Assignment 3
Architecting and Design

by Christoffer Lundström

March 21, 2020

Software Design - 2DV608
Examiner: Mauro Caporuscio
Linnaeus University

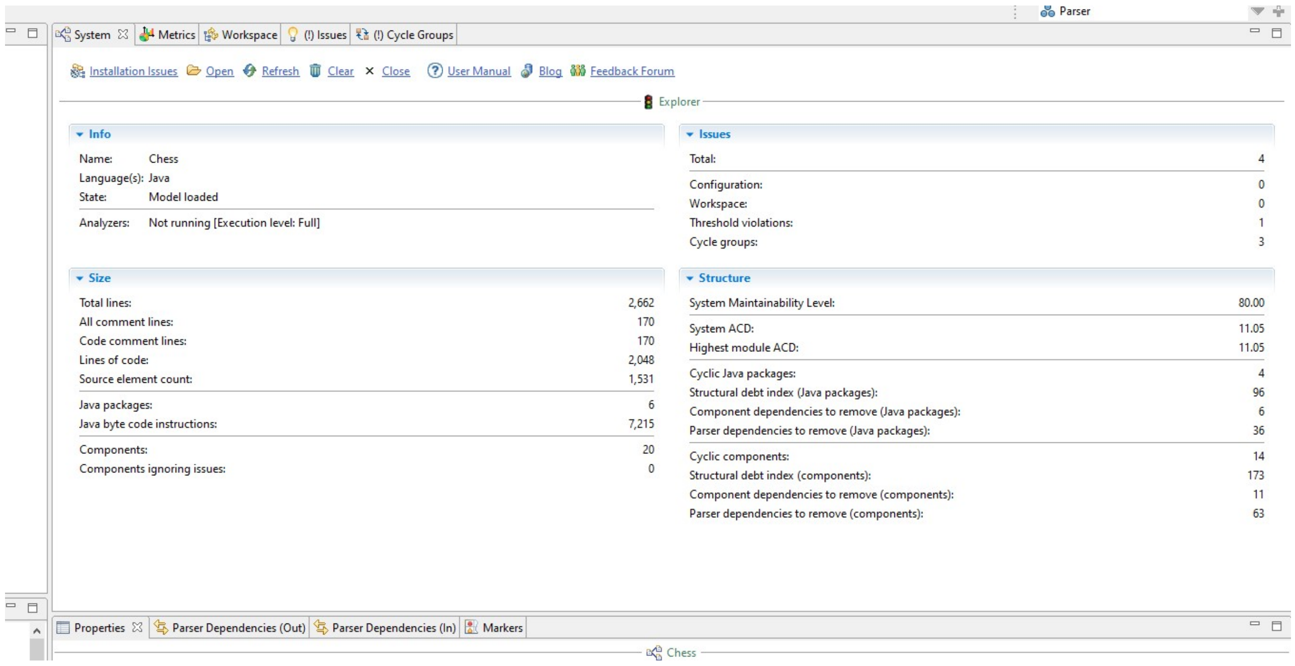
Contents

1 Introduction

This is a report intended to analyse issues in a legacy codebase and produce a re-engineering plan which will be the basis of a refactoring effort for the project.

2 Codebase Analysis

2.1 Static code analysis results



Info

Name:	Chess
Language(s):	Java
State:	Model loaded
Analyzers:	Not running [Execution level: Full]

Issues

Total:	4
Configuration:	0
Workspace:	0
Threshold violations:	1
Cycle groups:	3

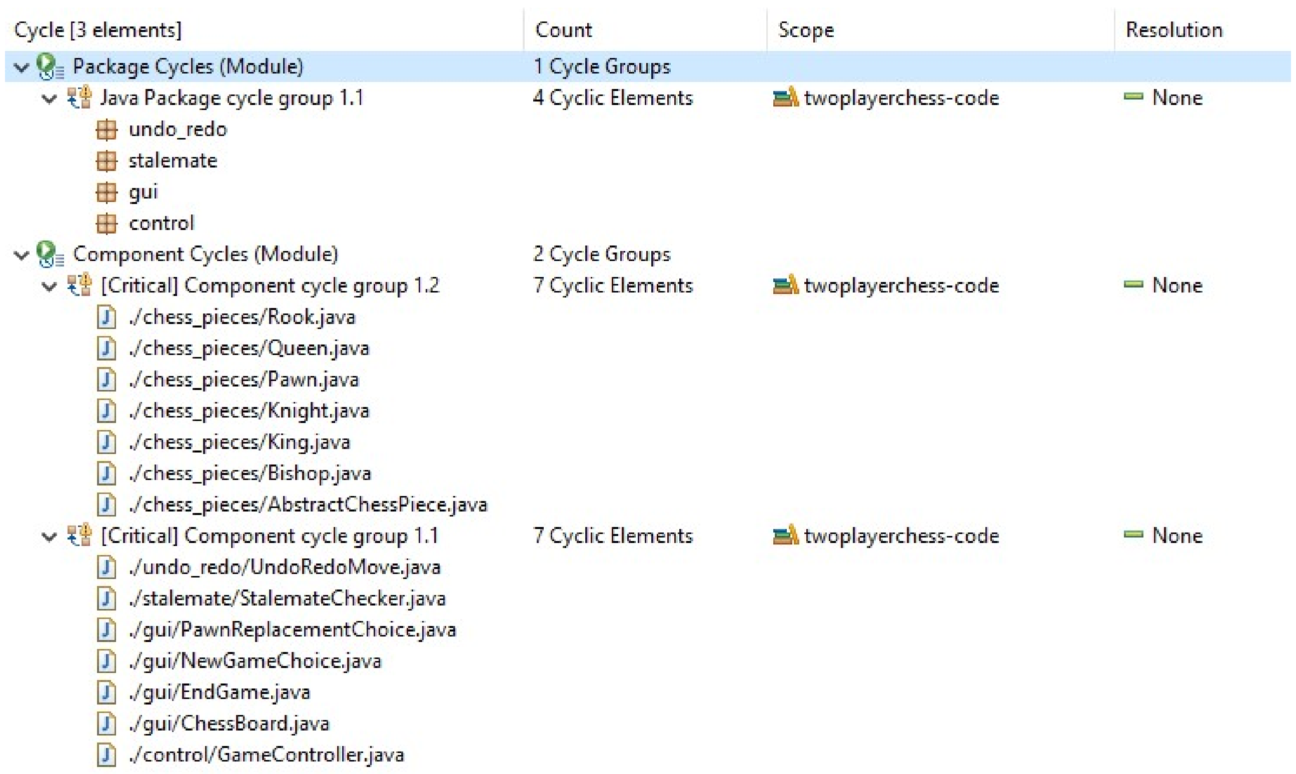
Size

Total lines:	2,662
All comment lines:	170
Code comment lines:	170
Lines of code:	2,048
Source element count:	1,531
Java packages:	6
Java byte code instructions:	7,215
Components:	20
Components ignoring issues:	0

Structure

System Maintainability Level:	80.00
System ACD:	11.05
Highest module ACD:	11.05
Cyclic Java packages:	4
Structural debt index (Java packages):	96
Component dependencies to remove (Java packages):	6
Parser dependencies to remove (Java packages):	36
Cyclic components:	14
Structural debt index (components):	173
Component dependencies to remove (components):	11
Parser dependencies to remove (components):	63

Figure 1: Results from Sonargraph Explorer.



Cycle [3 elements]	Count	Scope	Resolution
Package Cycles (Module)	1 Cycle Groups		
Java Package cycle group 1.1	4 Cyclic Elements	twoplayerchess-code	None
undo_redo			
stalemate			
gui			
control			
Component Cycles (Module)	2 Cycle Groups		
[Critical] Component cycle group 1.2	7 Cyclic Elements	twoplayerchess-code	None
./chess_pieces/Rook.java			
./chess_pieces/Queen.java			
./chess_pieces/Pawn.java			
./chess_pieces/Knight.java			
./chess_pieces/King.java			
./chess_pieces/Bishop.java			
./chess_pieces/AbstractChessPiece.java			
[Critical] Component cycle group 1.1	7 Cyclic Elements	twoplayerchess-code	None
./undo_redo/UndoRedoMove.java			
./stalemate/StalemateChecker.java			
./gui/PawnReplacementChoice.java			
./gui/NewGameChoice.java			
./gui/EndGame.java			
./gui/ChessBoard.java			
./control/GameController.java			

Figure 2: Cycle groups of Sonargraph Explorer.

2.2 Identified issues

2.2.1 Cyclic package dependencies

On package level there a few cyclic dependencies that needs to be addressed to reduce complexity and increase cohesion of the project.

Below I will briefly enumerate a list of cycles that together create cyclic dependencies across the project thus increasing complexity.

- *undo_redo* has a cycle with the *control* package.
- *stalemate* has a cycle with the *gui* package.
- *stalemate* has a cycle with the *control* package.
- *gui* has a cycle with the *control* package.

The only packages not cyclic dependent are *util* and *chess pieces*. The severity of these cycles are extreme. Any code modifications will propagate errors throughout the entire project and any new extensions or additions to the codebase will add to the technical debt and reduce overall re-usability and modularity.

To reduce cyclic dependencies between packages a general MVC pattern can be applied to sort application logic from view and controller logic. This will reduce complexity of the entire codebase and provide a higher level of cohesion for respective packages.

2.2.2 Cyclic component dependencies

This section will investigate the cyclic dependencies further by going down to class-level and find the causes of the dependencies. Some of them will vary in severity and the remedies will be presented in the next section.

1. Cycle group 1.1 (Figure 3) is caused by multiple interdependent components such as (StalemateChecker, GameController) or (Chessboard, PawnReplacementChoice). It is the result of poor architectural choices and a lack of interface segregation. Smaller classes ought to depend on an interface rather than a huge class like GameController. According to the dependency-inversion principle high level modules should not depend on low level modules. By depending on abstractions instead of direct classes the cycle can be broken.

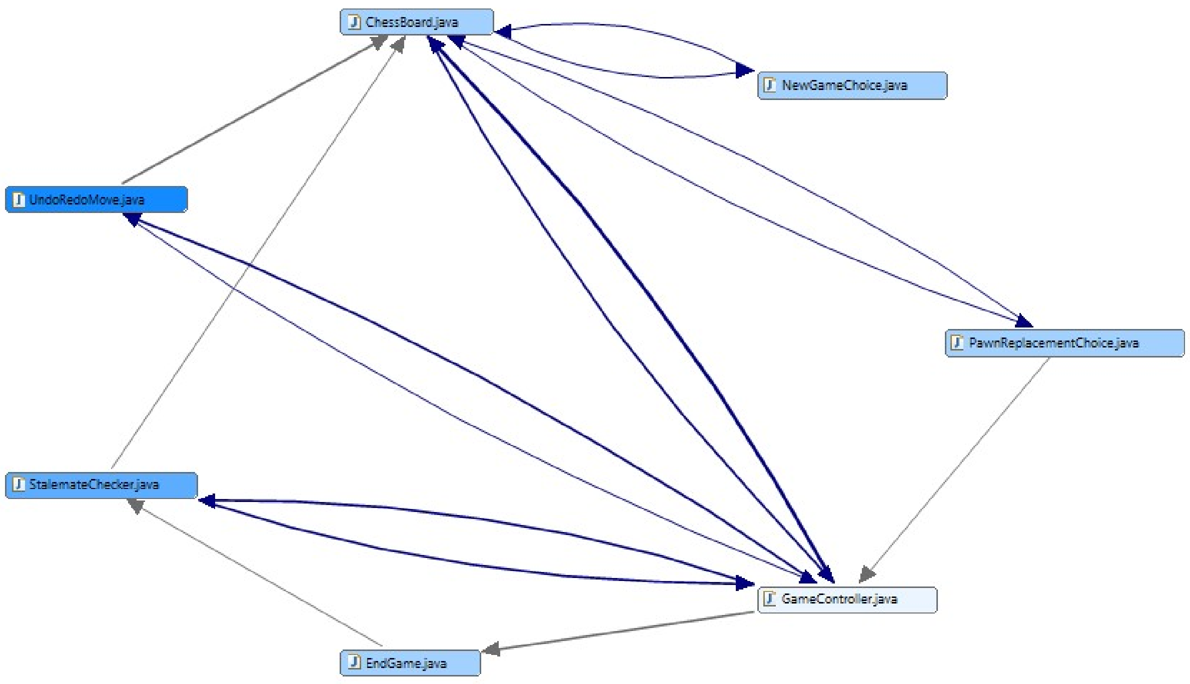


Figure 3: Cycle group 1.1

2. The root cause of the Cycle group 1.2 (Figure 2) is *AbstractChessPiece* which uses functions to create and clone instances of child components. This design makes the two classes interdependent and makes them tightly coupled. The group contains 6 classes that extend AbstractChessPiece which is a common code practice. Extending from a class naturally adds a dependency to the parent class. Abstractions however should not be dependent on its children. It is poor design and can in worst case scenarios create infinite recursion loops. One solution to this can be to instead use the factory pattern to generate instances of child components.

2.2.3 Other issues

3. Throughout the codebase multiple nested classes are found which are used by others. This reduces cohesion and increases complexity for imports. Commonly used classes should be de-nestled to provide more cohesion.

4. Commented code blocks, dead functions and constructors are found at multiple locations. This increases the administrative burden and might introduce bugs in future refactoring. It is also a waste of code lines and ought to be removed from the codebase. VCS will have a history of the code and thus having commented codeblocks is unnecessary. Class NewGameChoice can be merged with chessboard to remove cyclic dependency.

5. GUI is tightly coupled to logic classes and not easily replaced. This reduces modularity and re- usability. A much better solution is to abstract the GUI via interfaces. This will decouple application logic from a the view layer and allow for multiple implementations (for example different languages of the user interface) and break the interdependent relation.

6. Duplicate code found in UndoRedoMove. This is a common cause of bugs because a developer might forget to edit both pieces of code intended to do the same thing. Common practice is to use a function instead.

3 Re-engineering plan

This section will provide a plan to remove and remedy the identified issues discovered in the previous section. To fully eliminate cyclicity in the codebase a combination of package and component changes is made.

3.1 Package structure changes

Following changes are made to the codebase packages according to the Mikado method. These changes will provide the first step in eliminating package and component cycles. A clear structure of packaging facilitates cohesion of classes and reduces interdependencies between packages. The new package structure will allow for higher cohesion and easier implementation of the MVC.

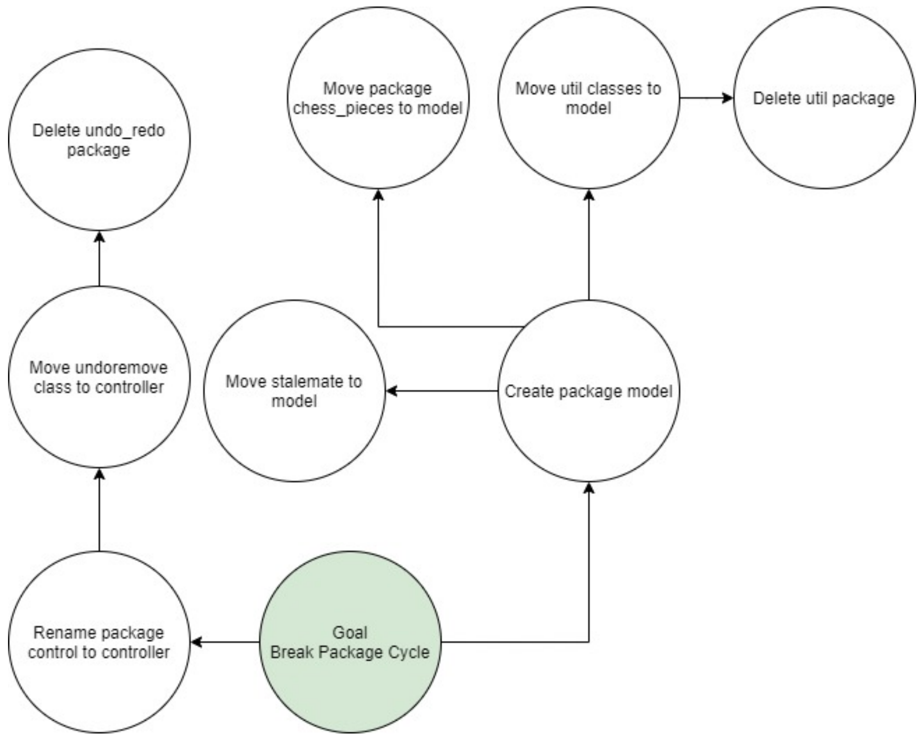

Delete undo_redo package
Move package chess_pieces to model
Move util classes to model
Delete util package
Move undoremove class to controller
Move stalemate to model
Create package model
Rename package control to controller
Goal Break Package Cycle

Figure 4: Steps of refactoring.

3.2 Component changes

This section will cover solutions to the numbered issues from section 2.2.2 Cyclic class dependencies and 2.2.3. Other issues.

1. After the initial package refactoring the next step of the top-down approach is to modify the components of each package so they adhere to MVC. The pattern states that model or logic components should be free from dependencies to other packages. A controller should be used to interconnect the view-layer with the model-layer.

By extracting interfaces (Figure 5) from the controllers and letting models depend on abstractions the models are free from controller dependency and the cyclic group is eliminated.

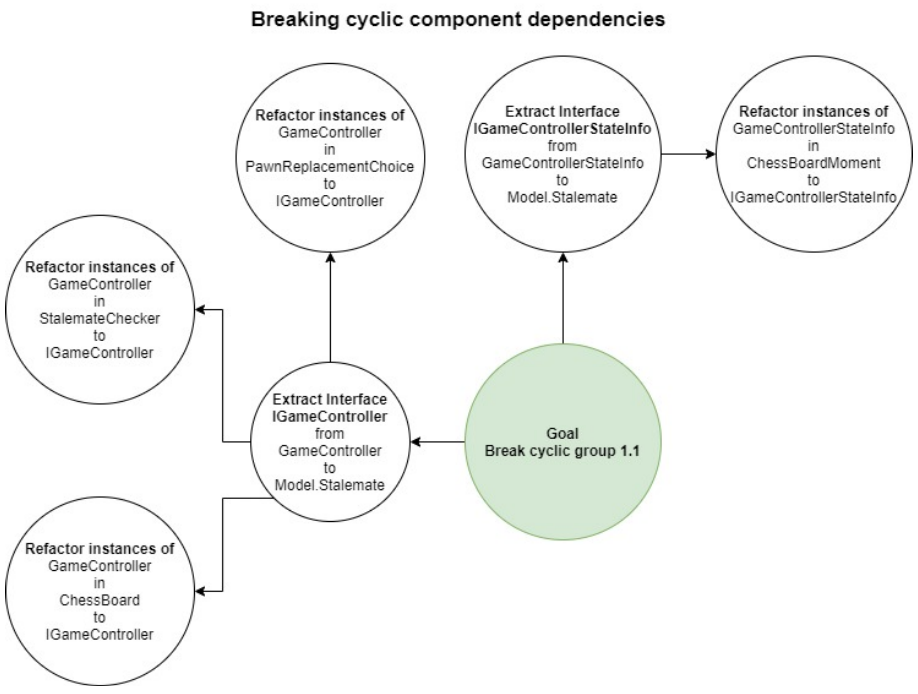

Breaking cyclic component dependencies
Goal
Break cyclic group 1.1
Extract Interface IGameController from GameController to Model.Stalemate
Refactor instances of GameController in PawnReplacementChoice to IGameController
Refactor instances of GameController in StalemateChecker to IGameController
Refactor instances of GameController in ChessBoard to IGameController
Extract Interface IGameControllerStateInfo from GameControllerStateInfo to Model.Stalemate
Refactor instances of GameControllerStateInfo in ChessBoardMoment to IGameControllerStateInfo

Figure 5: Breaking cyclic components (group 1.1).

2. The cyclic behavior of *AbstractChessPiece* can be remedied through use of the factory pattern (figure 6). This can be used remove the dependency from *AbstractChessPiece* to its children. Factory pattern is a good choice for this error because it solves the cyclic dependency and is a common Java pattern.

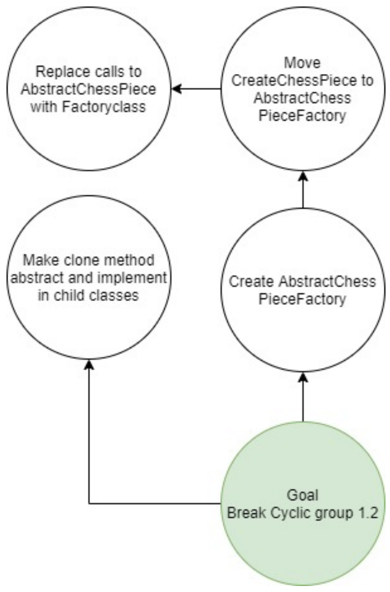

Replace calls to AbstractChessPiece with Factoryclass
Move CreateChessPiece to AbstractChess PieceFactory
Make clone method abstract and implement in child classes
Create AbstractChess PieceFactory
Goal
Break Cyclic group 1.2

Figure 6: Breaking cyclic AbstractChessPiece dependencies (group 1.2).

3.3 Other issues solutions

The big cyclic errors and their solutions are covered with previous Mikado graphs. Other issues 2.2.3 will however be briefly described below.

3. Move AbstractChessPiece.Colour to outer package. This allows for classes to depend on the outer package instead of AbstractChessPiece.

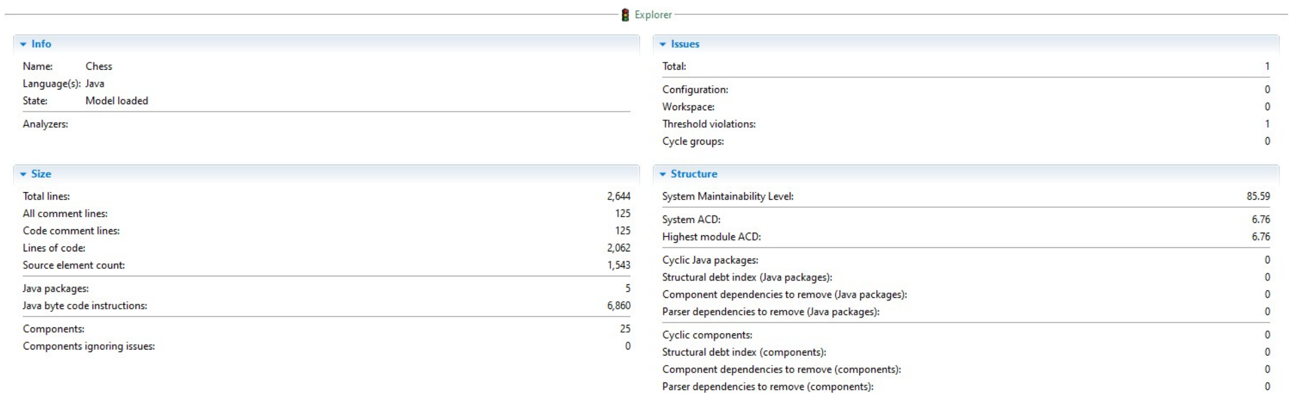
4. Dead code should be removed along with redundant classes. Cyclic component NewGameChoice should be eliminated by merging with chessboard class. This will remove the cyclic dependency with chessboard.

5. By extracting a chessboard interface more cyclic components (pawnreplacement) can be avoided.

6. Duplicate code in UndoRedoMove (updateVisualBoard) should be extracted to a function to avoid future bugs. It is easier to replace code in one place rather than two.

4 Refactoring results

This section will display results of the refactoring.



Explorer

Info

Name:	Chess
Language(s):	Java
State:	Model loaded
Analyzers:	

Size

Total lines:	2,644
All comment lines:	125
Code comment lines:	125
Lines of code:	2,062
Source element count:	1,543
Java packages:	5
Java byte code instructions:	6,860
Components:	25
Components ignoring issues:	0

Issues

Total:	1
Configuration:	0
Workspace:	0
Threshold violations:	1
Cycle groups:	0

Structure

System Maintainability Level:	85.59
System ACD:	6.76
Highest module ACD:	6.76
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 7: Final Sonargraph Explorer analysis.

Property	Before	After
System maintainability level	80.00	85.59
System ACD	11.05	6.76
Highest module ACD	11.05	6.76
Cycle Groups	3	0
Threshold violations	1	1
Cyclic packages	4	0
Structural debt index (packages)	96	0
Component dependencies to remove(packages)	6	0
Parser dependencies to remove (packages)	36	0
Cyclic components	14	0
Structural debt index (components)	176	0
Component dependencies to remove (components)	11	0
Parser dependencies to remove (components)	63	0

Table 1: Results of refactoring. Before and after.

4.0.1 Discussion

The results of the refactoring plan shows that the codebase has been significantly improved. All package and component cycles and structural debt has been completely cleared and the average complexity density is down by nearly 40% which is a substantial improvement. The chess game works like previously, but is now more re-usable, modular and has a higher cohesion because of the clear element separation through use of the model-view-controller, dependency-inversion principle and factory pattern.