Linnéuniversitetet
Kalmar Växjö

2DV608 Assignment 3 (AD)

Design for Quality
Architecting Software
Software Component Quality
Re-engineering Legacy Software

Author: Rashed Qazizada
Supervisor: Mauro Caporuscio
Semester: Spring 2020
Course name: Software Engineering Design

Linnéuniversitetet
Kalmar Växjö

Contents

1 Task 1 Codebase Analysis

1.1 Downloaded the file *twoplayerchess-code.zip*
1.2 Imported the source code in *Sonargraph Explorer*
1.3 Analysed the codebase w.r.t the Java/Default Quality Model and show the results (i.e., screenshot).
1.4 For each identified issue, I have discussed and mitigated my solution:

- The type of issue (e.g., coupling, cohesion, complexity, …) and its severity
- The source of the issue (e.g., architecture, classes, methods, code…)
- How it should be mitigated/solved (e.g., architectural pattern, design pattern, …), and the rationale.

1.4.1 Successfully executed

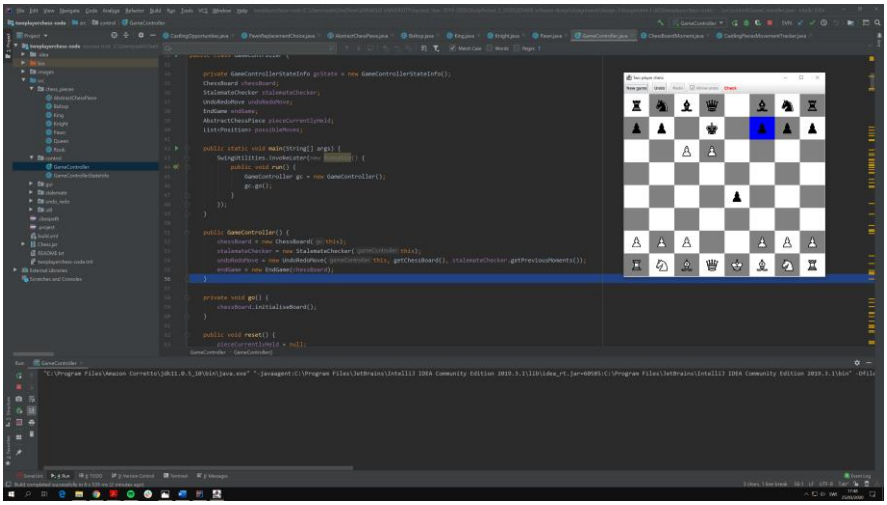


Figure 1: twoplayerchess
Source: Assignment 3 (AD) twoplayerchess-code

1.4.2 Component dependencies

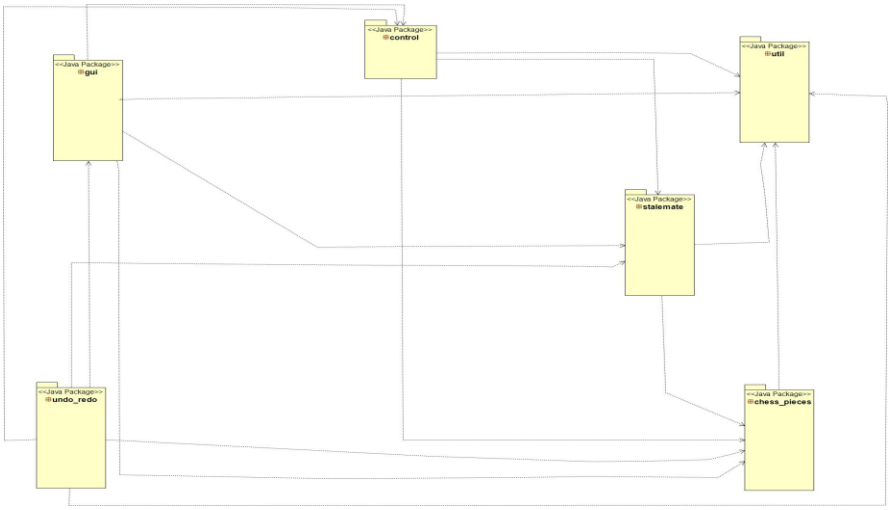


Figure 2: twoplayerchess
Source: Assignment 3 (AD) twoplayerchess-code

1.4.3 Class diagram dependencies

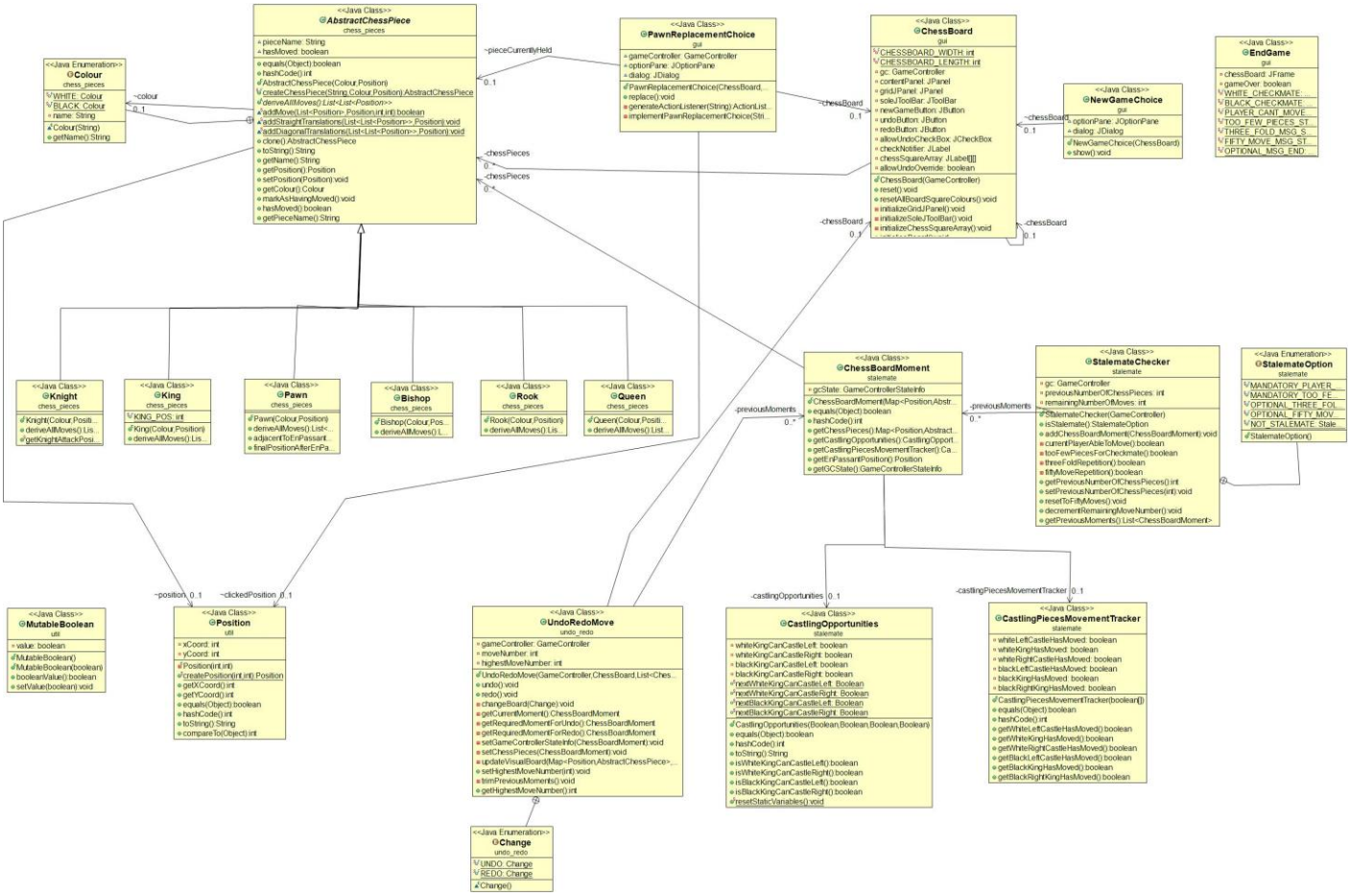


Figure 3: twoplayerchess
Source: Assignment 3 (AD) twoplayerchess-code

Note: Based on my previous knowledge I usually started with the class diagram and the overview how packages and Java class are related. However, I found sonargraph a great tool to tackle software issues and have closer look at the interdependencies.

1.4.4 Sonargraph Explorer

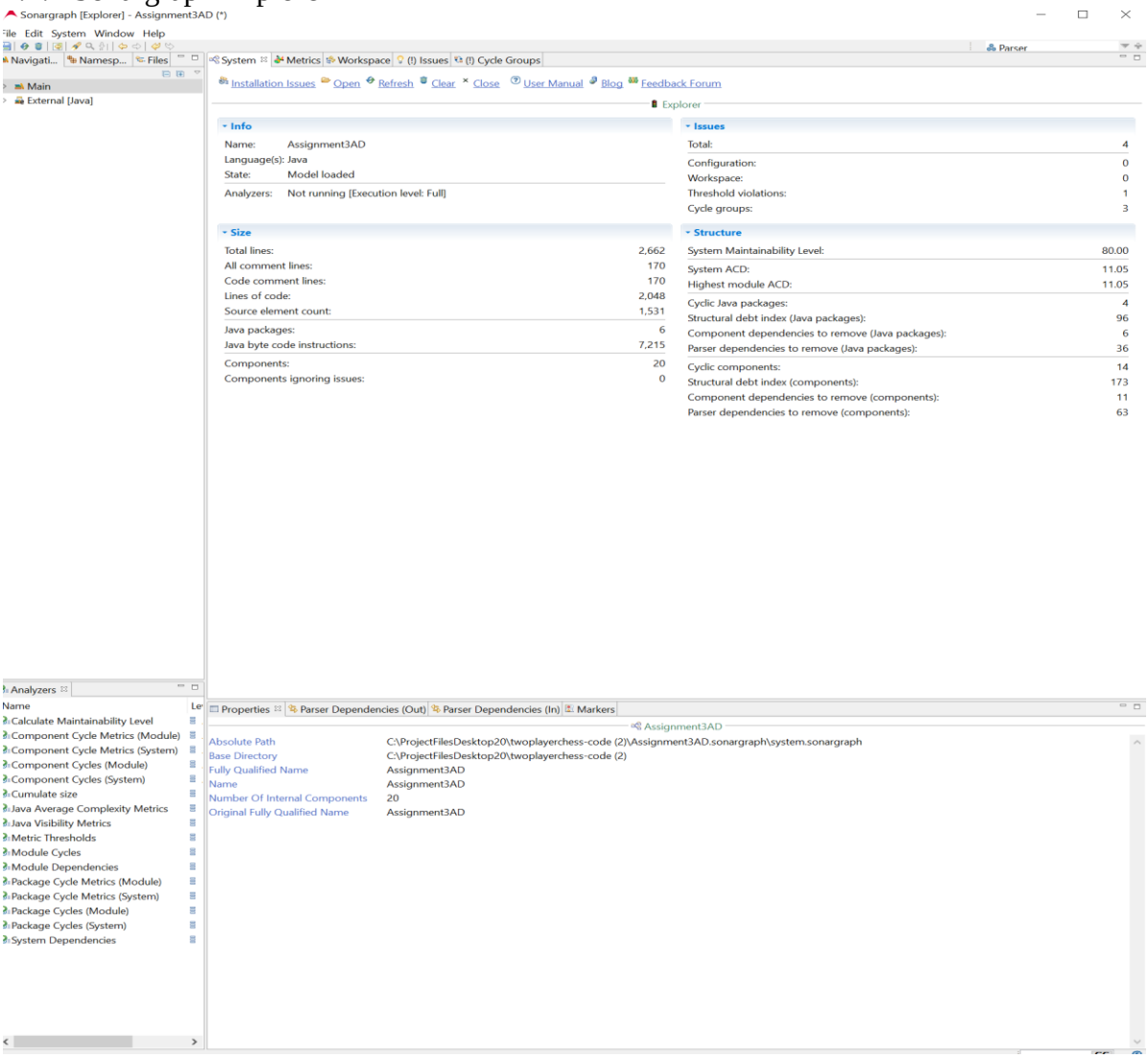


Figure 4: Sonargraph system view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.4.1 System box analyzing twoplayerchess key points:

- 2048 lines of codes,
- 6 Java packages,
- 20 components.

1.4.4.2 Structure box:

The level of structural erosion in the twoPlayerChess software:

- Average Component Dependencies “ACD” is 11.05
- 11.05 Java files depends on average, any changes to a file it might affect 11.05 other files in average.
- Java packages are 6 in total in this project
- 173 structural debt indices in total in this project
- Cyclic components are 4 out of 6 components.
    - 4 out 6 components indicate high coupling
    - Reducing cycling components will increases reusability
    - Following the preceding three design principles will have a great impact on the software:

- Increase cohesion, reducing coupling and increasing abstraction all these three principles indeed increases reusability
- Class must be generalized so that various business could use the software
- Avoiding too much generic or being too specific for using a special library
- Focus to design the system on abstractions and keep the design as simple as possible
- Dividing the packages, modules, class into related Functional, Layers, Communicational, Procedural and Temporal types

1.4.5 Metrics view

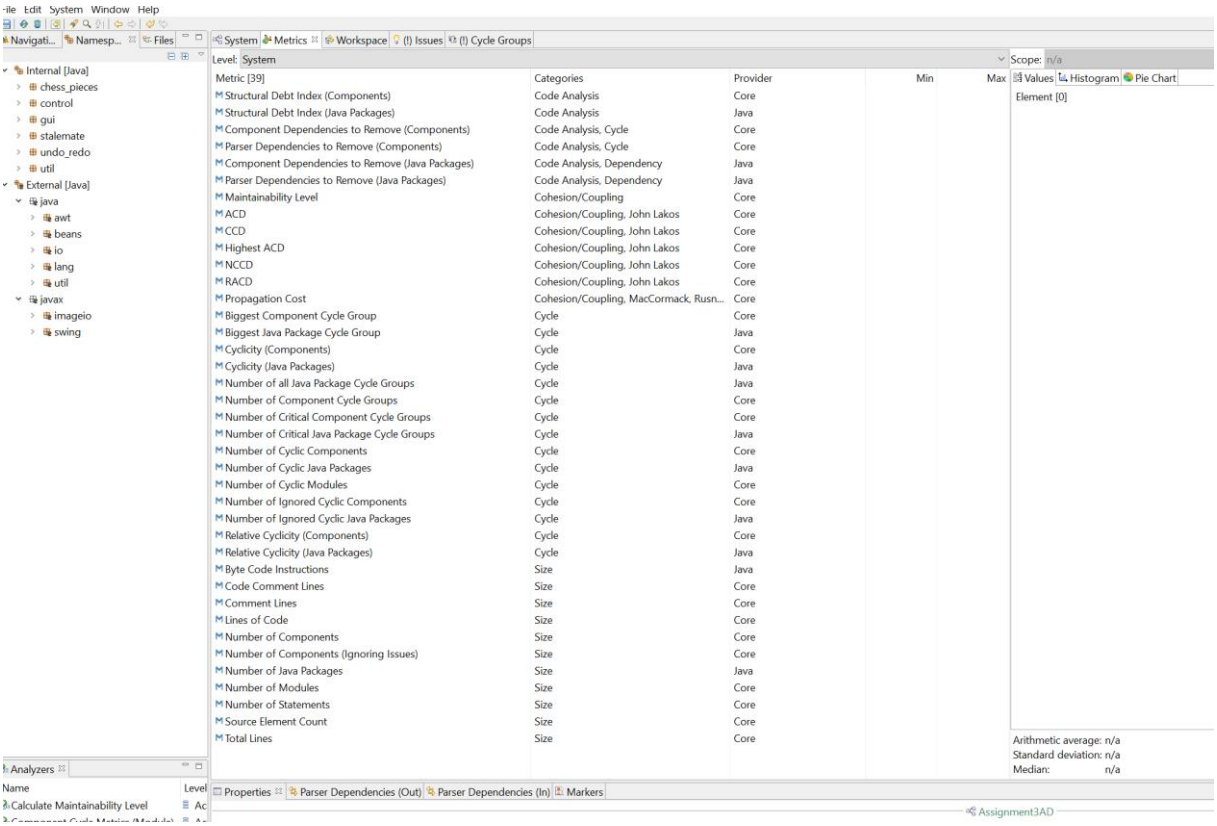


Figure 5: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.6 Metrics view

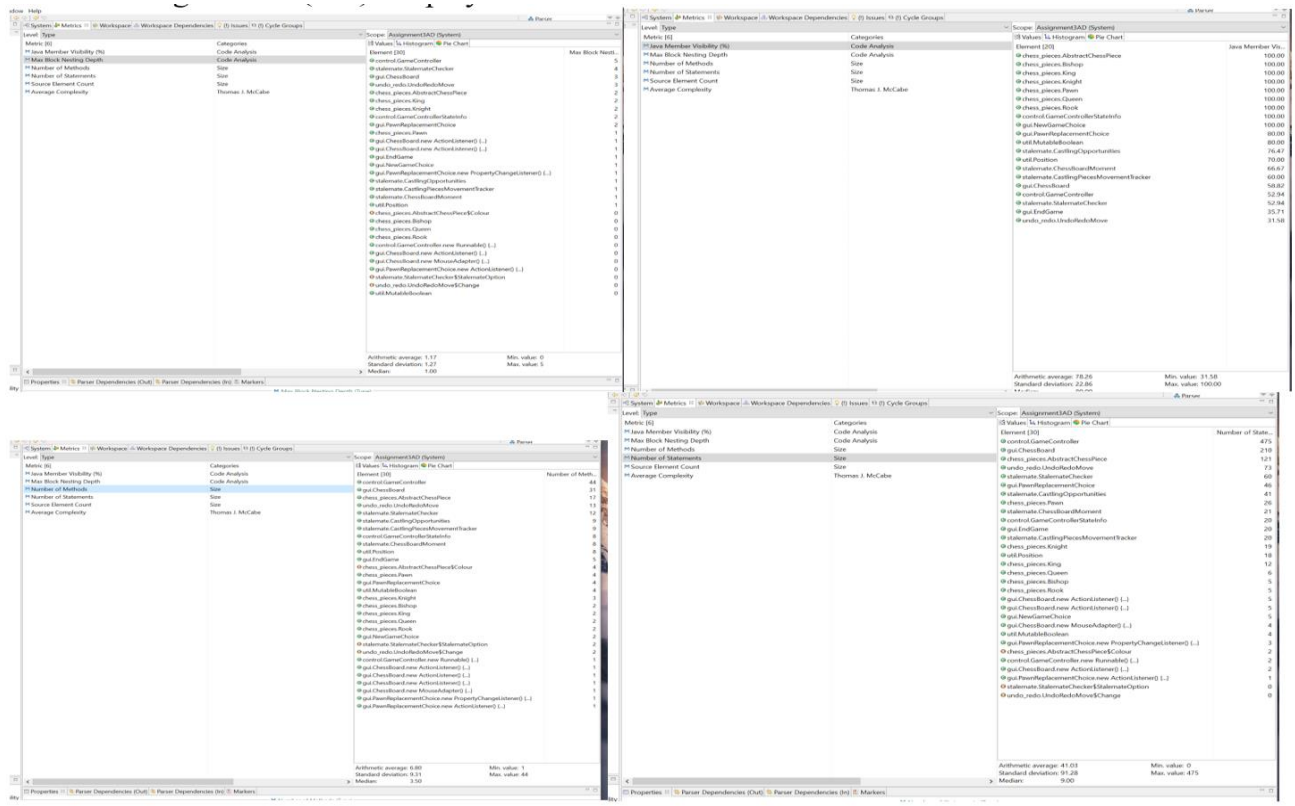


Figure 6: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.7 Metrics Average Complexity view

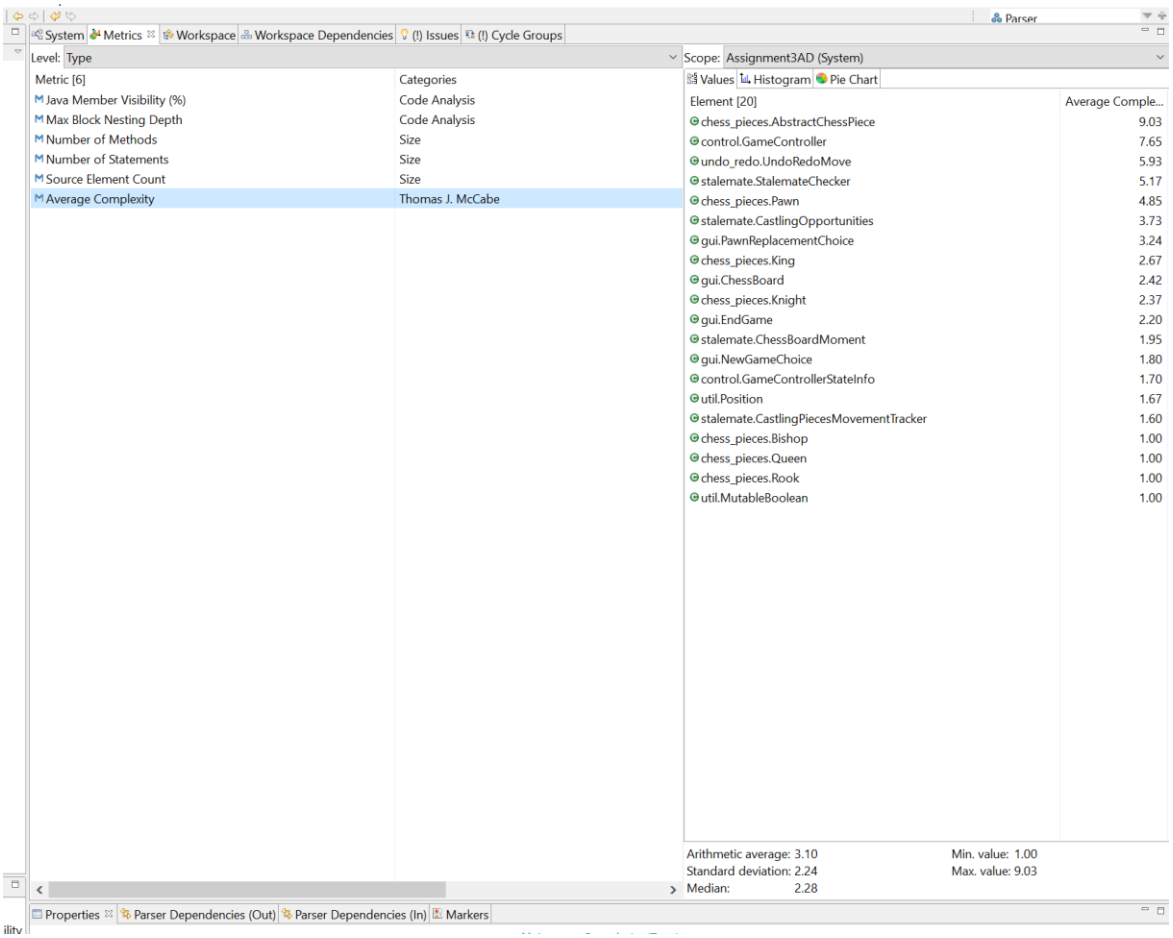


Figure 7: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

Metrics average complexity shows that chess_pieces.AbstractChessPiece class is the most complex class in this project which needs to be considered and control.GameController class is the second most complex class with high coupling which these class are an issue.

1.4.8 Modified Cyclomatic Complexity Routine

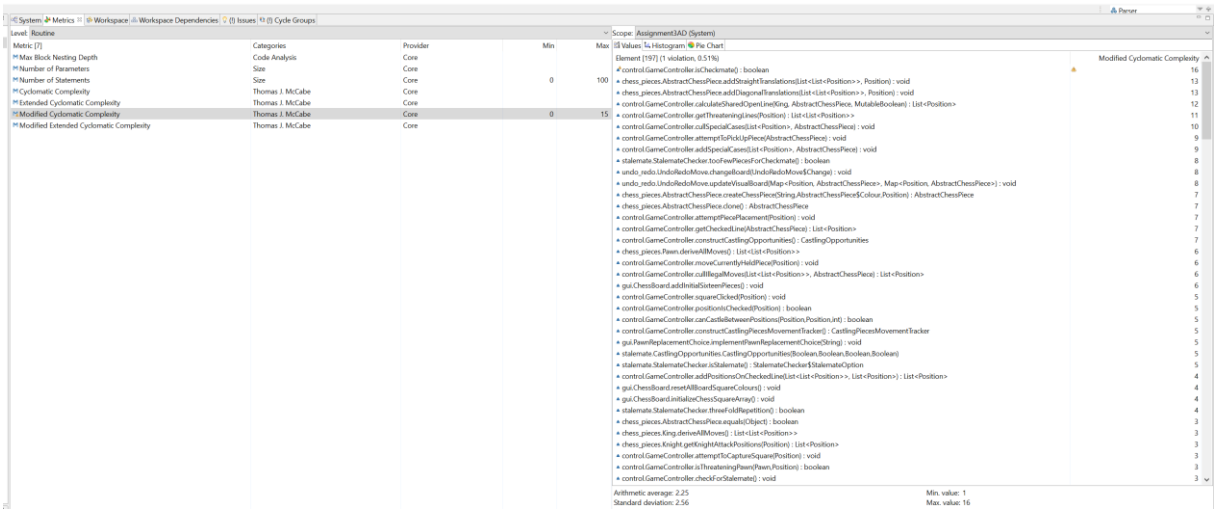


Figure 8: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

Modified Cyclomatic complexity shows that control.GameController.isCheckMate():Boolean have 16 warnings which is the most complex method in this project and control.GameController.isCheckMate():Boolean has high degree of coupling which reduces reusability and flexibility in the software. Sonargraph shows this result that the design principle rules were violated. However, the main goal of designing a software is to focus on design pronciples. Less coupling increases cohesion and reusability.

1.4.9 Show in exploration view in and out.

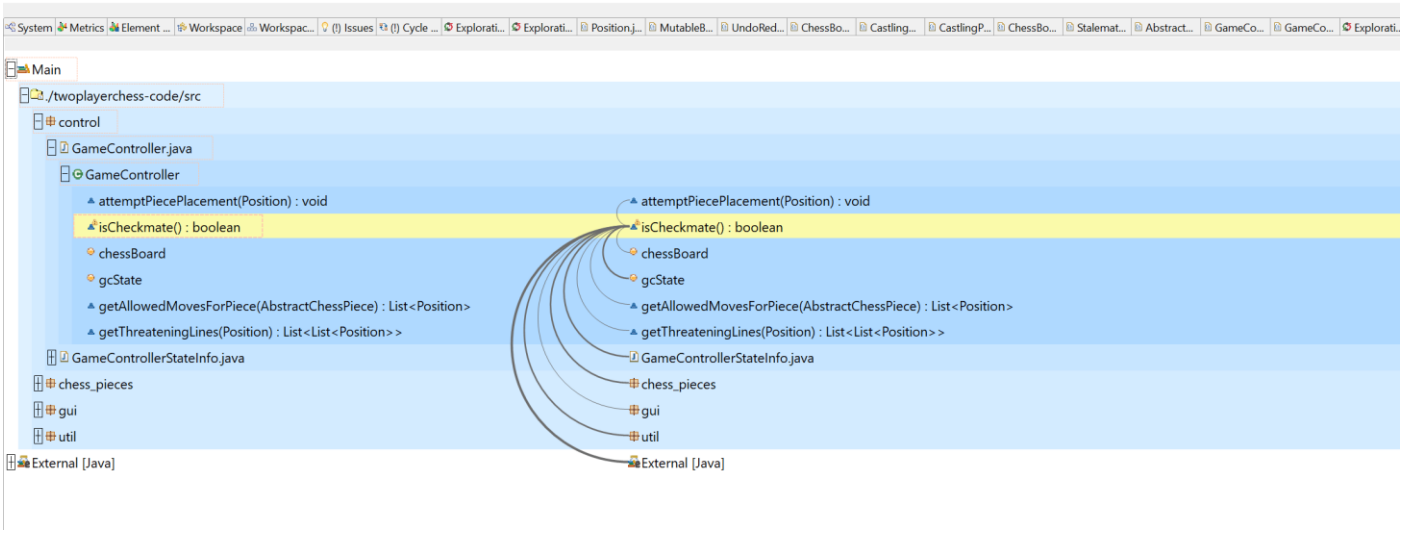


Figure 9: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.10 Show in source

```
    }

    private boolean isCheckmate() {
        King kingInCheck = chessBoard.getKing(gcState.currentPlayerToMove);
        List<List<Position>> threateningLines = getThreateningLines(kingInCheck.getPosition());
        List<Position> possibleKingMoves = getAllowedMovesForPiece(kingInCheck);

        // Definite checkmate
        if (threateningLines.size() > 1 && possibleKingMoves.size() == 0)
            return true;

        gcState.checkBlockingMoves = new ArrayList<Position>();
        if (threateningLines.size() == 1) {
            for (Position checkBlockingPosition : threateningLines.get(0)) {
                gcState.checkBlockingMoves.add(checkBlockingPosition);
            }
        }

        // Add potential knight threats which are not covered by threatening lines.
        List<Position> knightAttackPositions = Knight.getKnightAttackPositions(kingInCheck.getPosition());
        for (Position knightPosition : knightAttackPositions) {
            AbstractChessPiece possibleKnight = chessBoard.getPieceAtPosition(knightPosition);
            if (possibleKnight == null)
                continue;
            if (possibleKnight instanceof Knight && possibleKnight.getColour() != gcState.currentPlayerToMove)
                gcState.checkBlockingMoves.add(knightPosition);
        }

        // Can return at this point if the king himself can move out of check
        if (possibleKingMoves.size() > 0)
            return false;

        // Otherwise, may still be checkmate. Need to see if any of the pieces can move onto one of
        // the checkBlockingPosition's
        List<AbstractChessPiece> currentPlayersPieces = chessBoard.getPlayersPieces(gcState.currentPlayerToMove);
        for (AbstractChessPiece chessPiece : currentPlayersPieces) {
            if (chessPiece instanceof King)
                // King cannot block the check of himself.
                continue;
            List<Position> allowedMoves = getAllowedMovesForPiece(chessPiece);

            for (Position checkBlockingMove : gcState.checkBlockingMoves) {
                if (allowedMoves.contains(checkBlockingMove)) {
                    // It's possible for the check to be blocked, so not checkmate.
                    return false;
                }
            }
        }

        // Final check to see if attacking pawn can be taken by en passant
        if (gcState.checkBlockingMoves.size() == 1 && gcState.checkBlockingMoves.get(0).equals(gcState.enPassantPosition)) {
            for (int i = -1; i < 2; i += 2) {
                Position potentialFriendlyPawnPosition = Position.createPosition(gcState.enPassantPosition.getXCoord() + i,
                        gcState.enPassantPosition.getYCoord());
                AbstractChessPiece friendlyPawn = chessBoard.getPieceAtPosition(potentialFriendlyPawnPosition);
                if (!(friendlyPawn instanceof Pawn))
                    continue;
                List<Position> pawnAllowedMoves = getAllowedMovesForPiece(friendlyPawn);
                if (pawnAllowedMoves.contains(((Pawn)friendlyPawn).finalPositionAfterEnPassant(gcState.enPassantPosition)))
                    return false;
            }
        }

        // No moves can block the threatening line, so checkmate.
        return true;
    }
```

Figure 10: Sonargraph metrics view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.11 Cycle Groups view

Cycle groups view provided an overview about all the cyclic groups in the system. Cycle groups shows that Java package cycle group Main as a single package contains 4 sub-packages which 4 out of 6 packages are cycling in a single cycle group. Furthermore, this indicates that this package has high degree of coupling.

For this project, the interdependencies between modules are the priority. Additionally, these interdependencies will force the developer to bring changes in other modules if any changes are made. The designer should carefully resolve these issues while designing the software in the earlier stages.

The designer should ensure that the software allow other business to hide details, “designing a flexible system”, thus reducing complexity. Software engineer should use their knowledge to make each design decision based on the requirements. Moreover, the designers should be creative and push their boundaries with caution to deliver best product low-coupling and high cohesion.

1.4.12 Cycle groups Java Package cycle view

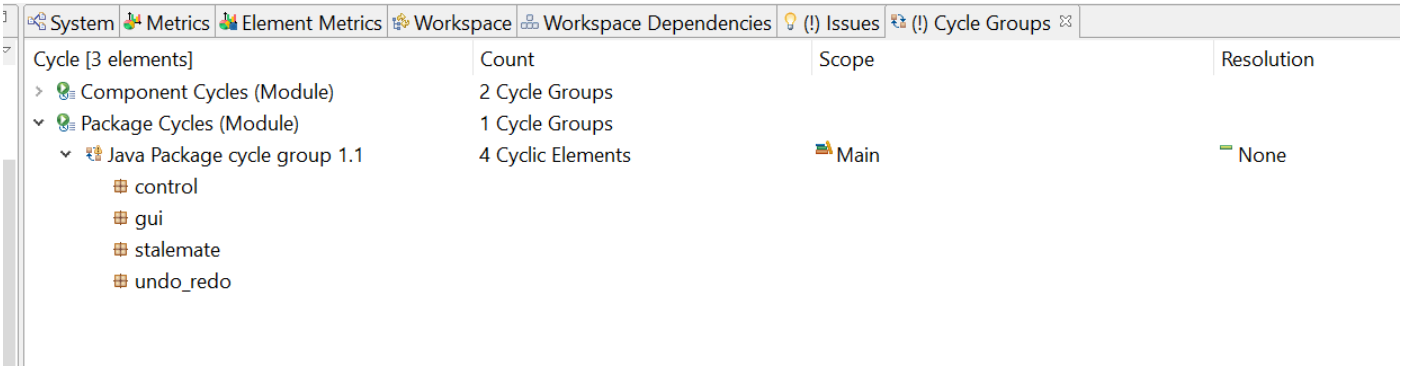


Figure 11: Sonargraph Cycle Groups view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.13 Visualizing the Java package cycle group

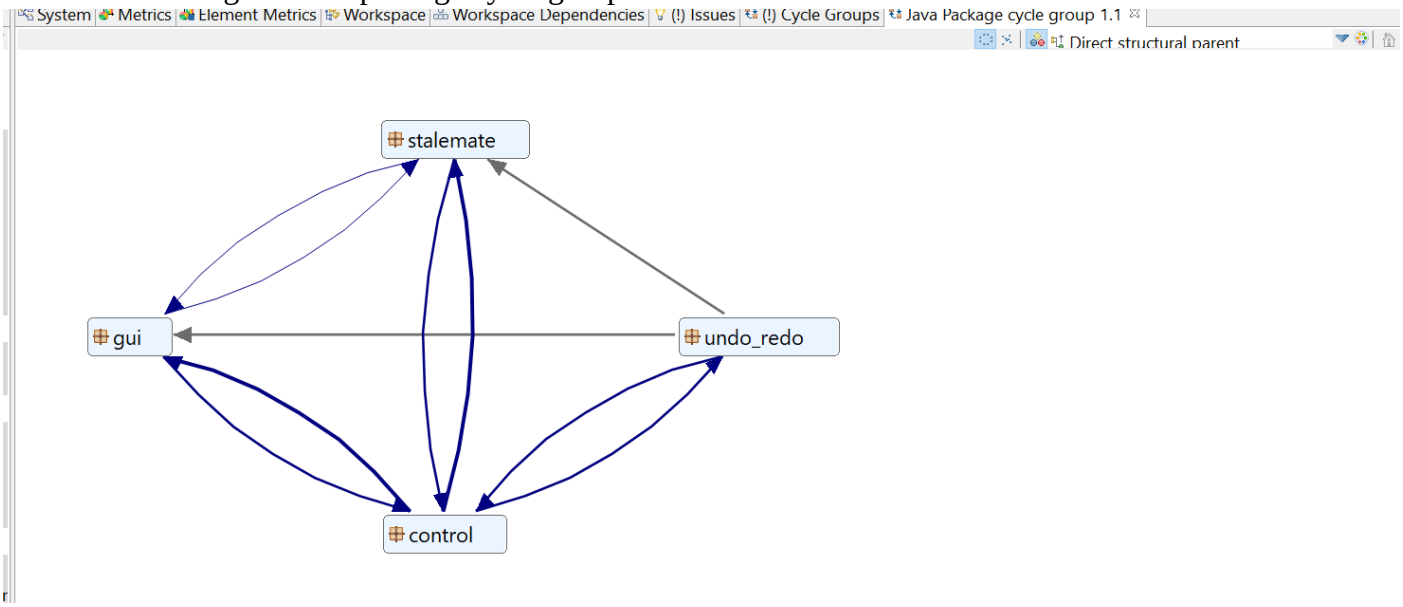


Figure 12: Sonargraph Cycle Groups/Java package group 1.1
Source: Assignment 3 (AD) twoplayerchess-code

1.4.14 Component Cycle (Module)

Component Cycle have two groups that each group have 7 cycling elements. In the figure below there are 7 Java files that are on a cyclic relationship

1.4.15 Show in cycle view

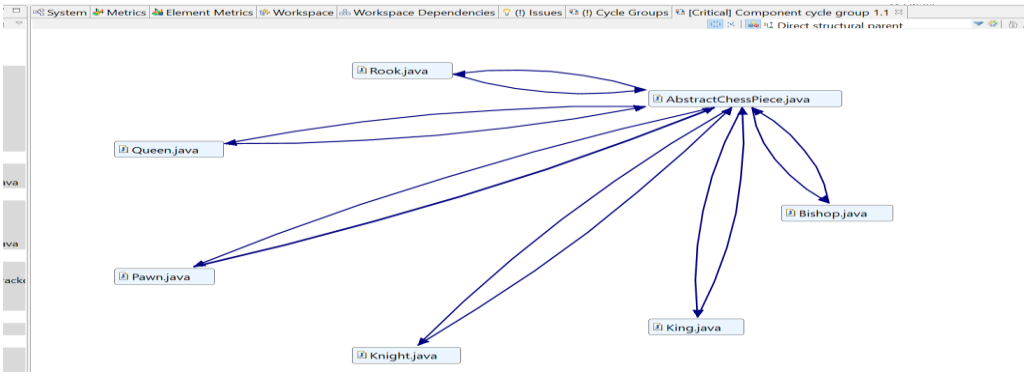


Figure 13: Sonargraph Cycle Groups view
Source: Assignment 3 (AD) twoplayerchess-code

1.4.16 In exploration view

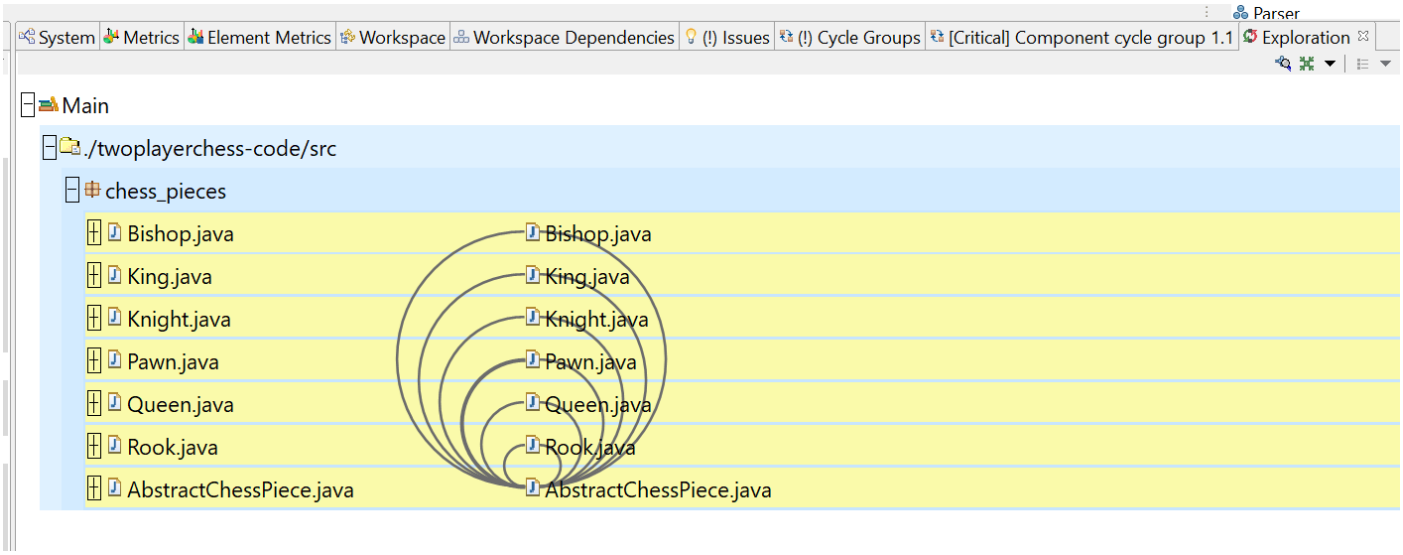


Figure 14: Sonargraph Cycle Groups/Exploration view
Source: Assignment 3 (AD) twoplayerchess-code

The arcs indicate dependencies on both ways left, top-down and right, bottom-up. Top-down arcs are the most used dependency inversion principle according to Robert C.Martin [1]. Robert purpose to build on abstractions, not on implementations because best pattern for a flexible architecture is with low coupling.

In top-down design the engineer first start design at the very high-level structure of the system and then gradually work down to detailed decisions low-level constructs. However, bottom-up design engineers make decisions about the reusable low-level modules first then will decide how to create high-level constructs.

The designer of twoplayerchess has used a mix of top-down and bottom-up approach.

- Top-down design provide the system a good structure [1]
- Bottom-up design is beneficial for reusable modules [1]

As Lethbridge states in his book Architecting and designing software that the overall goals of good design will indeed increases the profit by reducing the cost and ensuring that most of requirements are to meet the customer needs and also top-down and bottom-up design will increase qualities as follow:

- Usability
- Efficiency
- Reliability
- Maintainability

- Reusability … [1]

1.4.17 Single arc dependency

Engineers “Designers” should avoid dependencies bottom-up this means that the designer has designed on implementation not on abstractions [1]. In addition, inversion principle suggests cycle free bottom-up [1]. The designer should try to sort elements in a way to minimize upward dependencies.

Sonargraph helps the designer to see all the incoming and outgoing dependencies within a specific package. At some cases it is very vital to know a specific package or class dependency. Below figure will show how this feature in sonargraph works

1.4.18 util package position: all the incoming and outgoing dependencies

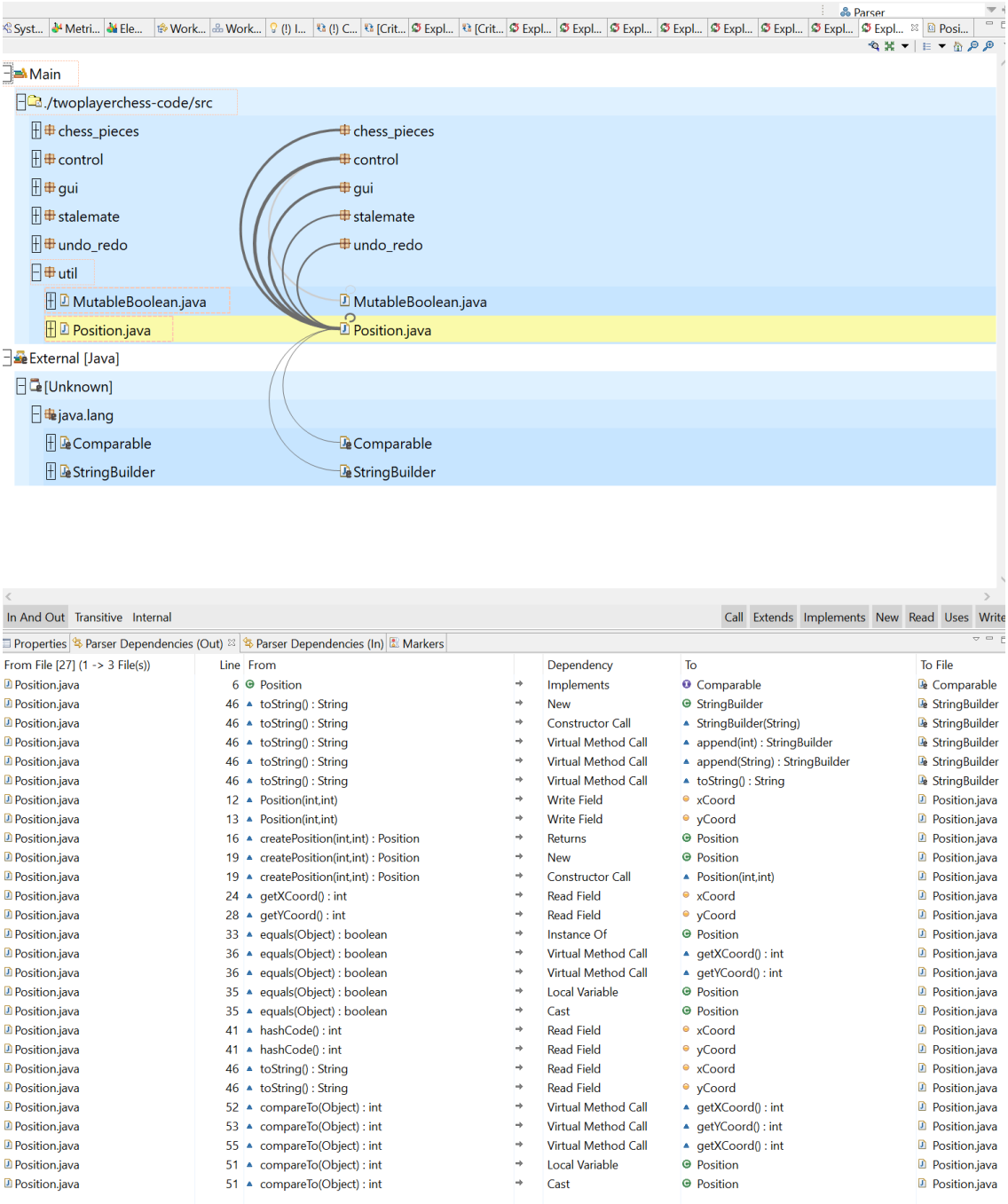


Figure 15: util package dependency all incoming and outgoing dependencies
Source: Assignment 3 (AD) twoplayerchess-code

1.4.19 Issues view

The issues view contains all the findings of sonargraph, all the cycles, threshold violations, workspace problems. Sonargraph show that as threshold violation range in this project is 0-15. The isCheckmate():boolean method has violated the threshold. The designer must modify the threshold accordingly.

This view shows all possible violation and severity of the software. In the figure below:

- Namespace Cycle Group in Java Module 'Main' contains 4 cyclic packages severity **High**
- Threshold Violation: Modified cyclomatic complexity =16(allowed 0-15) severity **High**
- Critical Component Cycle 7 Group "Java Module" Main contains 7 cyclic components Severity **Error**
- Critical Component Cycle 7 Group "Java Module" Main contains 7 cyclic components Severity **Error**

Sonargraph is a great tool to find all the issues. Taking these errors and warnings assists the designer to deliver what the customer might expect at the end.

The focus of this task to cut down the cyclic components. Cyclic component indicates interdependencies between classes and packages which in this project there are two cyclic component that are cycling which I hopefully manage to cut the cyclic dependencies in Task 2.

1.4.20 Issues view

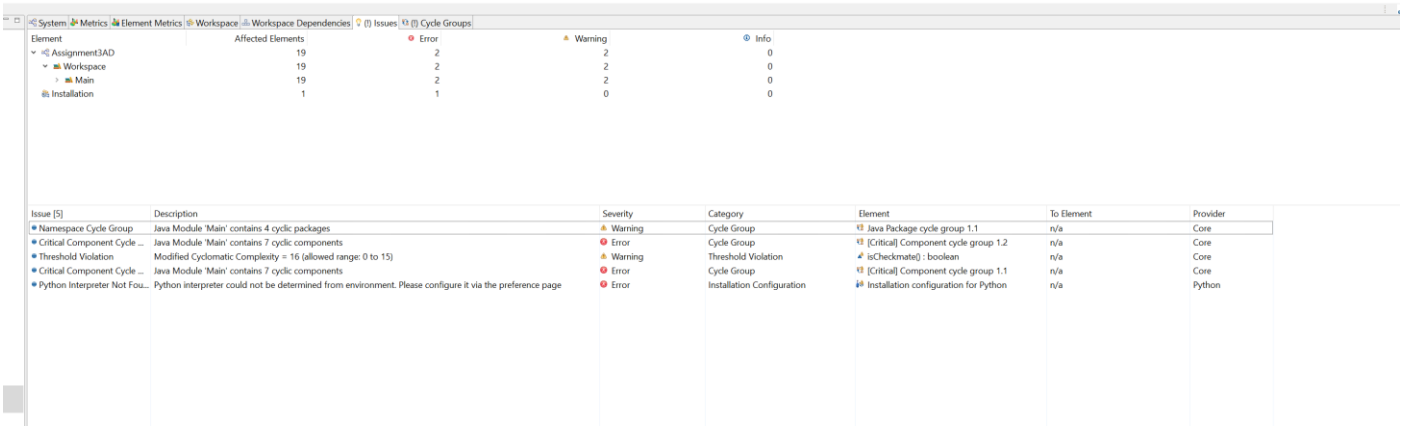


Figure 16: issues view
Source: Assignment 3 (AD) twoplayerchess-code

End of Task 1

2 Task 2

2.1 Re-engineering plan

In section 1.4.19 Issues view, I explained all possible violation and severity of this legacy software. The two-cycle component that are causing errors are in in chess_pieces package and control package

- *Component cycle group 1.1*
- *Component cycle group 1.2*

I already inspected the code and by using Sonargraph I could catch up errors and warnings. This way I can reduce all type of coupling: content, routine call, type use, and inclusion/imports.

Int section 1.4.16 in exploration view I discussed that the designer has used mix of top-down and bottom-up approach. Robert C.Martin purpose to build on abstractions not on implementations because best pattern for a flexible architecture is low coupling and high cohesion. Building on abstraction will increase profit and reduces costs.

To a free cyclic component, I did static and dynamic analysis to cut cyclic dependencies. Additionally, I created interfaces and moved methods and classes into different classes, methods, and packages. For instance, createChessPiece() methods was tightly coupled to the rest of the design. I created a separate class to reduce the coupling and increase the cohesion.

To the best of my knowledge, I kept myself withing the following principles:

- Six golden rules for a successful project
- Inspection checklist
- Code review checklist

2.2 Exploratory refactoring

My findings are: -

- Finding a starting point: started playing around with the code using eclipse auto generate class diagrams and using Sonargraph I increased my understanding of the code and the architecture.
- Code state: very bad "complexed design" component complexity were violating the architecture rules.
- After the static and dynamic analysis I decided to refactor the code

2.3 Refactoring plan

There are many ways to remove dependencies. Splitting main errors into small problems and then work with them individually are the best practice to avoid other issues. One problem at the time. To achieve this goal I used the Mikado method to mitigate my solutions:

2.3.1 Component cycle group 1.1
2.3.2 Component cycle group 1.2

First, 2.3.1 Component cycle group 1.1:

- package: chess_pieces
- Class AbstractChessPiece
- Method createChessPiece()

AbstractChessPiece class was tightly coupled with createChessPiece() and clone() method. These methods made cyclic component dependencies.

createChessPiece(): I decided to create another class ChesspieceFactory in control package to cut the cyclic component dependencies. I could create an interface and implement createChessPiece() method on each chess piece or could create createChessPiece() on each chess piece. However, I Preferred to create a concrete class.

clone (): Each time there was call this clone() method was giving a copy of that object but first clone() method wash checking in AbsractChessPiece class what am I? then return that object call. Now, each chess piece has clone () method. I put clone one chess piece to remove component dependencies. Lastly, AbstractChessPiece class does not know who uses it. Any other class can extend to it now the component dependency has been successfully removed.

Second, 2.3.2 Component cycle group 1.2:

The GameController is calling ChessBoard, ChessBoard is calling GameController, GameContoller is calling UndoRedoMove, UndoRedoMove is calling GameController big miss here.

Many different approaches:

- StalemateChecker should not know about the GameController
- Create IGameController interface
- Create IUndoRedoMove interface
- Moved around classes

At last, I could not remove cyclic component dependencies. However, I added one extra to cyclic component dependency in section main error 2.3.2.

2.3.2.1 A possible re-engineering plan consists of the following set of actions (specified according to the Mikado Method):

- Component cycle group 1.1

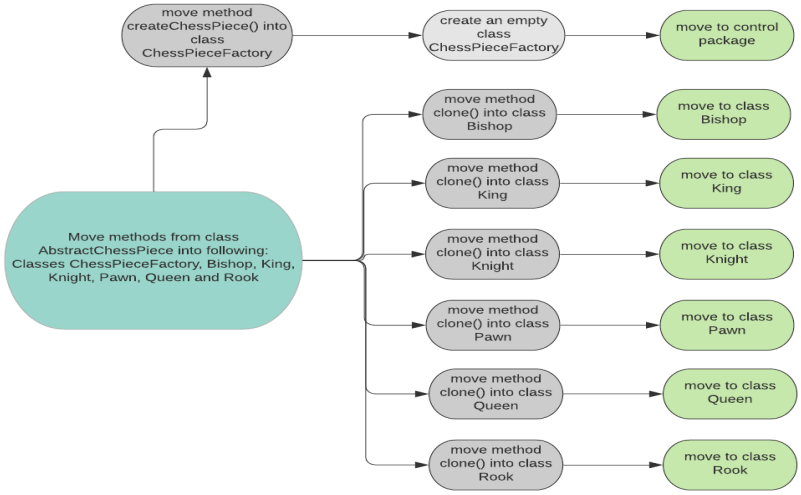


Figure 17: Re-engineering plan component cycle group 1.1

- Component cycle group 1.2

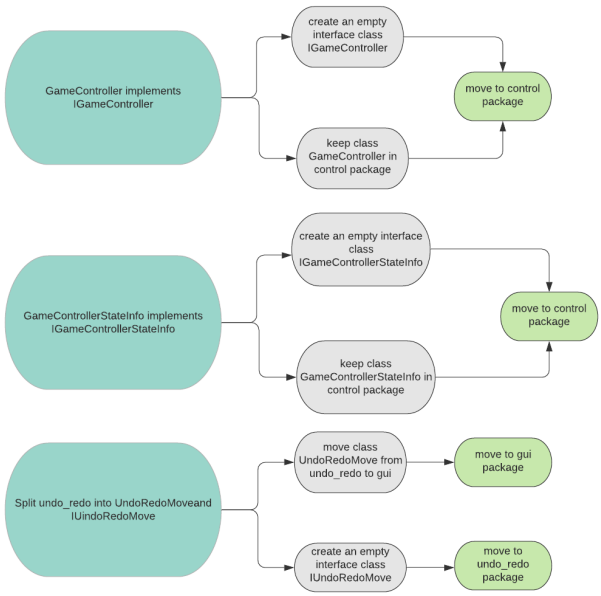


Figure 18: Re-engineering plan component cycle group 1.2

End of Task2

3 Task 3

3.1 Sonargraph Explorer System view

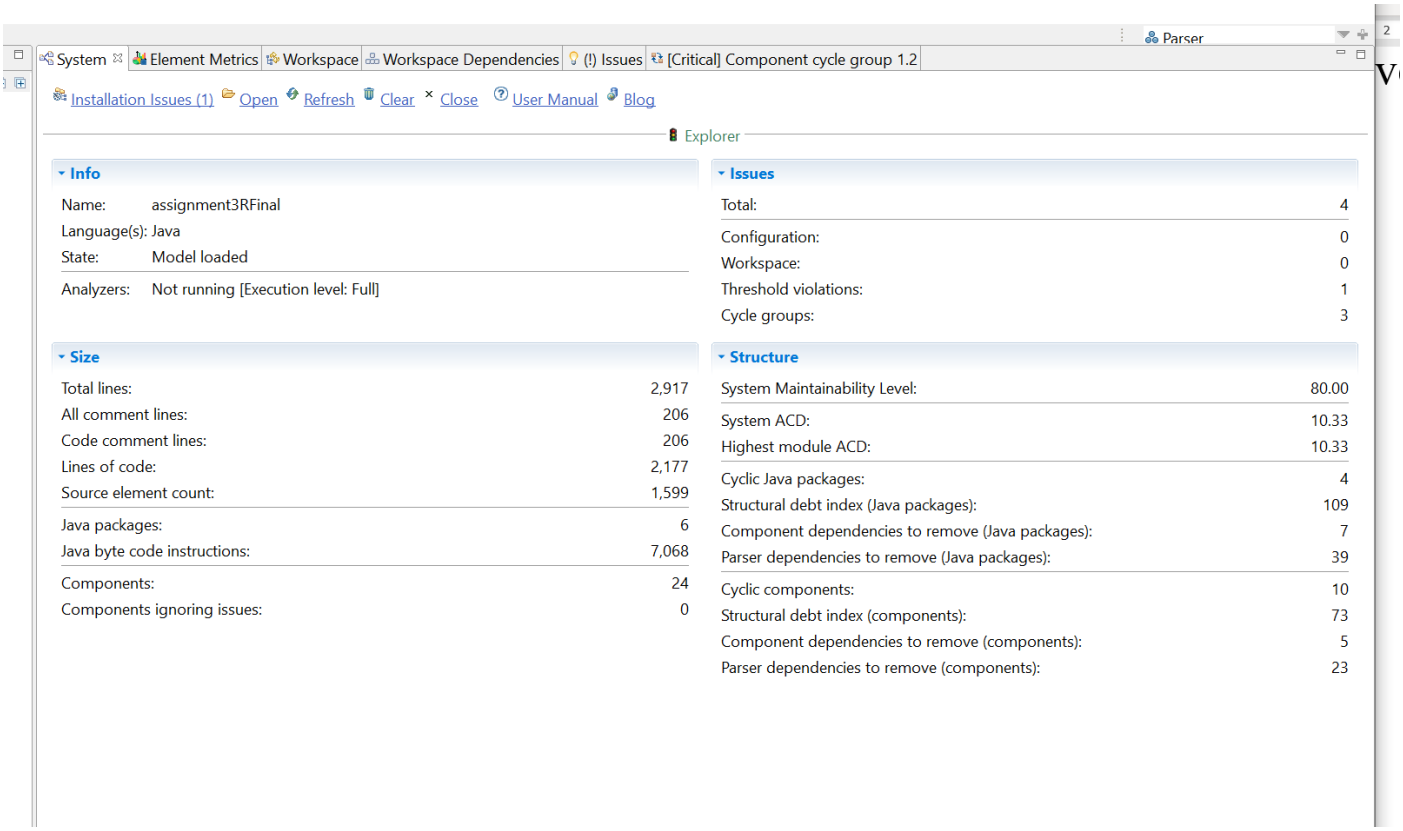


Figure 19: System view

Source: Assignment 3 (AD) twoplayerchess-code

3.1.1.1 System box analyzing twoplayerchess key points:

Current	Old
• 2917 lines of codes,	2048 lines of codes
• 6 Java packages,	6 Java packages
• 24 components	20 components

3.1.1.2 Structure box:

Current	Old
• ACD 10.33	ACD 11.05
• 6 Java packages,	6 Java packages
• 73 structural debt indices	173 structural debt indices

3.1.2 Metrics Average Complexity

Old

Current

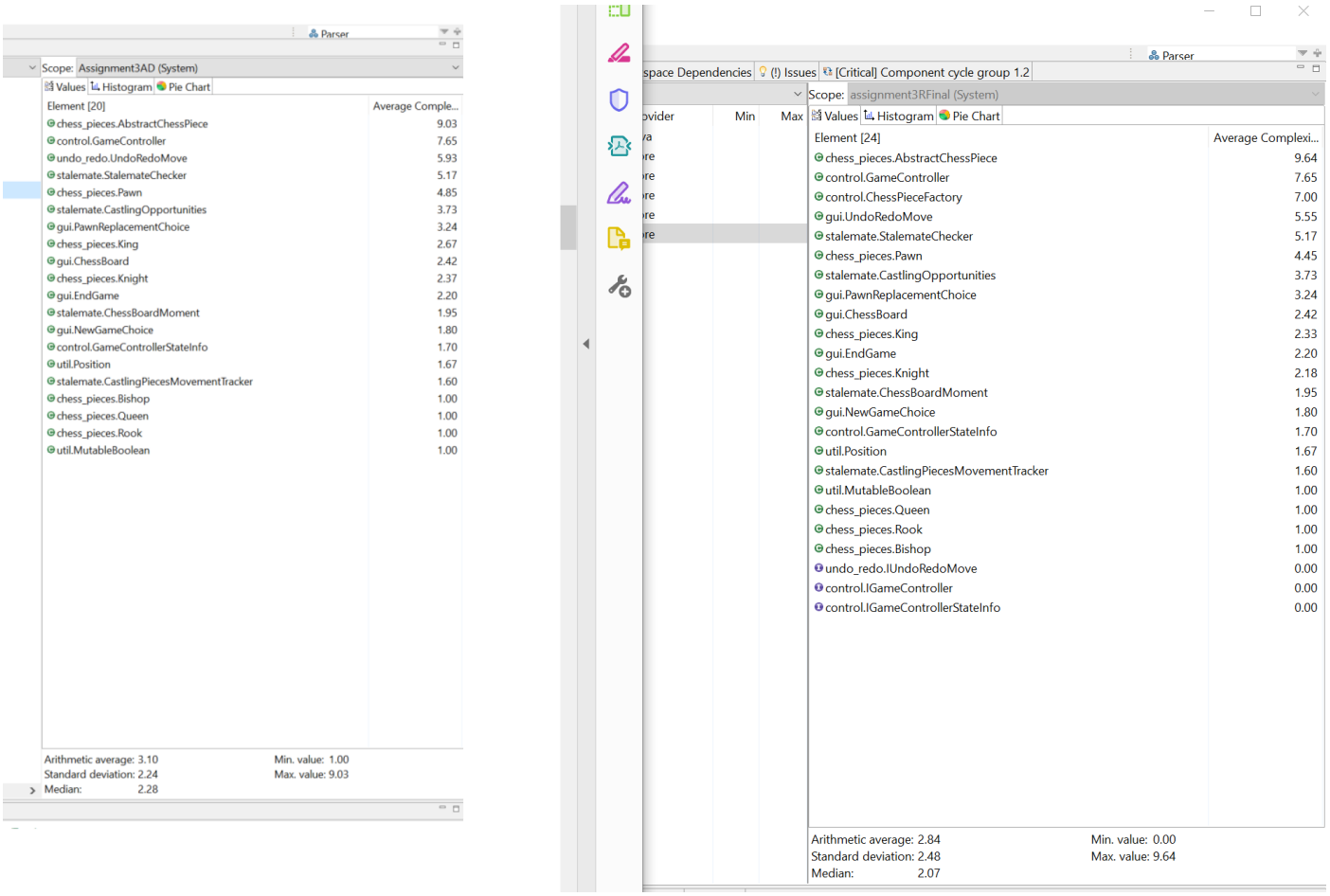


Figure 20: Metrics view
Source: Assignment 3 (AD) twoplayerchess-code

3.1.2.1 Modified Cyclomatic Complexity Routine Current

Scope: assignment3RFinal (System)

Values | Histogram | Pie Chart

Max: 100 | 15

Element [203] (1 violation, 0.49%)	Modified Cyclomatic Complexity
control.GameController.isCheckmate() : boolean	16
chess_pieces.AbstractChessPiece.addStraightTranslations(List<List<Position>>, Position) : void	13
chess_pieces.AbstractChessPiece.addDiagonalTranslations(List<List<Position>>, Position) : void	13
control.GameController.calculateSharedOpenLine(King, AbstractChessPiece, MutableBoolean) : List<Position>	12
control.GameController.getThreateningLines(Position) : List<List<Position>>	11
control.GameController.cullSpecialCases(List<Position>, AbstractChessPiece) : void	10
control.GameController.attemptToPickUpPiece(AbstractChessPiece) : void	9
control.GameController.addSpecialCases(List<Position>, AbstractChessPiece) : void	9
gui.UndoRedoMove.updateVisualBoard(Map<Position, AbstractChessPiece>, Map<Position, AbstractChessPiece>) : void	8
stalemate.StalemateChecker.tooFewPiecesForCheckmate() : boolean	8
gui.UndoRedoMove.changeBoard(IUndoRedoMove$Change) : void	7
control.ChessPieceFactory.createChessPiece(String,AbstractChessPiece$Colour,Position) : AbstractChessPiece	7
control.GameController.attemptPiecePlacement(Position) : void	7
control.GameController.getCheckedLine(AbstractChessPiece) : List<Position>	7
control.GameController.constructCastlingOpportunities() : CastlingOpportunities	7
gui.ChessBoard.addInitialSixteenPieces() : void	6
chess_pieces.Pawn.deriveAllMoves() : List<List<Position>>	6
control.GameController.moveCurrentlyHeldPiece(Position) : void	6
control.GameController.cullIllegalMoves(List<List<Position>>, AbstractChessPiece) : List<Position>	6
gui.PawnReplacementChoice.implementPawnReplacementChoice(String) : void	5
stalemate.CastlingOpportunities.CastlingOpportunities(Boolean,Boolean,Boolean,Boolean)	5
stalemate.StalemateChecker.isStalemate() : StalemateChecker$StalemateOption	5
control.GameController.squareClicked(Position) : void	5
control.GameController.positionIsChecked(Position) : boolean	5
control.GameController.canCastleBetweenPositions(Position,Position,int) : boolean	5
control.GameController.constructCastlingPiecesMovementTracker() : CastlingPiecesMovementTracker	5
gui.ChessBoard.resetAllBoardSquareColours() : void	4
gui.ChessBoard.initializeChessSquareArray() : void	4
stalemate.StalemateChecker.threeFoldRepetition() : boolean	4
control.GameController.addPositionsOnCheckedLine(List<List<Position>>, List<Position>) : List<Position>	4
gui.EndGame.informThatPlayerMayDeclareStalemate(StalemateChecker$StalemateOption,AbstractChessPiece$Colour) : v...	3
gui.PawnReplacementChoice.replace() : void	3
gui.ChessBoard.getKing(AbstractChessPiece$Colour) : King	3
gui.ChessBoard.getPlayersPieces(AbstractChessPiece$Colour) : List<AbstractChessPiece>	3
stalemate.CastlingOpportunities.equals(Object) : boolean	3

Arithmetic average: 2.18
Standard deviation: 2.50
Median: 1.00

Min. value: 1
Max. value: 16

Figure 21:Modified Cyclomatic Complexity Routine

Element [199] (0 violations, 0.00%)	Modified Cyclomatic Complexity
control.GameController.isCheckmate() : boolean	15
chess_pieces.AbstractChessPiece.addStraightTranslations(List<List<Position>>, Position) : void	13
chess_pieces.AbstractChessPiece.addDiagonalTranslations(List<List<Position>>, Position) : void	13
control.GameController.calculateSharedOpenLine(King, AbstractChessPiece, MutableBoolean) : List<Position>	12
control.GameController.getThreateningLines(Position) : List<List<Position>>	11
control.GameController.cullSpecialCases(List<Position>, AbstractChessPiece) : void	10
control.GameController.attemptToPickUpPiece(AbstractChessPiece) : void	9
control.GameController.addSpecialCases(List<Position>, AbstractChessPiece) : void	9
stalemate.StalemateChecker.tooFewPiecesForCheckmate() : boolean	8
undo_redo.UndoRedoMove.changeBoard(UndoRedoMove$Change) : void	8
undo_redo.UndoRedoMove.updateVisualBoard(Map<Position, AbstractChessPiece>, Map<Position, AbstractChessPiece...	8
chess_pieces.AbstractChessPiece.createChessPiece(String,AbstractChessPiece$Colour,Position) : AbstractChessPiece	7
chess_pieces.AbstractChessPiece.clone() : AbstractChessPiece	7
control.GameController.attemptPiecePlacement(Position) : void	7
control.GameController.getCheckedLine(AbstractChessPiece) : List<Position>	7
control.GameController.constructCastlingOpportunities() : CastlingOpportunities	7
chess_pieces.Pawn.deriveAllMoves() : List<List<Position>>	6
control.GameController.moveCurrentlyHeldPiece(Position) : void	6
control.GameController.cullIllegalMoves(List<List<Position>>, AbstractChessPiece) : List<Position>	6
gui.ChessBoard.addInitialSixteenPieces() : void	6
control.GameController.squareClicked(Position) : void	5
control.GameController.positionIsChecked(Position) : boolean	5
control.GameController.canCastleBetweenPositions(Position,Position,int) : boolean	5
control.GameController.constructCastlingPiecesMovementTracker() : CastlingPiecesMovementTracker	5
gui.PawnReplacementChoice.implementPawnReplacementChoice(String) : void	5
stalemate.CastlingOpportunities.CastlingOpportunities(Boolean,Boolean,Boolean,Boolean)	5
stalemate.StalemateChecker.isStalemate() : StalemateChecker$StalemateOption	5
control.GameController.addPositionsOnCheckedLine(List<List<Position>>, List<Position>) : List<Position>	4
gui.ChessBoard.resetAllBoardSquareColours() : void	4
gui.ChessBoard.initializeChessSquareArray() : void	4
stalemate.StalemateChecker.threeFoldRepetition() : boolean	4
chess_pieces.AbstractChessPiece.equals(Object) : boolean	3
chess_pieces.King.deriveAllMoves() : List<List<Position>>	3
chess_pieces.Knight.getKnightAttackPositions(Position) : List<Position>	3
control.GameControllerStateInfo.clone() : GameControllerStateInfo	3

Arithmetic average: 2.24
Standard deviation: 2.52
Median: 1.00

Min. value: 1
Max. value: 15

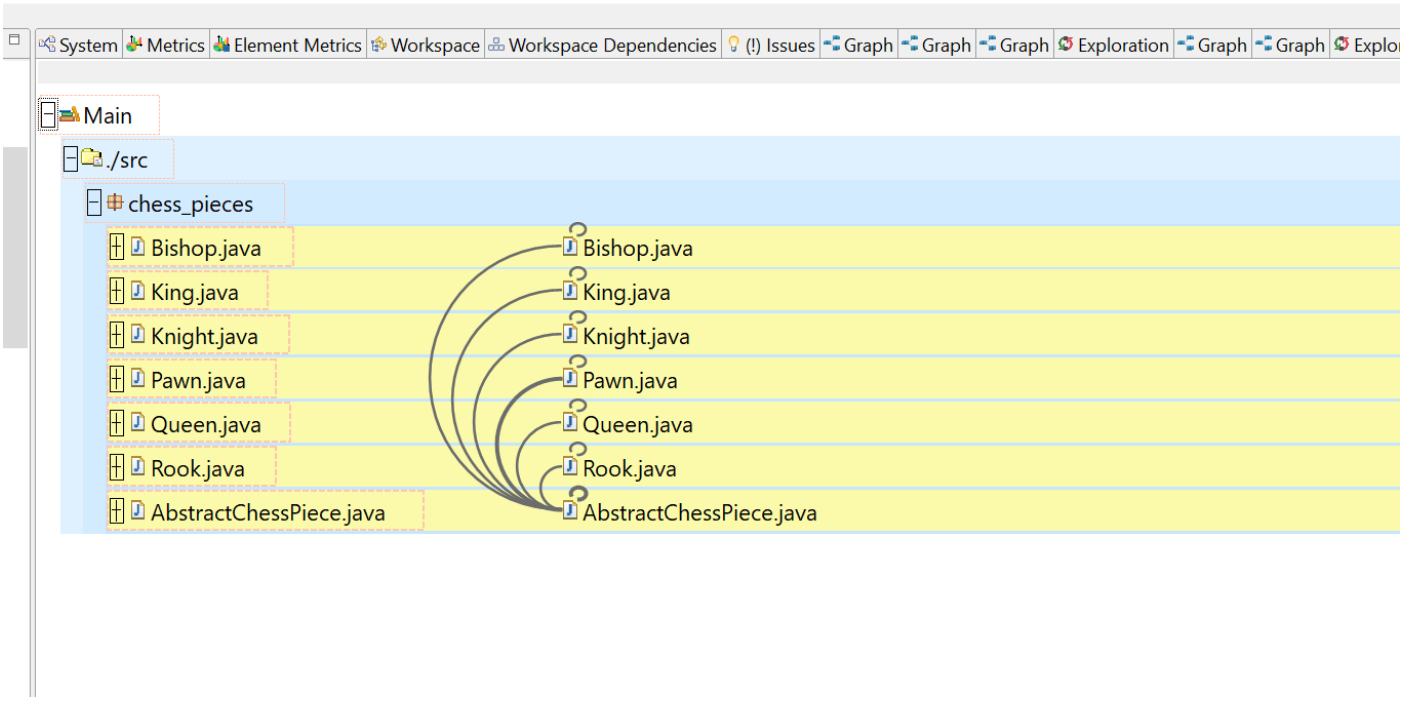
3.1.2.2 Show in cycle view

Cyclic component removed

3.1.2.3 Show in cycle view

Dependencies removed

current



Old

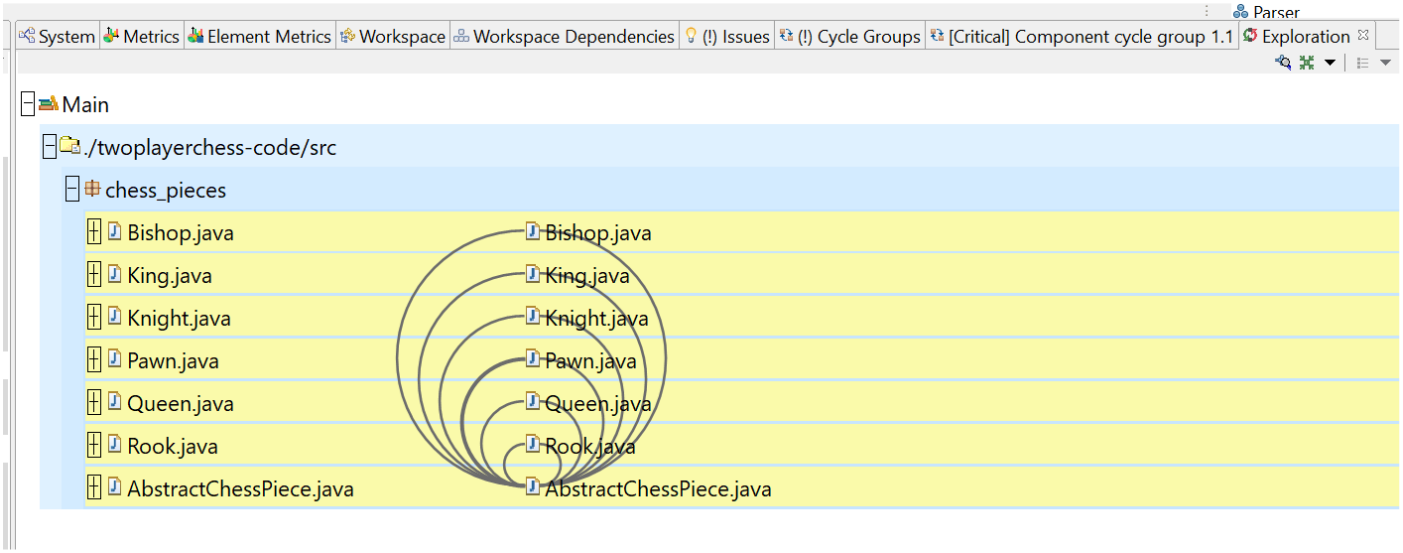
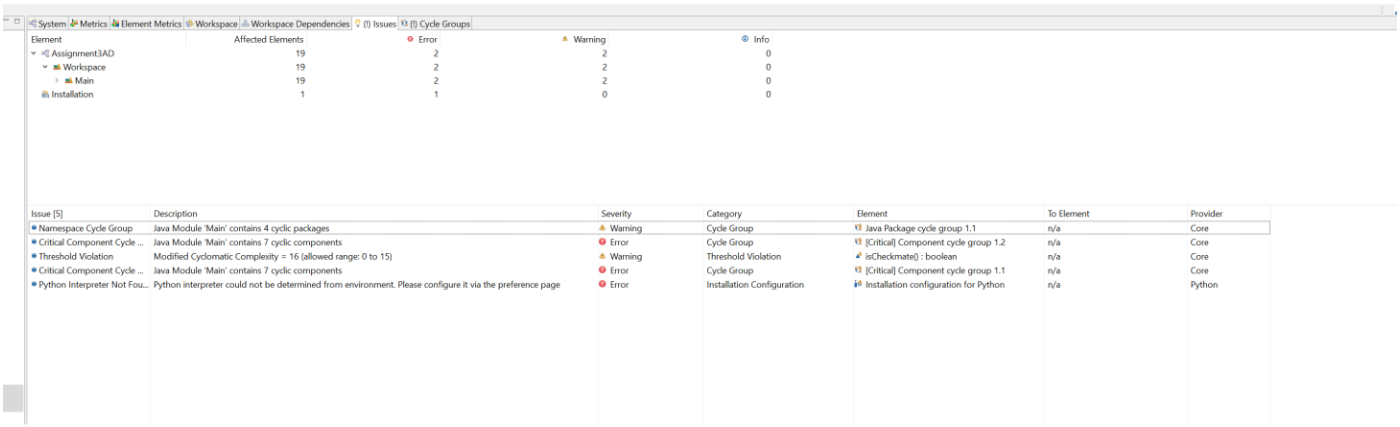


Figure 22: Cycle Groups view
Source: Assignment 3 (AD) twoplayerchess-code

3.1.3 Issues view

Old



Current

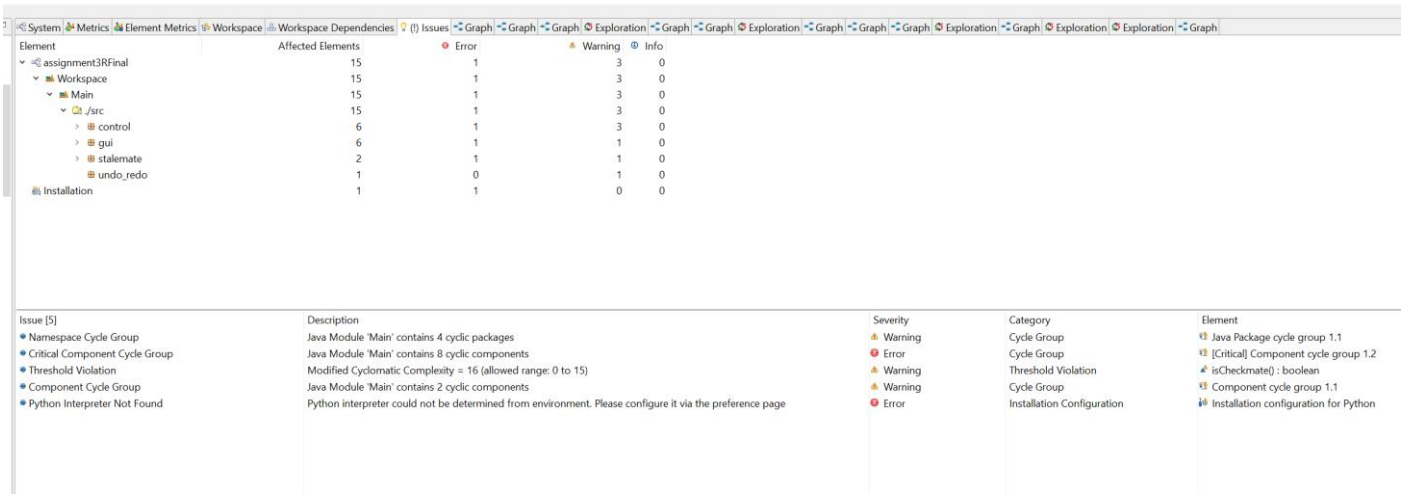


Figure 23: issues view
Source: Assignment 3 (AD) twoplayerchess-code

At last, I have managed to cycle free one of the main errors. However, added extra cyclic component “Java Module ‘Main’ contains 8 cyclic components, Element [Critical] Component cycle group1.2.

End of Task 3

4 Bibliography

[1] L. Laganiere, Architecture and designing software, city: publisher, 2005.