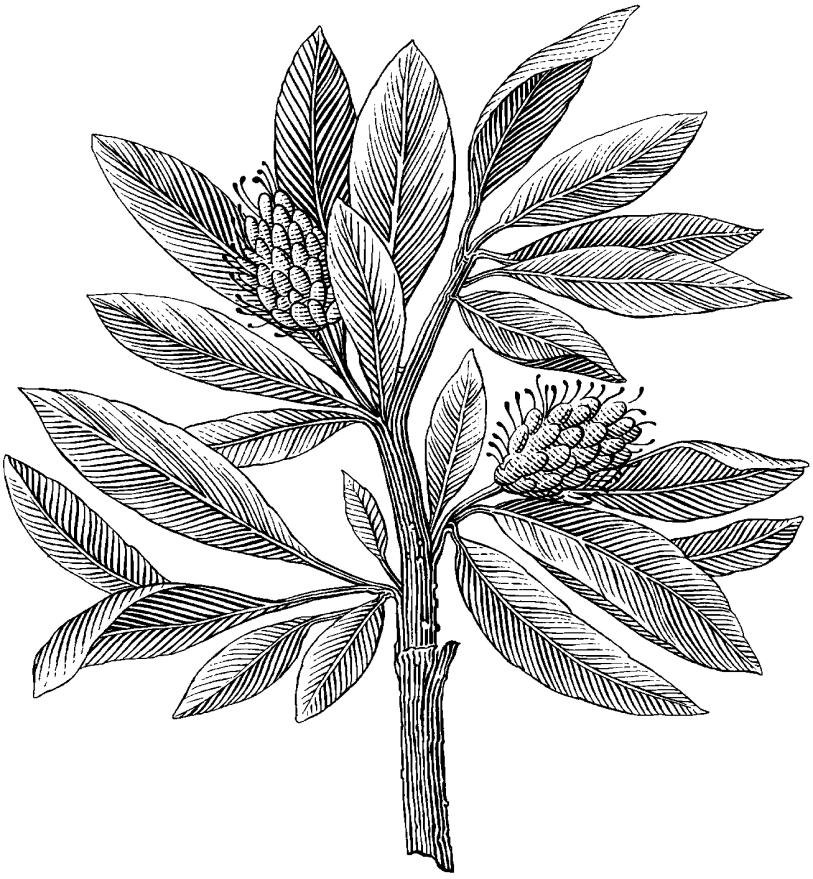
Linnéuniversitetet
Kalmar Växjö

Assignment 3

Architecting and Design

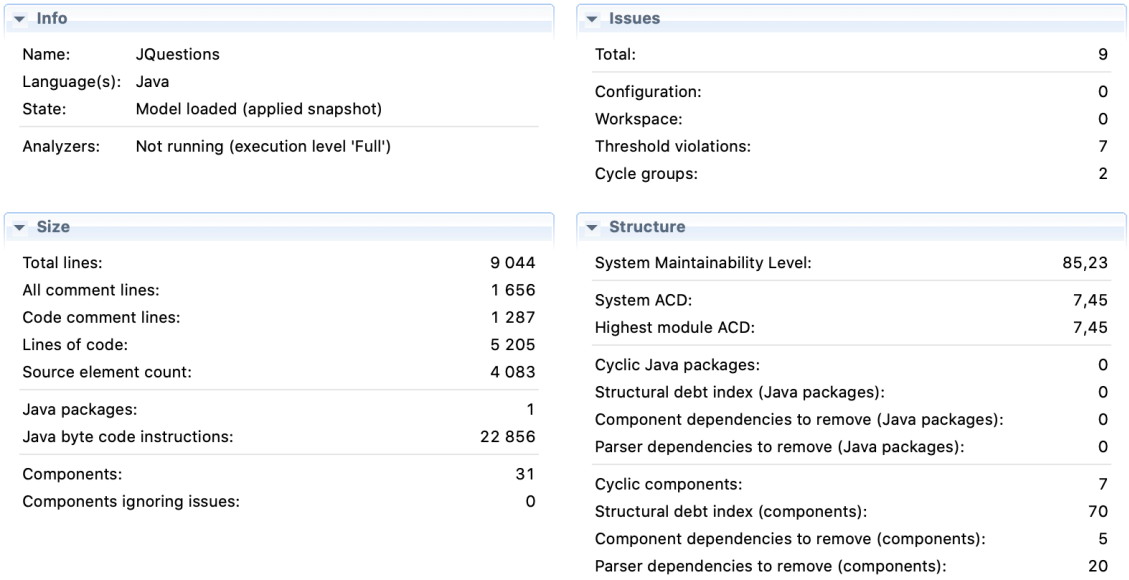
2DV608



Author: Omer Irfan
Semester: Spring 2021
Email: oi222ay@student.lnu.se

Table of Contents

1 Task 1 - Codebase Analysis



Info

Name:	JQuestions
Language(s):	Java
State:	Model loaded (applied snapshot)
Analyzers:	Not running (execution level 'Full')

Issues

Total:	9
Configuration:	0
Workspace:	0
Threshold violations:	7
Cycle groups:	2

Size

Total lines:	9 044
All comment lines:	1 656
Code comment lines:	1 287
Lines of code:	5 205
Source element count:	4 083
Java packages:	1
Java byte code instructions:	22 856
Components:	31
Components ignoring issues:	0

Structure

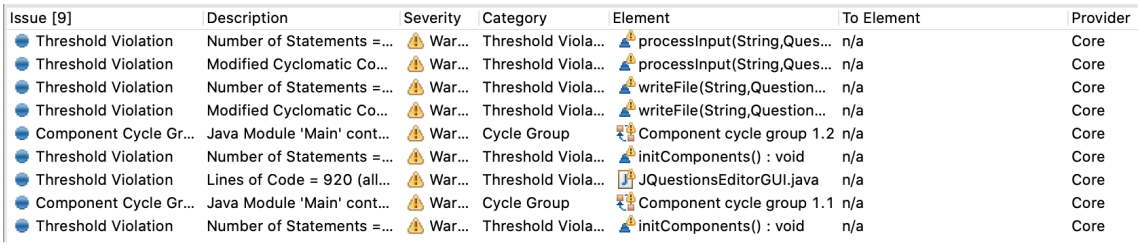
System Maintainability Level:	85,23
System ACD:	7,45
Highest module ACD:	7,45
Cyclic Java packages:	0
Structural debt index (Java packages):	0
Component dependencies to remove (Java packages):	0
Parser dependencies to remove (Java packages):	0
Cyclic components:	7
Structural debt index (components):	70
Component dependencies to remove (components):	5
Parser dependencies to remove (components):	20

Figure 1 System Analysis

The figure above provides some basic information on the size, structure and issues with the source code under analysis, JQuestions. The 'Size' section in Figure 1 shows that there are 5205 lines of code with one single package and 31 components (classes).

The structure in Figure 1 mentions the System ACD as 7.45. Hence, each class depend on around 7.45 other classes. There are 7 cyclic components which result in a structural debt index of 70. The number of component dependencies to remove are 5, this will be important during the nest two tasks.

1.1 Issues



Issue [9]	Description	Severity	Category	Element	To Element	Provider
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	processInput(String,Ques...	n/a	Core
Threshold Violation	Modified Cyclomatic Co...	War...	Threshold Viola...	processInput(String,Ques...	n/a	Core
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	writeFile(String,Question...	n/a	Core
Threshold Violation	Modified Cyclomatic Co...	War...	Threshold Viola...	writeFile(String,Question...	n/a	Core
Component Cycle Gr...	Java Module 'Main' cont...	War...	Cycle Group	Component cycle group 1.2	n/a	Core
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	initComponents() : void	n/a	Core
Threshold Violation	Lines of Code = 920 (all...	War...	Threshold Viola...	JQuestionsEditorGUI.java	n/a	Core
Component Cycle Gr...	Java Module 'Main' cont...	War...	Cycle Group	Component cycle group 1.1	n/a	Core
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	initComponents() : void	n/a	Core

Figure 2 Issues in System

Figure 2 lists all issues in the system. There are 9 issues, 7 of which are Threshold violations and 2 are Component cycle group issues. The severity of all these issues are only warnings. Threshold violations refer to issues within methods of a class which include an excess number of statements or increased complexity within methods. These are often caused by complex if-statements or loops.

IOUtils.java	2	0	4	0
IOUtils	2	0	4	0
processInput(String,QuestionPool...	1	0	2	0
writeFile(String,QuestionPool,bool...	1	0	2	0

Issue [9]	Description	Severity	Category	Element	To Element	Provider
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	processInput(String,Ques...	n/a	Core
Threshold Violation	Modified Cyclomatic Co...	War...	Threshold Viola...	processInput(String,Ques...	n/a	Core
Threshold Violation	Number of Statements =...	War...	Threshold Viola...	writeFile(String,Question...	n/a	Core
Threshold Violation	Modified Cyclomatic Co...	War...	Threshold Viola...	writeFile(String,Question...	n/a	Core

Figure 3 IOUtils.java issues

Four of the threshold violation issues lie in the IOUtils.java class. Specifically in the methods processInput() and writeFile(). For both these methods the number of statements allowed is 100 and the cyclomatic complexity allowed is 15, both these ranges are exceeded which results in the threshold violation. The issues are due to high complexity. This is likely caused by the huge scale of if-else statements in both methods. Not only due to such long if-else methods but also due to having multiple embedded loops within these statements. The same is for all the other Threshold Violations seen in Figure 2.

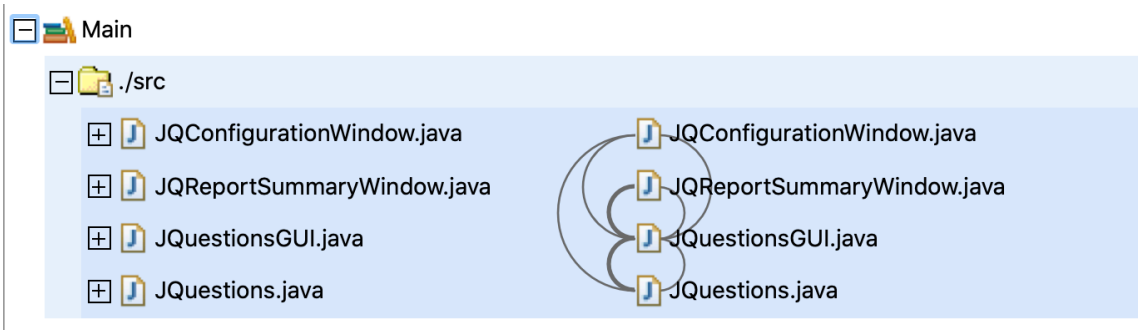


Figure 4 Component Cycle Group 1.1

Figure 4 shows one of the cycle groups which need to be fixed. The issue in this case high coupling. This is an architectural issue caused by dependencies between the classes shown in Figure 4. The same can be said of component cycle group 1.2 in Figure 5 below.

Figure 5 Component Cycle Group 1.2

Both cycles show high coupling which can be damaging for the architecture of a system. The arcs in Figures 4 and 5 indicate dependencies in both directions. To mitigate these cyclical dependencies single arc dependencies will be needed. Instead of having the dependencies go both ways, it should flow in one direction, in this case to the left. This will be done through re-engineering the code with Mikado's method. The main goal is to mitigate the cyclical dependencies.

2 Task 2 - Re-engineering Plan

I split my plan between the two cyclical groups and then further split those groups into two parts to make it easier to mitigate the dependencies at hand.

2.1 Cycle Group 1.1

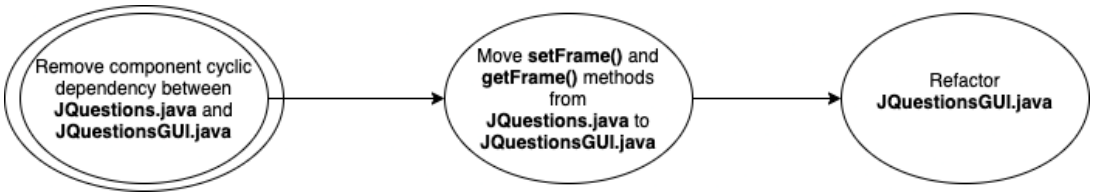


Figure 6 Cycle between JQuestions and JQuestionGUI

Both cycles have similar issues. They both include a few methods that can and should be moved from class A to class B. In this case those classes are presented in Figure 6 according to Mikado's method. This would maintain the functionality of the methods achieving not only low coupling but also high cohesion. Since the methods moved fit the GUI class better as that is what is manipulated.

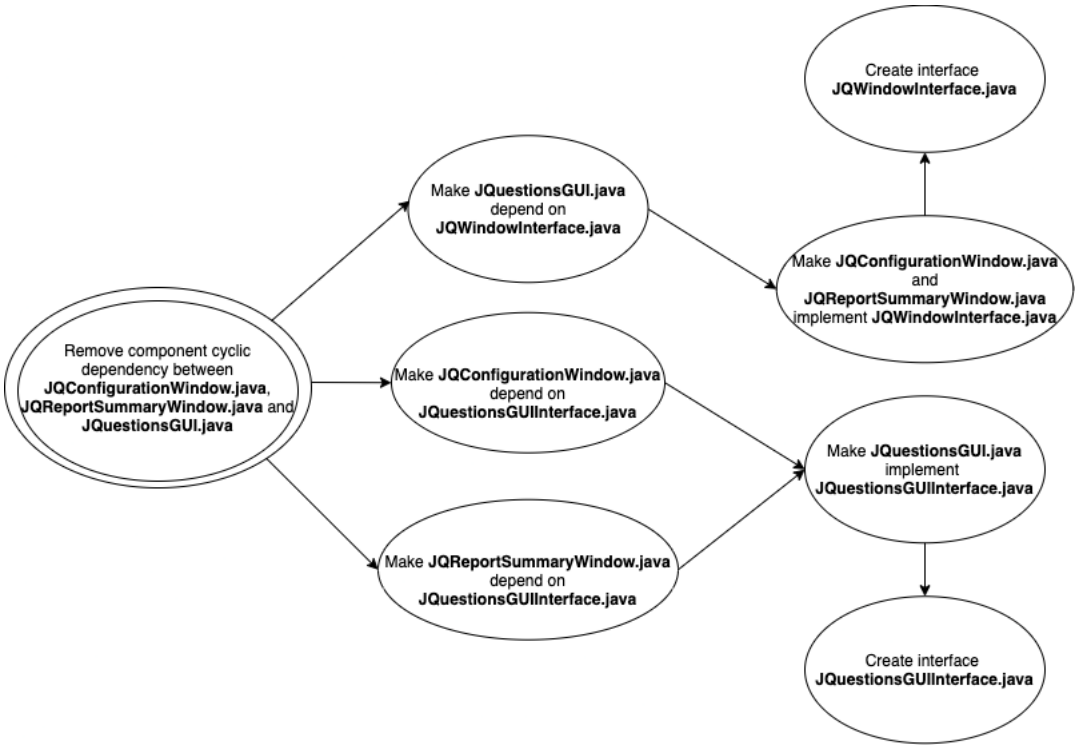


Figure 7 Creating Interfaces JQWindowInterface and JQuestionsGUIInterface

After the refactoring shown in Figure 6, I will mitigate the remaining dependencies of the cycle group. The plan can be seen in Figure 7. Since this is a closely coupled set of classes, they will require two interfaces for successful decoupling. The first will be for the two windows, JQConfigurationWindow and JQReportSystemWindow. This removes dependencies of JQuestionsGUI on the latter mentioned classes. After that I will create an interface for JQuestionsGUI to input into the window classes. This way they depend on the interface rather than the GUI object, finally diminishing the dependencies.

2.2 Cycle Group 1.2

Figure 8 Cycle between JQuestionsEditor and JQuestionEditorGUI

As mentioned in the previous section in Figure 8 I highlight the plan to refactor and move certain methods from the Editor class to its respective GUI class to achieve high cohesion and love coupling.

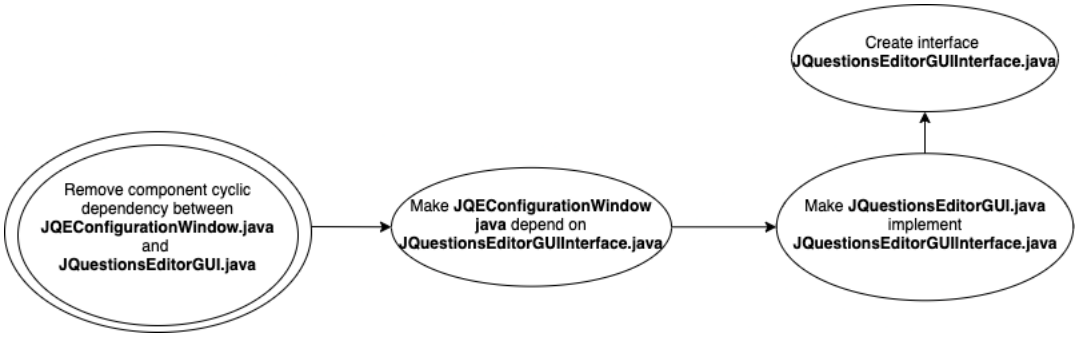
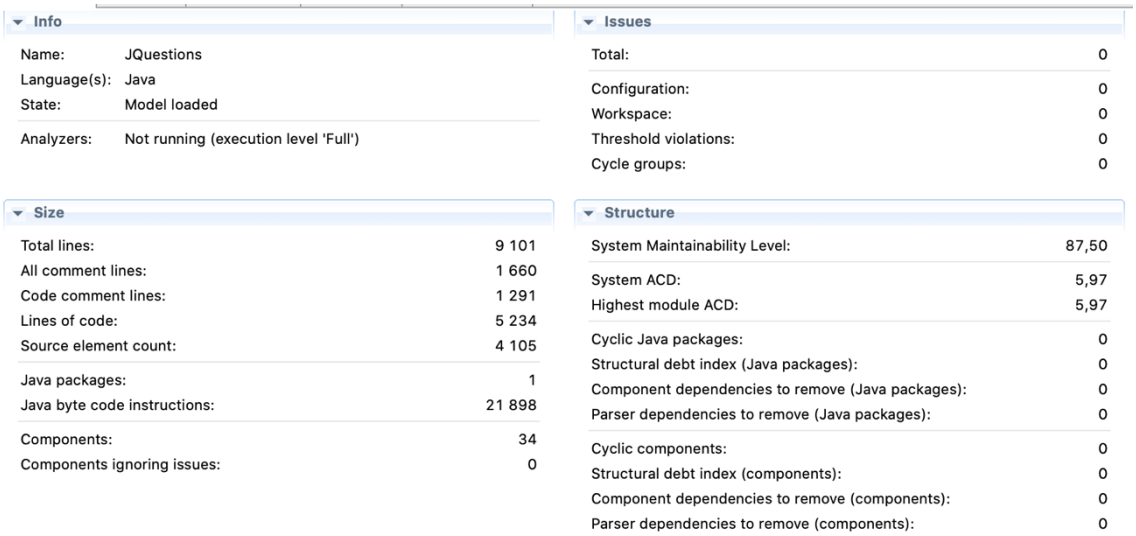


Figure 9 Creating Interface JQuestionsEditorGUIInterface

The second part of this cycle group also seems like a simple fix. I plan to create an interface for the GUI layer, as only a single method is in use by the ConfigurationWindow it should not cause too much trouble. That method will be included in the interface and implemented by the GUI (already contains said method).

3 Task 3 - Re-engineering



Info

Name:	JQuestions
Language(s):	Java
State:	Model loaded
Analyzers:	Not running (execution level 'Full')

Issues

Total:	0
Configuration:	0
Workspace:	0
Threshold violations:	0
Cycle groups:	0

Size

Total lines:	9 101
All comment lines:	1 660
Code comment lines:	1 291
Lines of code:	5 234
Source element count:	4 105
Java packages:	1
Java byte code instructions:	21 898
Components:	34
Components ignoring issues:	0

Structure

System Maintainability Level:	87,50
System ACD:	5,97
Highest module ACD:	5,97
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 10 Updated System Analysis

Figure 10 shows an updated system analysis of the size, issues and structure of the system after re-engineering according to the plan in Task 2. The most notable difference is the lack of issues.

The re-engineering of the cyclical dependencies resulted in their demise thus, zero issues. Another big difference from before is in the System ACD. It has decreased from 7.45 all the way to 5.97 meaning the coupling of the system of a whole has decreased. The figure also shows an increase in the components, this is of course due to the inclusion of interfaces within the system. The latter is also the reason for the increased lines of code.

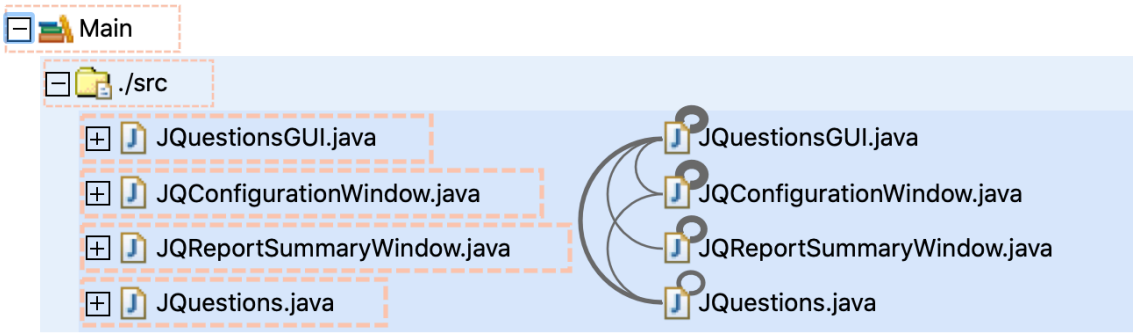


Figure 11 Cycle Group 1.1

As seen in Figure 11 all the component cyclical dependencies in group 1.1 have been mitigated after implementing the re-engineering plan. For the cyclical dependency between JQuestions and JQuestionsGUI the idea of creationg a separate interface was explored but only resulted in more cyclical dependencies and increased complexity. Thus, the planned approach was taken and successful.

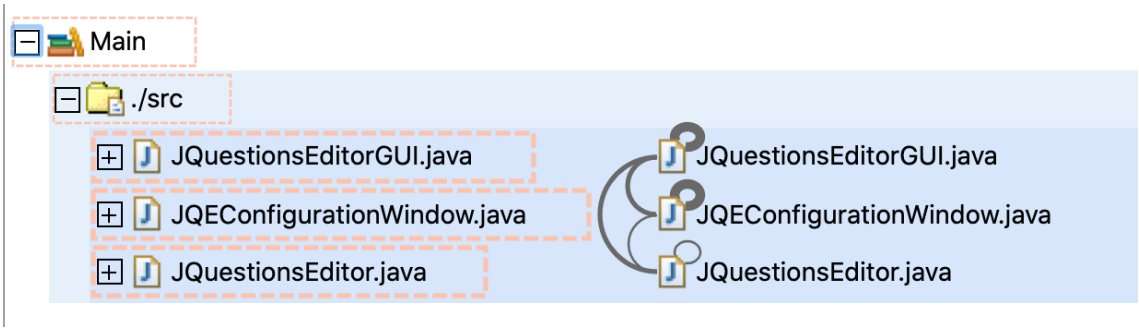


Figure 12 Cycle Group 1.2

Similar to cycle group 1.1, the same idea of a separate interface for the dependency between JQuestionsEditor and JQuestionesEditorGUI in group 1.2 was explored. It led to the same conclusion. However, the initial plan worked flawlessly as seen in Figure 12. All cyclical dependencies have been diminished reducing complexity and coupling. Thus, creating a more architecturally stable and strong system.