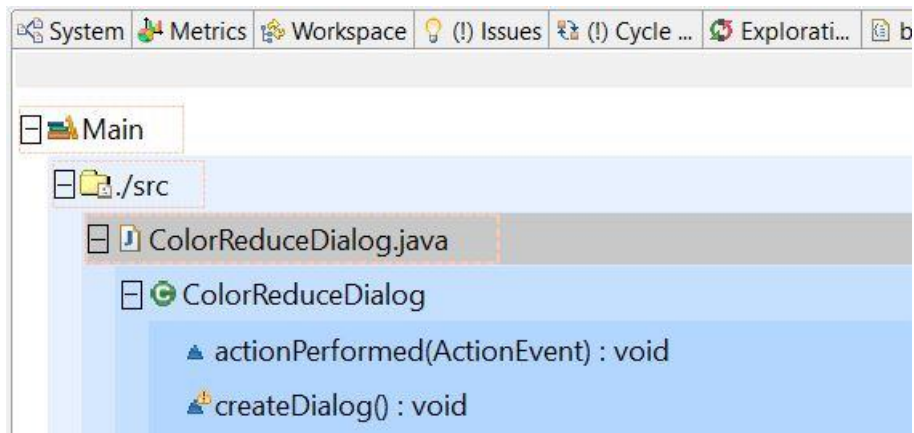


Figure 10

Figure 10 will demonstrate a new issue compared to the first one which is Threshold violation. Based on figure 10, the method `createDialog` is located in the class `ColorReduceDialog`. Moving on, based on figure 3, the Number of statements allowed to be in this method is supposed to be from the range of 0 until 100. However, in this case, the number has exceeded the range since it is 115. The main reason which is causing this problem is having many lines of codes. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

Figure 9 will demonstrate an issue which is Threshold violation. Based on figure 9, the method `createDialog` is located in the class `ImageRescaleDialog`. Moving on, based on figure 3, the Number of statements allowed to be in this method is supposed to be from the range of 0 until 100. However, in this case, the number has exceeded the range since it is 174. The main reason which is causing this problem is having many lines of codes. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

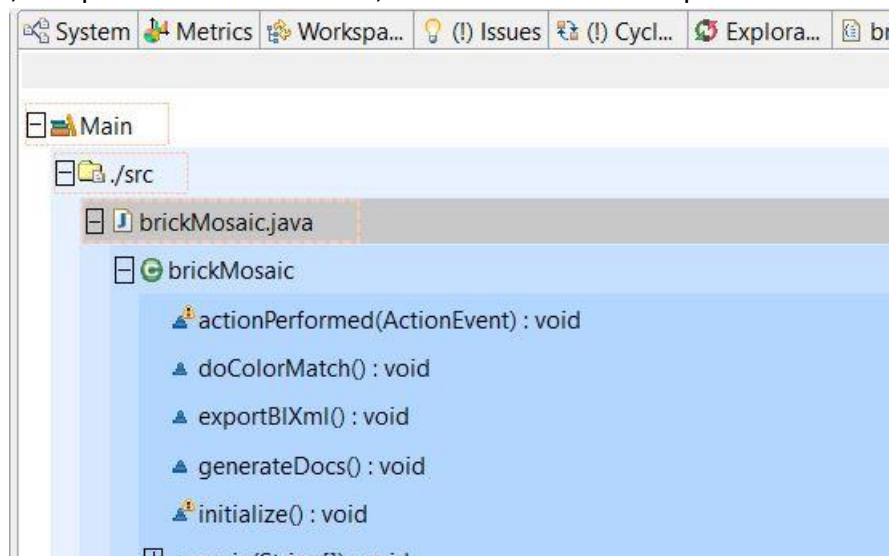


Figure 11

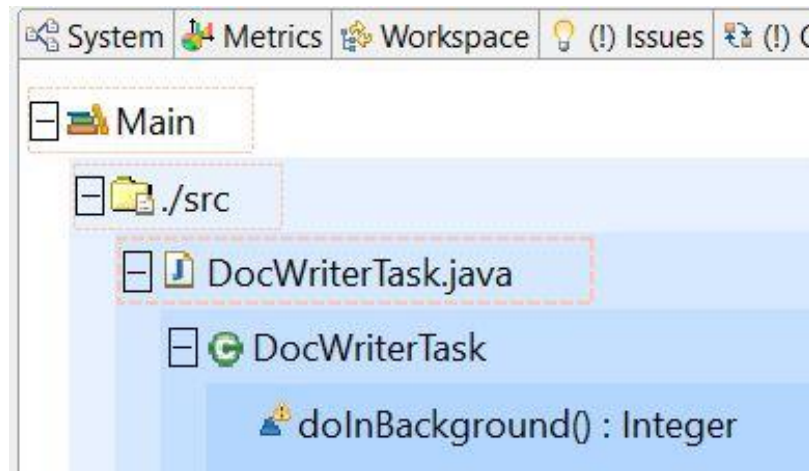


Figure 12

Figure 11 will demonstrate an issue which is Threshold violation. Based on figure 11, the method initialize is located in the class BrickMosaic. Moving on, based on figure 3, the number of statements allowed to be in this method is supposed to be from the range of 0 until 100. However, in this case, the number has exceeded the range since it is 192. The main reason which is causing this problem is having many lines of codes. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

Figure 11 will demonstrate an issue which is Threshold violation. Based on figure 11, the method actionPerformed is located in the class BrickMosaic. Moving on, based on figure 3, the number of statements allowed to be in this method is supposed to be from the range of 0 until 100. However, in this case, the number has exceeded the range since it is 120. The main reason which is causing this problem is having many lines of codes. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

Figure 11 will demonstrate an issue which is Threshold violation (Modified cyclomatic complexity). Based on figure 11, the method actionPerformed is located in the class BrickMosaic. Moving on, based on figure 3, the main problem, in this case, is allowing cyclomatic complexity more than required. What I mean by this is that the cyclomatic complexity (range) is from 0 to 15. However, it has exceeded the range since it is 24. That is, there are 24 lineary-independent execution ways. As a result, it will indicate that the code is required 24 different tests to be carried out for analyzing this method. This problem is for having enormous if statements which will reduce the code clarity and make it challenging to test it. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

Figure 12 will demonstrate an issue which is Threshold violation. Based on figure 12, the method `doInBackground` is located in the class `DocWriterTask`. Moving on, based on figure 3, the number of statements allowed to be in this method is supposed to be from the range of 0 until 100. However, in this case, the number has exceeded the range since it is 227. The main reason which is causing this problem is having many lines of codes. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

Figure 12 will demonstrate an issue which is Threshold violation (Modified cyclomatic complexity). Based on figure 12, the method `doInBackground` is located in the class `DocWriterTask`. Moving on, based on figure 3, the main problem, in this case, is allowing cyclomatic complexity more than required. What I mean by this is that the cyclomatic complexity (range) is from 0 to 15. However, it has exceeded the range since it is 20. That is, there are 20 lineary-independent execution ways. As a result, it will indicate that the code is required 20 different tests to be carried out for analyzing this method. This problem is for having enormous if statements which will reduce the code clarity and make it challenging to test it. Lastly, the severity of this issue is just a warning. That is, it will not cause a serious problem but by splitting the code between other methods, this problem can be solved, and it will result in simplified code.

The following figures will provide more information (in details) about the project.

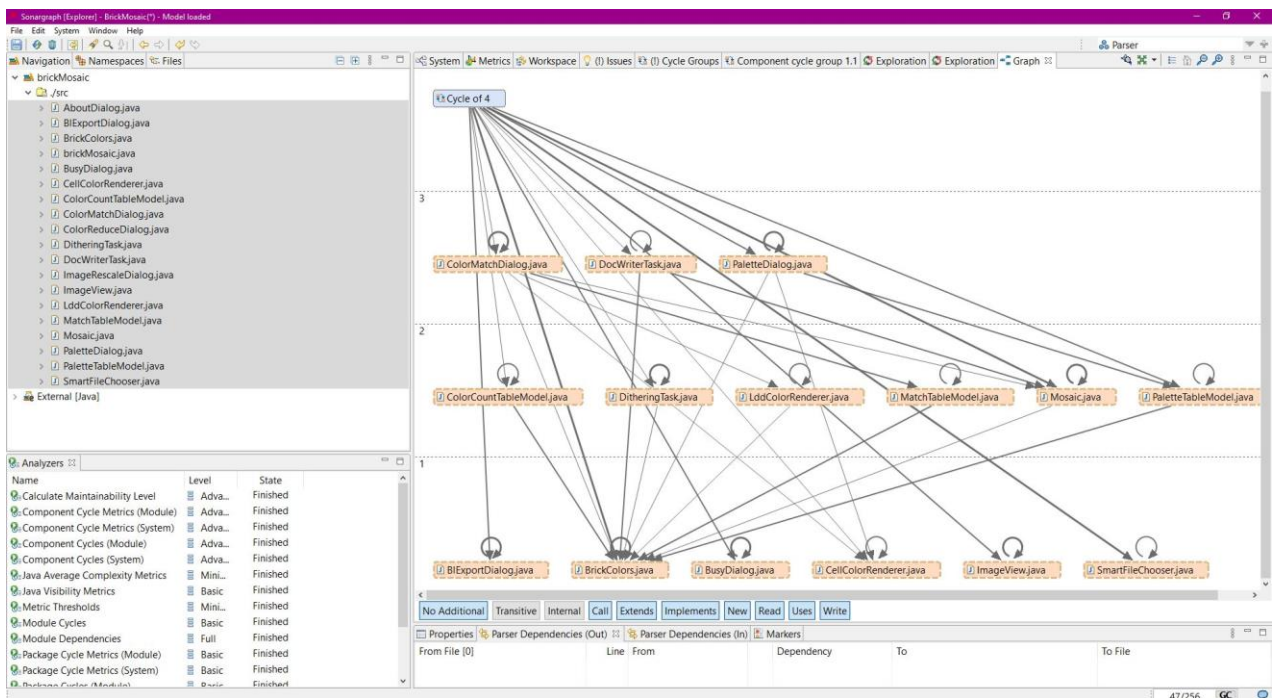


Figure 13

Level: Routine					Scope: BrickMosaic (System)	
Metric [7]	Categories	Provider	Min	Max	Values	Modified Cyclomatic Co...
Max Block Nesting D...	Code Analysis	Core			Element [147] (2 violations, 1.36%)	
Number of Parameters	Size	Core			brickMosaic.actionPerformed(ActionEvent) : void	24
Number of Statements	Size	Core	0	100	DocWriterTask.doInBackground() : Integer	20
Cyclomatic Complexity	Thomas J. Mc...	Core			ImageRescaleDialog.actionPerformed(ActionEvent) : void	15
Extended Cyclomatic ...	Thomas J. Mc...	Core			brickMosaic.readColors() : void	14
Modified Cyclomatic ...	Thomas J. Mc...	Core	0	15	DitheringTask.doInBackground() : BufferedImage	12
Modified Extended C...	Thomas J. Mc...	Core			Mosaic.colorCount() : int	10
					DitheringTask.errorFloydSteinberg(int,int,int,int,int,int) : int	9
					DitheringTask.errorDiffusion(int,int,int,int,int,int) : int	9
					brickMosaic.exportBIXml() : void	8
					ColorReduceDialog.actionPerformed(ActionEvent) : void	8
					ColorMatchDialog.actionPerformed(ActionEvent) : void	6
					ImageRescaleDialog.recalc() : void	6
					BusyDialog.actionPerformed(ActionEvent) : void	5
					CellColorRenderer.getTableCellRendererComponent(JTable,Object,boolean,boolean,int,int) ...	5
					ColorReduceDialog.doPalettImage(HashMap<Integer, Boolean>) : BufferedImage	5
					ImageView.paintComponent(Graphics) : void	5
					Mosaic.check() : boolean	5
					BrickColors.RGB2Lab(Color) : double[]	4
					BrickColors.getNearestColor(Color) : BrickColors	4
					BrickColors.getNearestColorFromPalette(Color, HashMap<Integer, BrickColors>) : BrickColo...	4
					brickMosaic.generateDocs() : void	4
					Arithmetic average: 2.33	Min. value: 1
					Standard deviation: 3.34	Max. value: 24
					Median: 1.00	
Properties Parser Dependencies (Out) Parser Dependencies (In) Markers						
Modified Cyclomatic Complexity (Routine)						
Categories	Thomas J. McCabe					
Description	As cyclomatic complexity but switch statements only add 1 independent from the number of cases.					
Fully Qualified Name	Metric Providers:Core Metrics:497982241					
Id	Core:Routine:ModifiedCcn					
Name	Modified Cyclomatic Complexity (Routine)					

Figure 14

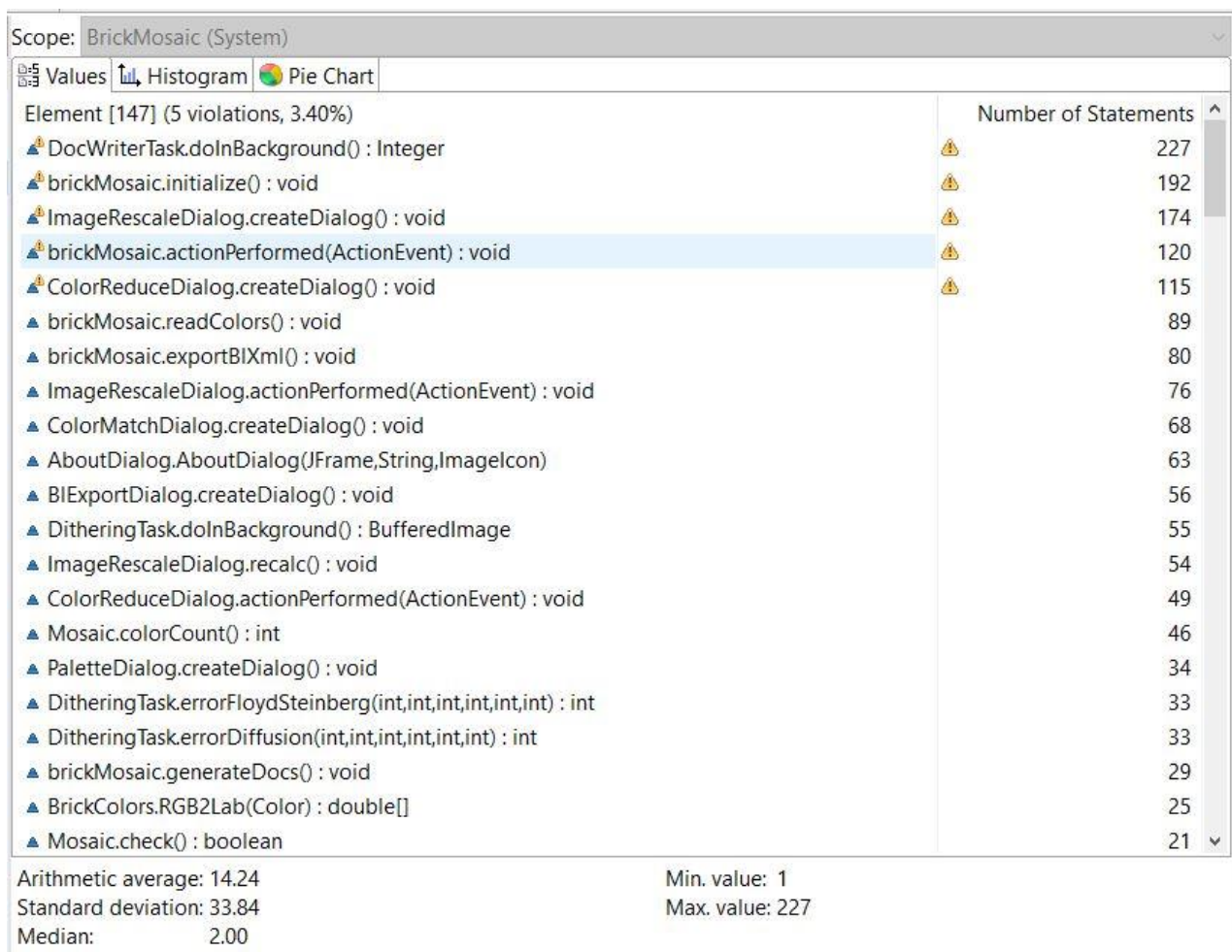


Figure 15

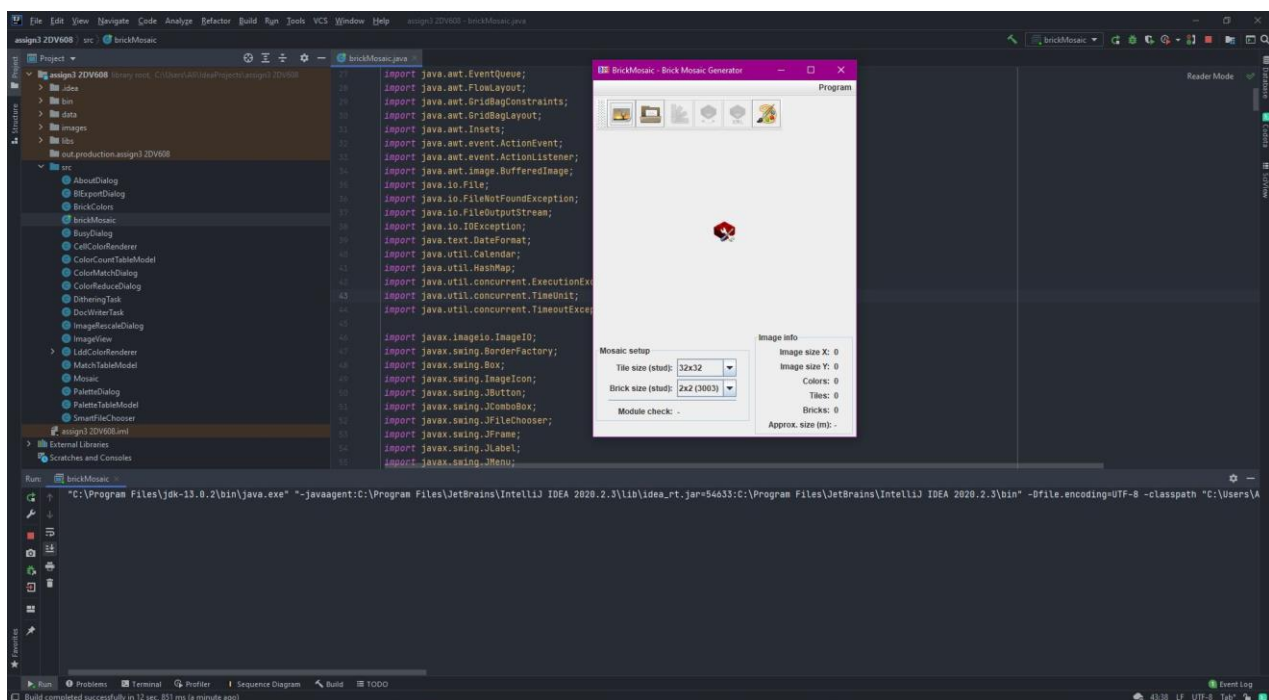


Figure 16

2. Task 2 – Re-engineering Plan

2.1. Resolve all issues found in Task 1

Removing first issue (cyclic dependency).

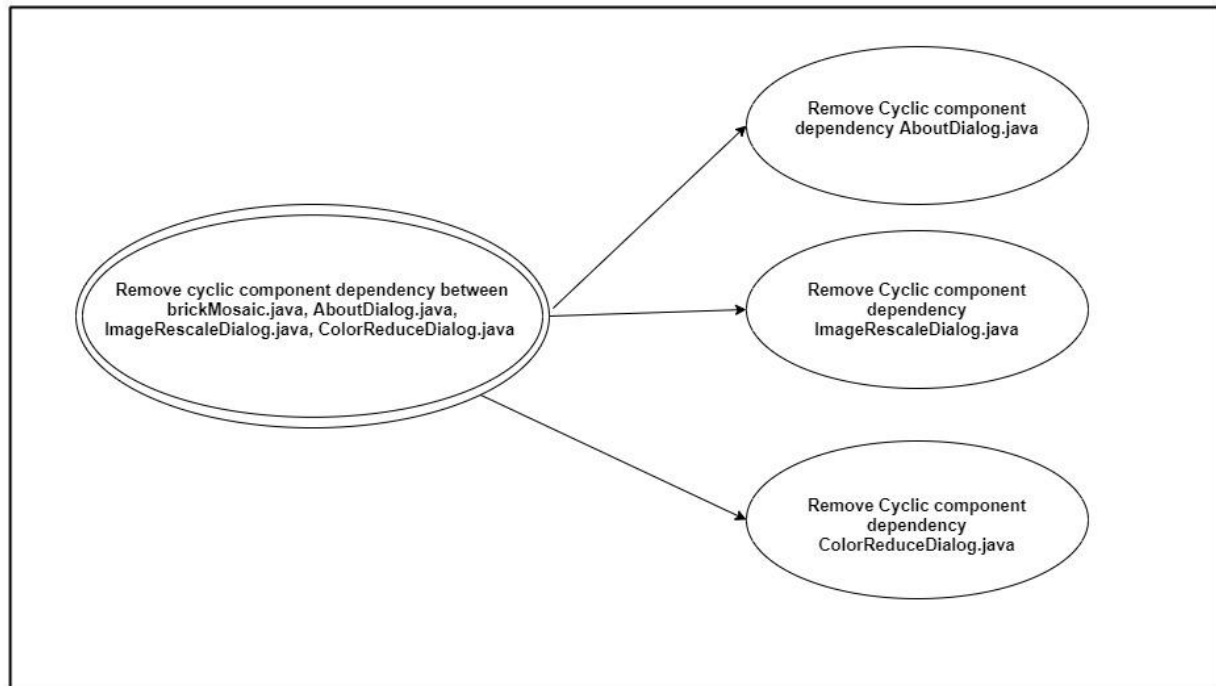


Figure 17

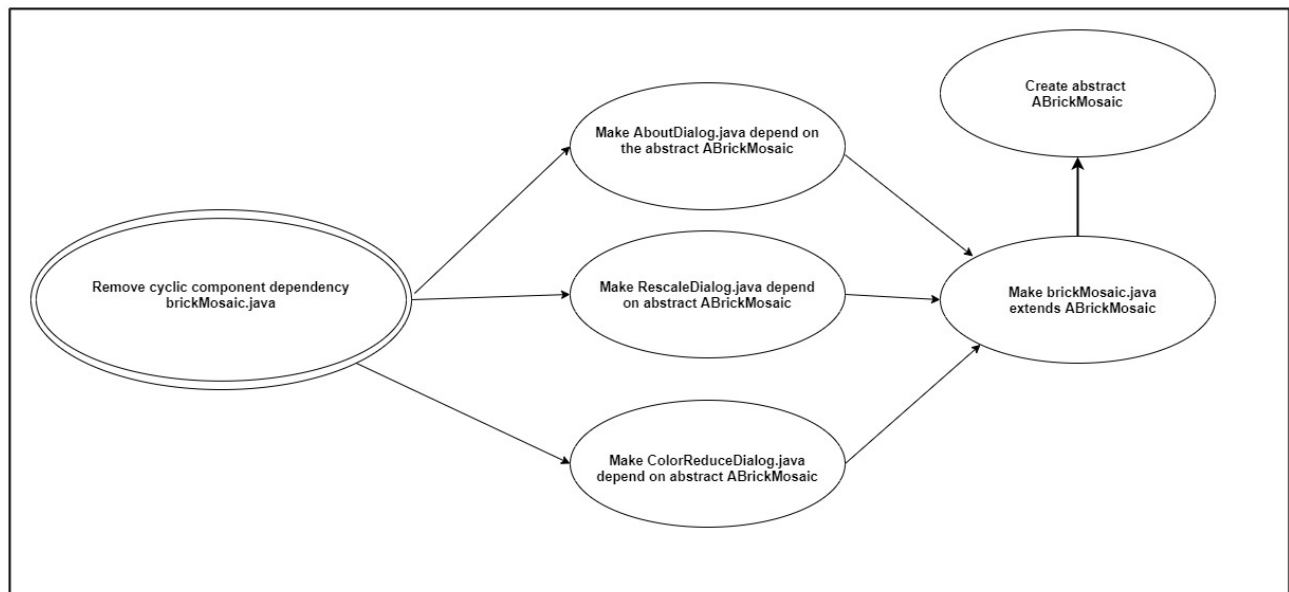


Figure 18

I have used the Mikado method to establish a plan to tackle the mentioned issues in figure 3. The information has been divided into two figures which are Figures 17 and 18. Moving on, the initial step for tackling and eliminating the cyclic dependency is by creating an abstract class. The reason I have chosen the abstract class over an interface class better resulted in system ACD. What I mean by this is that at the beginning I have approached this issue by creating an interface class of brickMosaic class. Consequently, I was able to fix the cyclic dependency issue. However, the system ACD result was not ideal at that point. As a result, I have changed my approach using an abstract class of the class brickMosaic. Undeniably, I was able to see the difference in the system ACD which is 3.40. Also, based on the provided lecture, the teacher has recommended that lower system ACD is better. That is why I have chosen abstract class over interface class to solve the cyclic dependency issue.

The number of 3 classes has been detected by the sonar graph application that is involved in the whole cyclic dependencies. To be more specific, classes AboutDialog.java, ImageRescaleDialog.java, and ColorReduceDialog.java are the ones that are involved in this issue.

Figure 18 is demonstrating the process of tackling this issue in more detail. What I mean by this is that an abstract class has been created from brickMosaic.java class. Therefore, the method's signatures from brickMosaic class have been added to the abstract class (ABrickMosaic.java). Consequently, the brickMosaic.java class will extend the abstract class. Therefore, by modifying the object of brickMosaic class in the mentioned classes (AboutDialog, ImageRescaleDialog, and ColorReduceDialog), the cyclic dependency issue will be solved because these classes are no longer dependent on the brickMosaic class.

All in all, to be able to tackle this issue, an abstract class will be created containing the necessary methods from brickMosaic.java class.

Removing Second issue (Threshold violation)

As it has been mentioned in part 1 (figure 3), 7 issues are about threshold violation which is either because of an extra number of statements or complexity (enormous if statements) which will make the code clarity and understanding challenging. To be able to tackle this issue, I have used the guidance of sonar graph application (issue section) to track down the problem sources. Consequently, I have noticed that the following methods should be modified.

Class	Method	Description
brickMosaic	actionPerformed(ActionEvent e)	Modified Cyclomatic complexity
brickMosaic	actionPerformed(ActionEvent e)	Number of Statements
brickMosaic	initialize (): void	Number of Statements
ColorReduceDialog	createDialog(): void	Number of Statements
ImageRescaleDialog	createDialog(): void	Number of Statements
DocWriterTask	doInBackground(): Integer	Modified Cyclomatic complexity
DocWriterTask	doInBackground(): Integer	Number of Statements

Moving on, I have created multiple methods for each of the above methods in the mentioned classes. What I mean by this is that I have divided the code and made it simpler by creating different methods. For instance, in case of having enormous if statements in just one method, I have inserted some of them into a new method to be able to solve both complexity and number of statements.

The below table will show what methods have been added to fix these issues.

Class	Method	Added method
brickMosaic	actionPerformed(ActionEvent e) & initialize (): void	initalizeHelper (): void, initalizeHelper1(): void, initalizeHelper2(): void, initalizeHelper3(): void, initalizeHelper4 (): void, initalizeHelper5(): void, conditions (ActionEvent e): void, conditions2(ActionEvent event): void conditions3(): void
ColorReduceDialog	createDialog(): void	createDialogHelper (): void
ImageRescaleDialog	createDialog(): void	dialogHelper (): void dialogHelper2 (): void
DocWriterTask	doInBackground(): Integer	doInBackgroundHelper (): void doInBackgroundHelper2 (): void doInBackgroundHelper3 (): void

Based on the above table, these methods in the added method section have a divided code from the original method which will result in code simplicity and solving the issues.

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

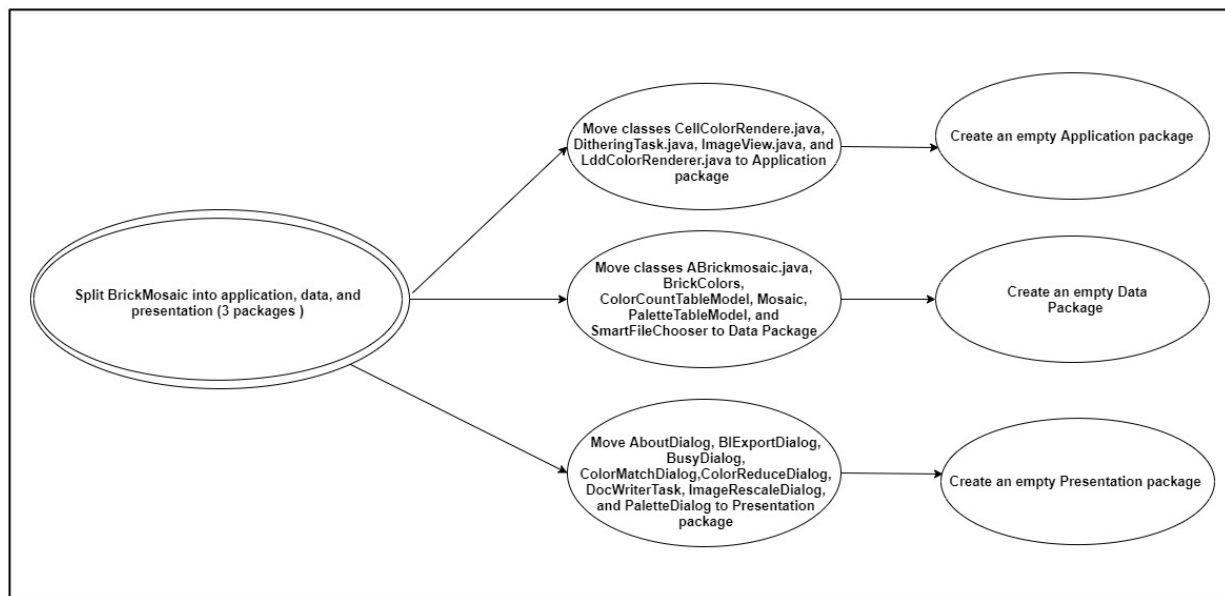


Figure 19

Figure 19 will demonstrate the multi-layer architectural pattern. That is, to approach this pattern, I have created 3 different packages named Application, Data, and Presentation. The main purpose behind this decision was to categorize the implementations based on their functionality. Consequently, the classes in this project have been transferred to the relevant packages. However, one of the classes is outside of the packages since it required more modification to make it execute properly and time shortage was the main problem that prevented me from fixing it. On the other hand, the rest of the classes have been added to the packages successfully and it was tested as well.

When it comes to the multi-layer architectural pattern, each layer can communicate with a layer underneath it quickly. Therefore, by using a multi-layer architectural pattern, it can be figured out that each layer has a well-built interface. That is, it will be used by the layer quickly above (API).

Regarding the complex system, it can be formed by superposing that are at increasing levels of abstraction.

3. Task 3 – Re-engineering

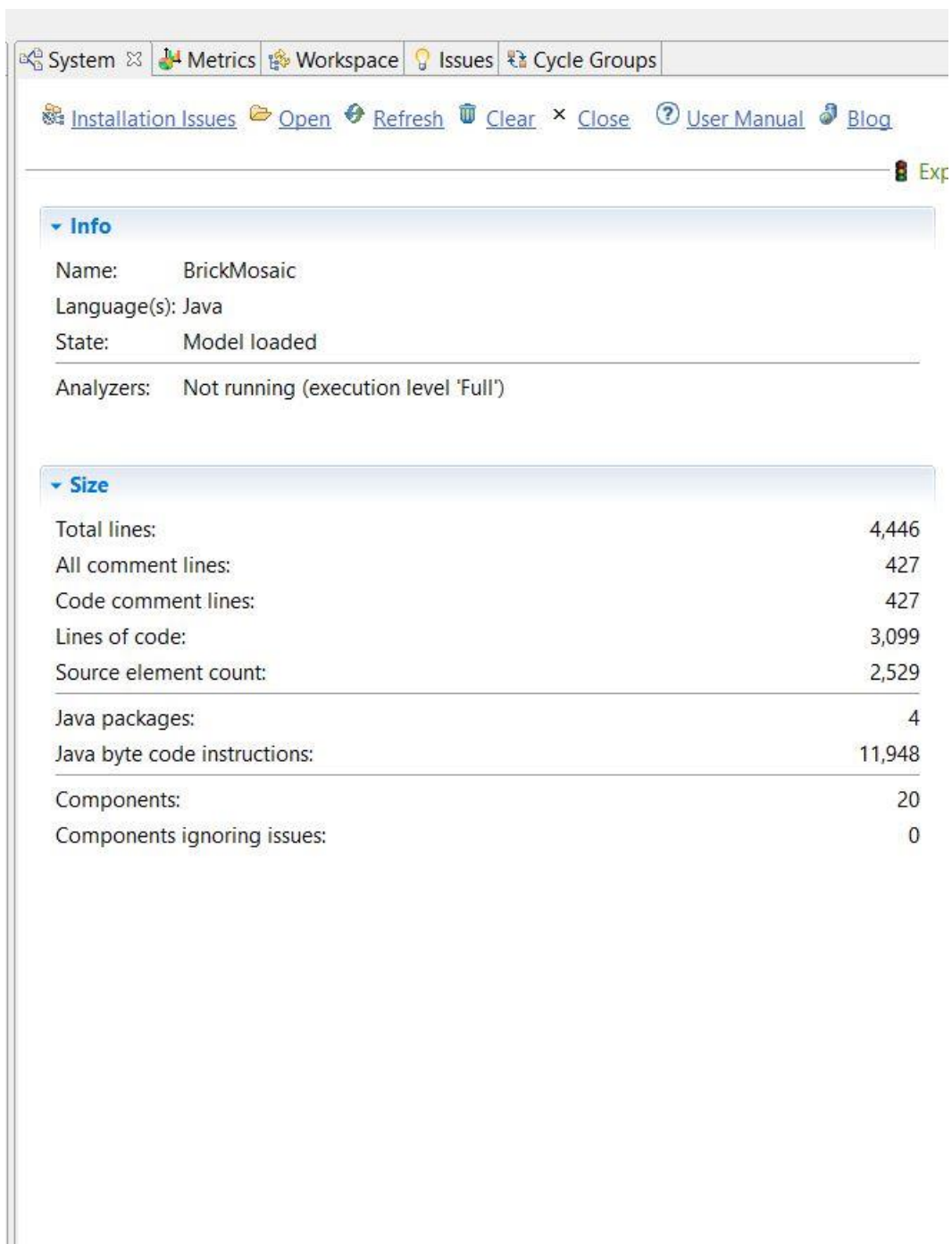


Figure 20

Figure 20 will indicate the first part of the system analysis. Consequently, a total number of 4,446 lines is existing. Consequently, 427 lines are considered all comments lines (it was 509) and 427 code comment lines. Regarding the java packages, only 4 packages have been detected. Lastly, the number of classes (components) is 20.

plorer

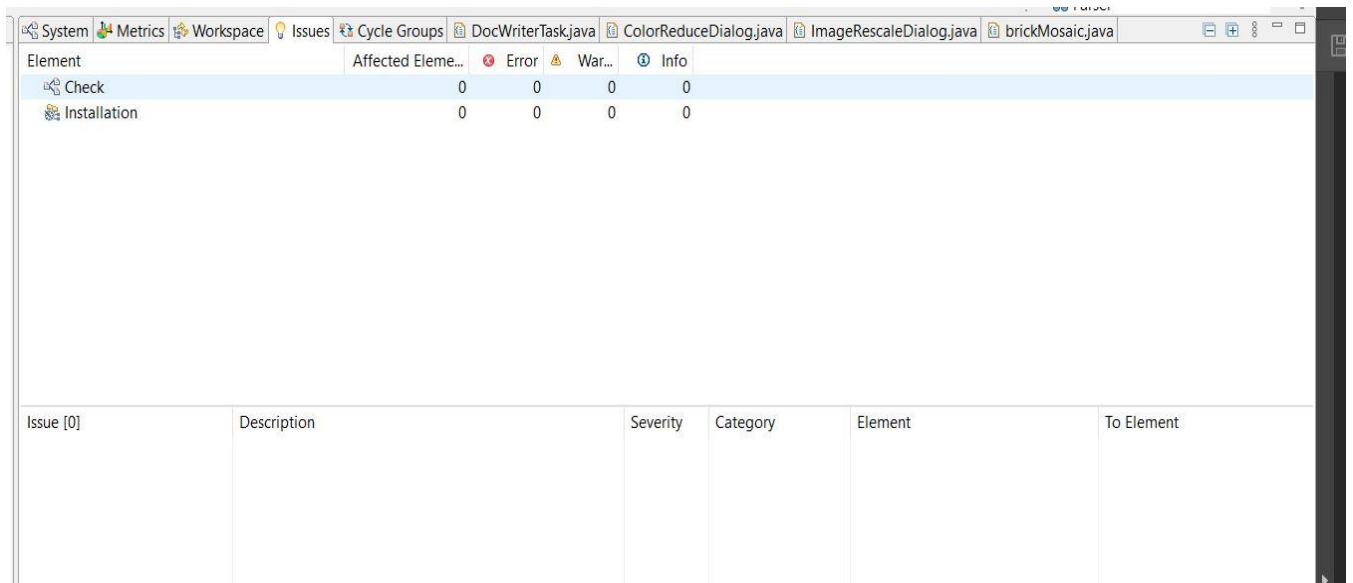
▼ Issues	
Total:	0
Configuration:	0
Workspace:	0
Threshold violations:	0
Cycle groups:	0
▼ Structure	
System Maintainability Level:	90.33
System ACD:	3.40
Highest module ACD:	3.40
Cyclic Java packages:	0
Structural debt index (Java packages):	0
Component dependencies to remove (Java packages):	0
Parser dependencies to remove (Java packages):	0
Cyclic components:	0
Structural debt index (components):	0
Component dependencies to remove (components):	0
Parser dependencies to remove (components):	0

Figure 21

Figure 21 will demonstrate the rest of the information belong to the system analysis. To begin, a piece of noticeable information in this figure would be the number of the issues which is 0. Moving on, it is undeniably clear that the maintainability level is quite high based on the size of the project which is 90.33. The ACD for the system has been saved as 3.40. What I mean by this is that each class is dependent on other classes with an average of 3.40. Also, there are 0 cyclic java packages. Therefore, the debt index would be 0. Noticeably, the number of dependencies would be 0 as well.

Moving on, in figure 20, the lines of the code has decreased from 3, 130 to 3,104 since the code has been simplified due to the Threshold violation issues. On the other hand, all comment lines are 508 now. Regarding the packages, the number of packages has increased from 1 to 4. Also, the number of components has increased by 1 as well since the abstract class has been added.

Regarding figure 21, a new remarkable fact about it is that there is no issue. What I mean by this is that all the issues have been fixed successfully. However, the system maintainability has remained the same. On the contrary, a huge change in system ACD can be seen because it is 3.40 now. Therefore, each class is dependent on other classes with an average of 3.40.



Element	Affected Eleme...	Error	War...	Info
Check	0	0	0	0
Installation	0	0	0	0

Issue [0]	Description	Severity	Category	Element	To Element
-----------	-------------	----------	----------	---------	------------

Figure 22

Figure 22 will prove that all the issues have been removed successfully. That is, all 8 issues have been eliminated. Also, the affected elements are 0.

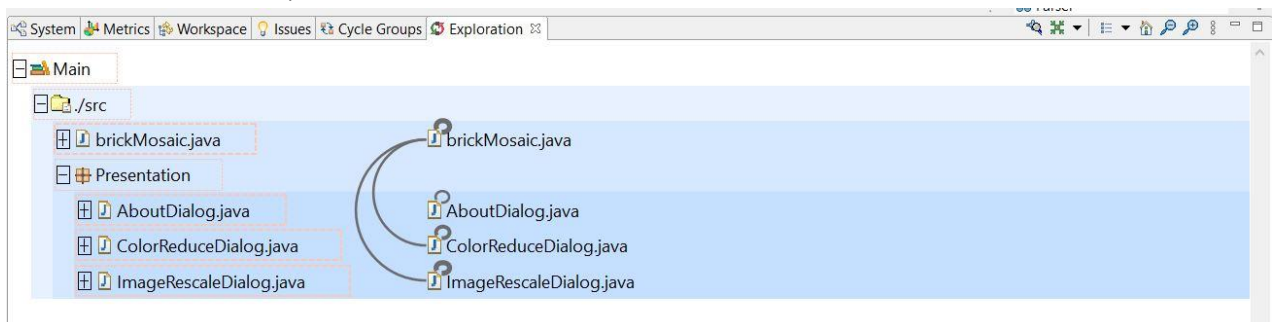


Figure 23

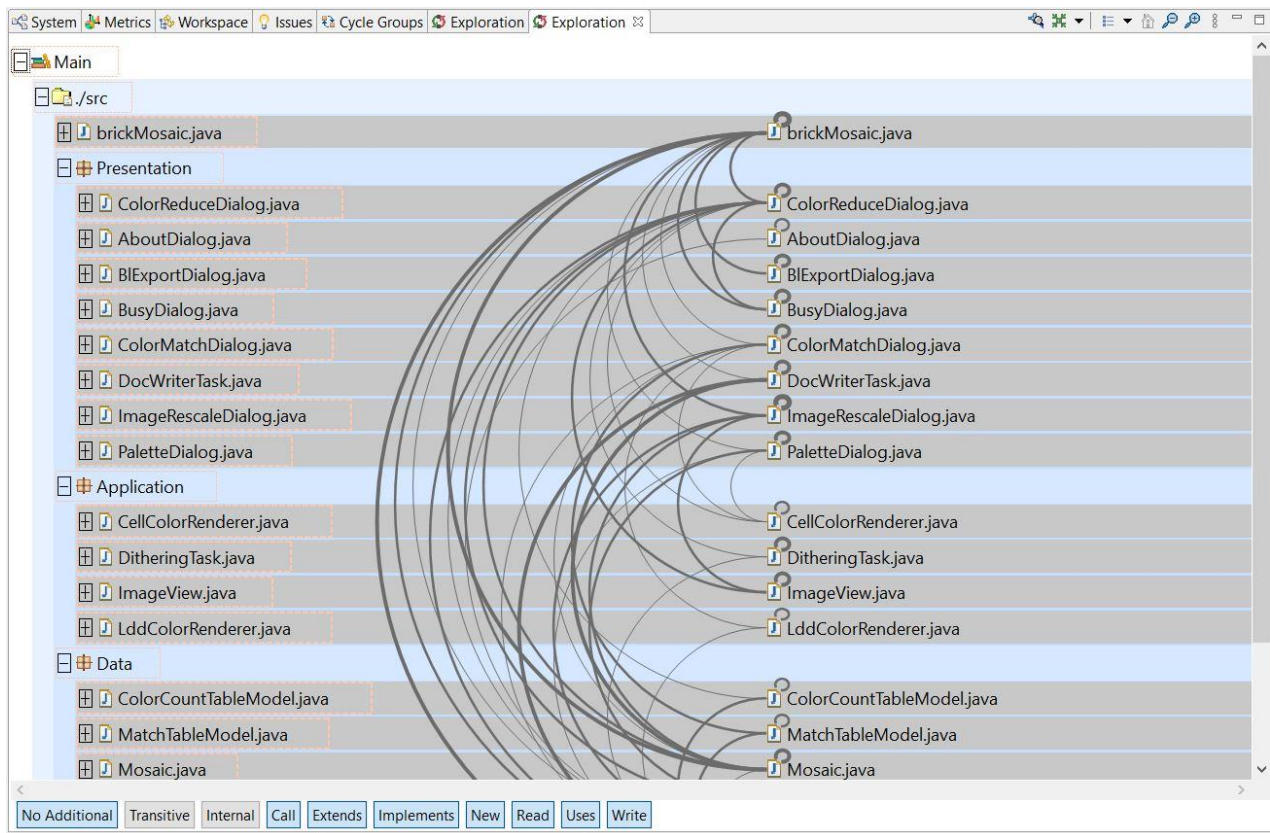


Figure 24

Based on figures 24 and 25, there is no upward going dependency (in exploration view) for the classes which have been detected for having a dependency. However, it undeniably will state that the dependency from the mentioned classes has been removed.

System Metrics Workspace Issues Cycle Groups				
Level: Routine				
Metric [7]	Categories	Provider	Min	Max
Max Block Nesting Depth	Code Analysis	Core		
Number of Parameters	Size	Core		
Number of Statements	Size	Core	0	100
Cyclomatic Complexity	Thomas J. McCabe	Core		
Extended Cyclomatic Complexity	Thomas J. McCabe	Core		
Modified Cyclomatic Complexity	Thomas J. McCabe	Core	0	15
Modified Extended Cyclomatic Complexity	Thomas J. McCabe	Core		

Figure 25

Figure 25 will indicate that there is no such class that can be referred to as a complex class which was a different situation at the beginning.