

Introduction

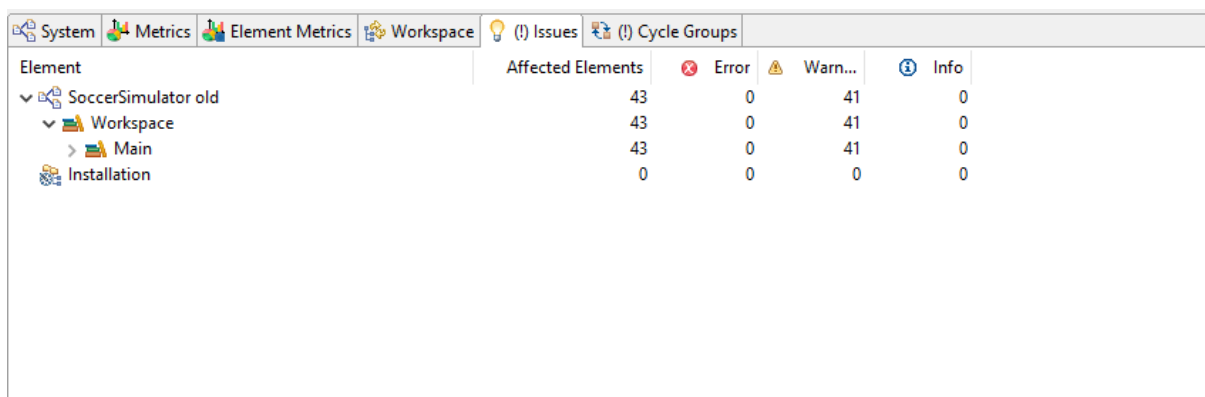
This report deals with the concepts of Architectural Design. The main focus is on solving the given problems by re-structuring a code/ project.

This process involves structuring the code using Model View Controller pattern (MVC) as well as using Multilayer Architectural pattern. This is done by creating a plan and applying the concepts of Design Principles and Architectural Patterns which were explained by the teacher in the lecture videos.

Task 1: Code analysis

- According to the given instructions, the source code ‘Soccer Simulator’ was imported and was run in the Sonargraph Explorer.
- An analysis was made in the code w.r.t the Java/Default Quality Model. The results that were obtained are presented below in figures 1,2 and 3.

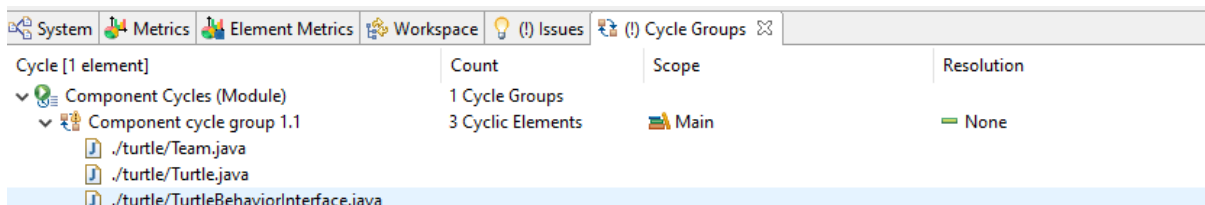
The figure below is from the issues section and we observe that there are 0 errors while the warnings are 41. Most of the warnings are workspace related. These errors occurred because the classes’ files or the code base are out-of-date.



System	Metrics	Element Metrics	Workspace	(!) Issues	(!) Cycle Groups
Element	Affected Elements	Error	Warn...	Info	
▼ SoccerSimulator old	43	0	41	0	
▼ Workspace	43	0	41	0	
> Main	43	0	41	0	
Installation	0	0	0	0	

Figure 1: Issues Section

In the figure below we inspected the cyclic groups sections and found that there was only one cyclic group warning. This is named as ‘Component cycle group 1.1’. The classes involved here are ‘Team.java, Turtle.java and TurtleBehaviourInterface.java’ from the ‘turtle’ package. This occurs due to cyclic dependencies among components which must be avoided.



System	Metrics	Element Metrics	Workspace	(!) Issues	(!) Cycle Groups
Cycle [1 element]		Count	Scope	Resolution	
▼ Component Cycles (Module)		1 Cycle Groups			
▼ Component cycle group 1.1		3 Cyclic Elements	Main	None	
./turtle/Team.java					
./turtle/Turtle.java					
./turtle/TurtleBehaviourInterface.java					

Figure 2: Cycle Groups section

In the figure below, it is a System view and it summaries the entire code base of the project by showing the project info, the issues, the number of cyclic dependencies and the other important features.

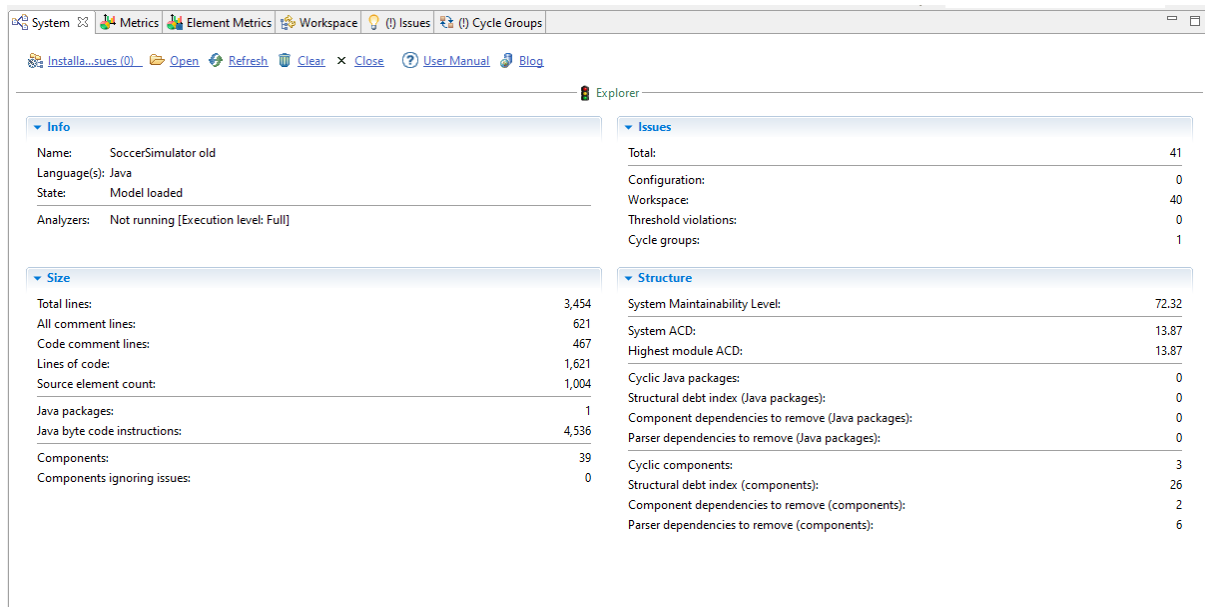


Figure 3: System section

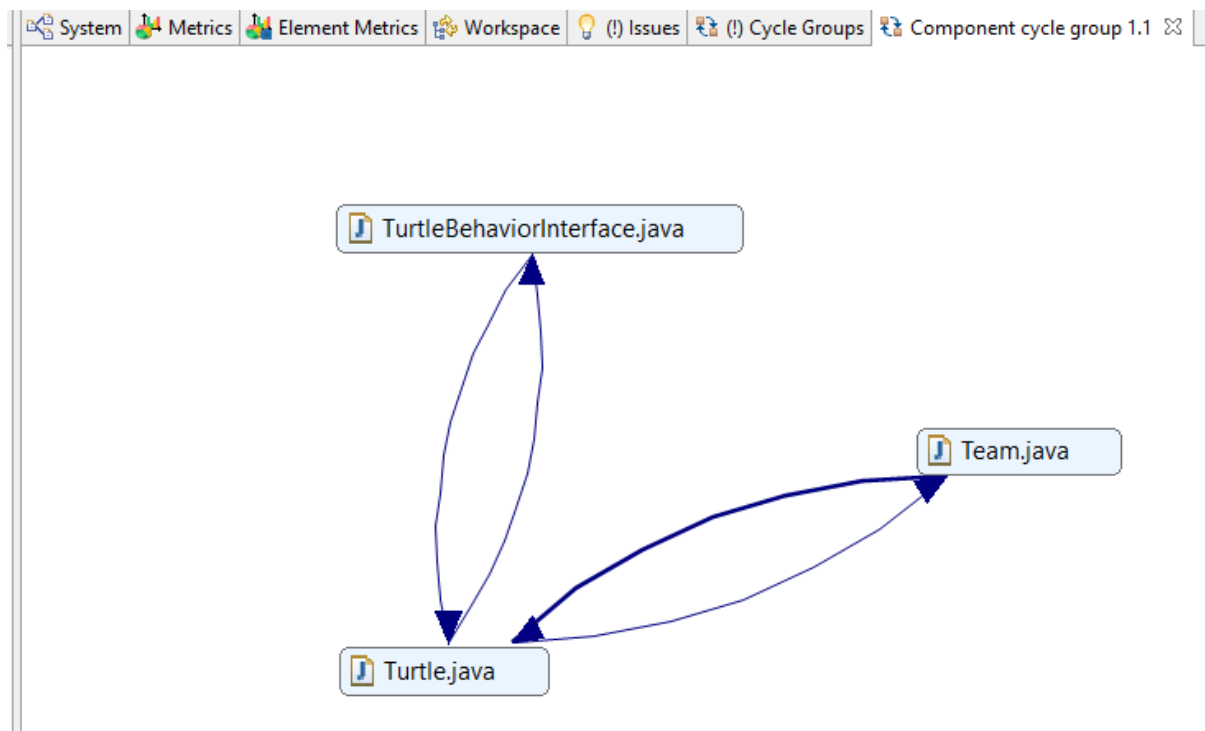


Figure 4: Graph View of Component Cycle group 1.1

From the above pictures we can conclude that the code contains only Warnings and no Errors. In figure 1, we observe that there are 41 issues.

The table below describes the issues in depth and also provides the mitigation strategy.

Total Issues [41]	Type of Issue	Description	Severity	Mitigation
40	Workspace	They occur due to internal errors when the workspace becomes corrupt and unreadable. This can happen when VM errors, such as OutOfMemoryError, Class File is out-of-date errors occur	Medium (Warning)	Class file out-of-date errors can be fixed by updating the directory of the classes.
1	Component Cyclic group	The cyclic dependencies must be avoided in the code as it causes understanding and maintainability issues as well as inhibits testing and code reuse.	Medium (Warning)	Reducing coupling among classes as this solves the problem of inter-dependencies

Task 2

A Re- engineering plan must be developed, and the code needs to be re-architected according to **Model View Controller Pattern** and **Multi-layer Architectural Pattern** to improve the overall Software Architecture to ensure easy readability and reusability of the code.

Strategy

Initially, the source contains only one package and that contained about 39 classes. To model the code in the MVC pattern, I used the Design Principle 1 which is Divide and Conquer. This ensures that the code is less complex as it is divided into smaller parts and therefore it is easy to understand. This also ensures code reusability wherever possible without having to replace or change other parts.

Fixing warnings and cyclic dependencies in code

- The Team class and 'Turtule' class has cyclic dependencies, to remove the dependency between them just move Turtle class to make it inner class inside Team class.
- Similarly, to remove the other dependency, delete 'setTurtule' method in the interface, then move the 'AbstractTurtleBehavior' to Turtle class as inner class, and then update the 'Turtule' constructor to make it receive 'AbstractTurtleBehavior' object instead of 'TurtleBehaviorinterface'.

Implementing the MVC pattern to the source code

This pattern consists of three packages namely; model, view and controller. According to a source from Wikipedia [1],

- Model: is the central component of the pattern as it manages the data, logic and rules of the application. It is independent of the user interface.
- View: represents the information such as chart, diagram or table.
- Controller: accepts input and converts it to commands for the model or view.

Therefore, the classes have been moved according to their functions to the model, view and controller packages. This scenario is shown in the figure below which implements the Mikado method.

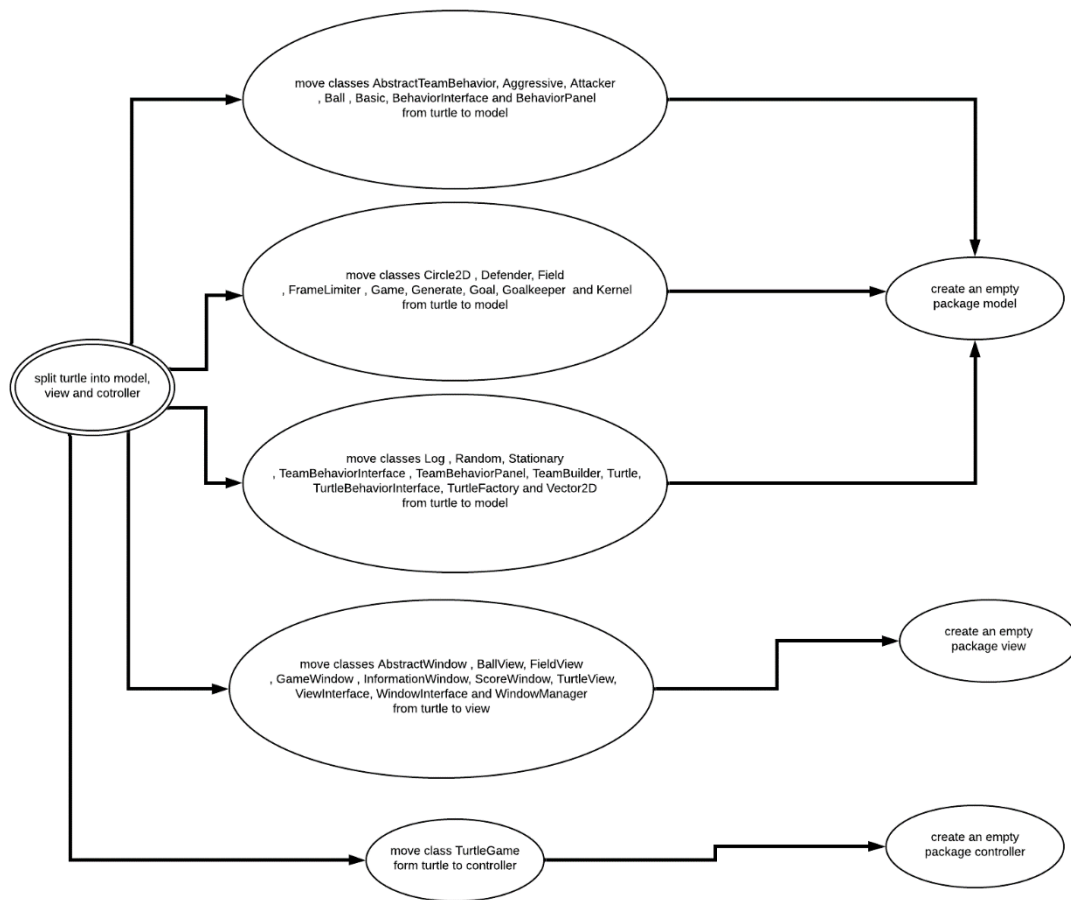


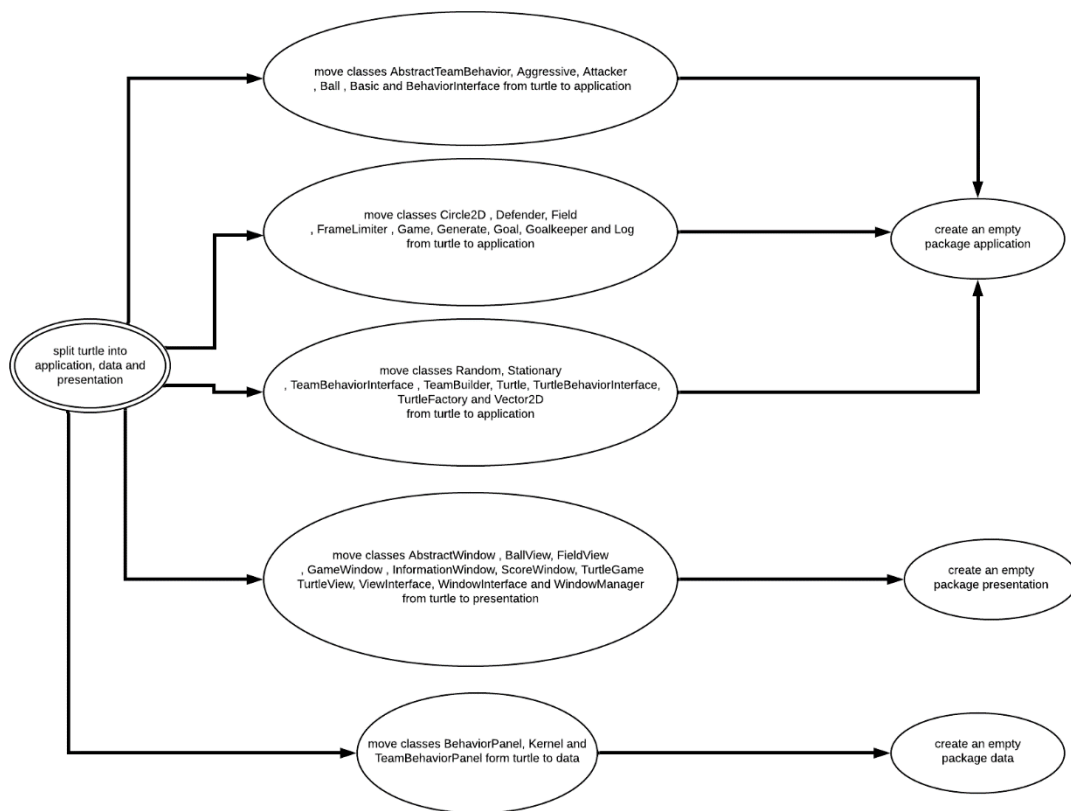
Figure 5: Mikado method for MVC

Implementing the Multi-layer Architectural Pattern to the source code

This pattern consists of three packages namely; application, data and presentation layer.

- Application: contains all the data, logic and rules of the application.
- Presentation: It contains all categories related to the presentation layer.
- Data: This is where all the data is stored.

Therefore, the classes have been moved according to their functions to the application, data and presentation packages. This scenario is shown in the figure below which implements the Mikado method.



Mikado method for Multi-layer Architectural Pattern

Task 3

The code was implemented according to the plan devised in Task 2. The warnings in the Component cycle group 1.1 is fixed. This can be seen in the results obtained below.

After implementing MVC pattern

System Metrics Element Metrics Workspace Issues					
Element	Affected Elements	Error	Warn...	Info	
Soccer-MVC	0	0	0	0	
Installation	0	0	0	0	

Figure 6: Issues Section for MVC

Explorer	
<div> <div>Info</div> <div> Name: Soccer-MVC Language(s): Java State: Model loaded <hr/> Analyzers: Not running [Execution level: Full] </div> </div>	
<div> <div>Size</div> <div> Total lines: 3,328 All comment lines: 584 Code comment lines: 438 Lines of code: 1,588 Source element count: 987 <hr/> Java packages: 3 Java byte code instructions: 4,396 <hr/> Components: 37 Components ignoring issues: 0 </div> </div>	
<div> <div>Issues</div> <div> Total: 0 Configuration: 0 Workspace: 0 Threshold violations: 0 Cycle groups: 0 </div> </div>	
<div> <div>Structure</div> <div> System Maintainability Level: 73.68 <hr/> System ACD: 12.78 Highest module ACD: 12.78 <hr/> Cyclic Java packages: 0 Structural debt index (Java packages): 0 Component dependencies to remove (Java packages): 0 Parser dependencies to remove (Java packages): 0 <hr/> Cyclic components: 0 Structural debt index (components): 0 Component dependencies to remove (components): 0 Parser dependencies to remove (components): 0 </div> </div>	

Figure 7: System section form MVC

After implementing Multi-layer Architectural Pattern

System	Metrics	Element Metrics	Workspace	Issues
Element				
Affected Elements		Error	Warn...	Info
multi layered		0	0	0
Installation		0	0	0

Figure 8: Issues Section for Multi-layer Architectural Pattern

Explorer	
<div> <div>Info</div> <div> Name: multi layered Language(s): Java State: Model loaded <hr/> Analyzers: Not running [Execution level: Full] </div> </div>	
<div> <div>Size</div> <div> Total lines: 3,355 All comment lines: 590 Code comment lines: 444 Lines of code: 1,597 Source element count: 979 <hr/> Java packages: 3 Java byte code instructions: 4,396 <hr/> Components: 37 Components ignoring issues: 0 </div> </div>	
<div> <div>Issues</div> <div> Total: 0 Configuration: 0 Workspace: 0 Threshold violations: 0 Cycle groups: 0 </div> </div>	
<div> <div>Structure</div> <div> System Maintainability Level: 73.68 <hr/> System ACD: 12.78 Highest module ACD: 12.78 <hr/> Cyclic Java packages: 0 Structural debt index (Java packages): 0 Component dependencies to remove (Java packages): 0 Parser dependencies to remove (Java packages): 0 <hr/> Cyclic components: 0 Structural debt index (components): 0 Component dependencies to remove (components): 0 Parser dependencies to remove (components): 0 </div> </div>	

Figure 9: System Section for Multi-layer Architectural Pattern

Compare the 2 architectures and the results obtained in SonarGraph

Property	old	MVC	Multi-layer
System maintainability level	72.32	73.68	73.68
System ACD	13.87	12.78	12.78
Cyclic components	3	0	0
Structural dept index (components)	26	0	0
Parser dependencies to remove (components)	6	0	0

The results obtained after implementing the MVC and Multi-layer patterns are the same. Both do not show any errors or warnings but in my opinion, MVC and Multi-layer pattern are different from each other because MVC allows coupling while the Layered architecture does not. In a Multi-layered architecture it allows message passing between the layers. Hence, both these patterns yield the same results. So, I think that MVC can be a better option when it comes to implementing the presentation layer of a project which uses a UI framework and the Multi-layer pattern can be used for other purposes like API and communication busses.

References

1. Wikipedia, 24 April 2020. Available at:
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>.
Accessed date: [2020-05-20]