



KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

Programų sistemų testavimas (T120B162)

Laboratorinio darbo Nr. 3 ataskaita

Darbą atliko IFF-7/14

Grupės studentai:

Rimvydas Neverauskas

Valentinas Kasteckis

Henrikas Juzutis

Eligijus Kiudys

Darbą priėmė:

lekt. BARISAS Dominykas

Kaunas, 2020

Turinys

1. Darbo tikslas.....	3
2. Įvadas	3
3. Kodo peržiūros kontrolinis sąrašas	3
4. Programos kodo peržiūra	3
5. Statinė kodo analizė naudojant įrankį	11
6. Išvados.....	16

Pav. 1 PMD statinės analizės rezultatai.....	11
Pav. 2 Geriausių praktikų statinės kodo analizės rezultatai	12
Pav. 3 Kodo stiliaus statinės analizės rezultatai	12
Pav. 4 Dizaino statinės kodo analizės rezultatai.....	13
Pav. 5 Dokumentacijos statinės kodo analizės rezultatai	13
Pav. 6 Linkimo į klaidas kodo statinės analizės rezultatai	13
Pav. 7 Daugelio gijų statinės kodo analizės rezultatai	14
Pav. 8 Našumo statinės kodo analizės rezultatai.....	14
Pav. 9 PMD Designer parašyta taisyklė.....	14
Pav. 10 Eksportuota taisyklė į XML	15
Pav. 11 Pridėta taisyklė	15
Pav. 12 Gauti mūsų taisyklės rezultatai	16

Lentelė 1 Kodo peržiūros lentelės šablonas	3
Lentelė 2 Pažeista mūsų sukurta statinės kodo analizės taisyklė	16

1. Darbo tikslas

Laboratorinio darbo tikslas yra įvertinti mūsų testuojamos programos kokybę naudojant statinio analizavimo metodus, bei įrankius

2. Įvadas

Laboratoriniam darbui pasirinktas objektinio programų projektavimo projektas laivų mūšis, kuris yra rašomas Java kalba. Statinė kodo analizė bus atliekama peržiūrint keletą klasių ir nustatant pažeidimus, bei naudosime statinės kodo analizės Intellij įskiepi PMD.

3. Kodo peržiūros kontrolinis sąrašas

Kodo peržiūrai naudosime šį kontrolinį kodo peržiūros sąrašą:

- Kodo formatavimas – ar tinkami eilučių atitraukimai nuo krašto, ar kodas rašomas nuosekliai, ar yra taikomos kodo rašymo taisyklės pvz., CamelCase naudojamas.
- Kodo architektūra – kodas turi būti struktūrizuotas, ar kodas išskaidytas į failus ar komponentus, ar kodas atitinka taikytus šablonus.
- Kodo gerosios praktikos – ar nėra „hard coding“, naudojamos konstantos ar konfigūracinės reikšmės. Komentarai, turi būti parašyta ne kas daroma, o kodėl tai daroma, aprašomi laikini sprendimai (jei tokie yra), yra pažymimi reikiami padaryti komentarai (TODO). Vengimas daugybės if/else blokų.
- Ne funkciniai reikalavimai – pakartotinis naudojimas, ar yra naudojamas „nepasikartok“ principas (tas pats kodas neturėtų būti kartojamas daugiau kaip du kartus). Išplėčiamumas – ar nesunku praplėsti egzistuojantį kodą. Naudingumas – ar vartotojui yra suprantama naudotojo sąsaja.

Lentelė 1 Kodo peržiūros lentelės šablonas

Failo pavadinimas	
Kodo fragmentas	
Kodo eilutė faile	
Klaidos kategorija	
Klaidos apibūdinimas	
Galimas sprendimas	

4. Programos kodo peržiūra

Failo pavadinimas	ClientShipPart.java
Kodo fragmentas	
<pre>151 public void RotatePart() { 152 Icon oldIcon = this.partImage; 153 BufferedImage bi = new BufferedImage(154 oldIcon.getIconWidth(), 155 oldIcon.getIconHeight(), 156 BufferedImage.TYPE_INT_ARGB); 157 Graphics g = bi.createGraphics(); 158 oldIcon.paintIcon(null, g, 0, 0); 159 g.dispose(); 160 this.partImage = new ImageIcon(rotateImage(bi, this.angle)); 161 }</pre>	
Kodo eilutė faile	151
Klaidos kategorija	Kodo formatavimas
Klaidos apibūdinimas	Nenaudojamas „CamelCase“ formatavimas
Galimas sprendimas	Pakeisti metodo pavadinimą į „rotatePart“

Failo pavadinimas	GameController.java
Kodo fragmentas	
<pre> 879 String coordinates = String.format("%d,%d", seaButton.getGridX(), seaButton.getGridY()); 880 881 if(GameData.shootType.equals(GameConstants.underWaterShoot)) 882 { 883 GameData.UnderWaterUsedInRound = true; 884 } 885 886 Command shootShipOnCommand = new ShootShipOnCommand(GameData.GameId, GameData.PlayerId, coordinates, GameData.shootType, connect); 887 invoker.register(commandName: "shootRegular", shootShipOnCommand); 888 String data = invoker.execute(commandName: "shootRegular"); 889 JSONObject jsonData = new JSONObject(data); </pre>	
<pre> 940 if(GameData.shootType.equals(GameConstants.underWaterShoot)) 941 { 942 GameData.UnderWaterUsedInRound = true; 943 } 944 945 Command shootShipOnCommand = new ShootShipOnCommand(GameData.GameId, GameData.PlayerId, coordinates, GameData.shootType, connect); 946 invoker.register(commandName: "shootRegular", shootShipOnCommand); 947 String data = invoker.execute(commandName: "shootRegular"); 948 JSONObject jsonData = new JSONObject(data); </pre>	
Kodo eilutė faile	879-889 ir 940-948
Klaidos kategorija	Ne funkciniai reikalavimai
Klaidos apibūdinimas	Kodo duplikacija
Galimas sprendimas	Galime sukurti metodą šiam kodui ir panaudoti tą metodą vietoje kodo duplikacijų

Failo pavadinimas	ClientMap.java
Kodo fragmentas	<pre> 28 29 public int getRows() { return rows; } 32 33 public int getCols() { return cols; } 36 </pre>
Kodo eilutė faile	29, 33
Klaidos kategorija	Architektūrinė
Klaidos apibūdinimas	Nenaudojami metodai
Galimas sprendimas	Panaikinti nenaudojamus metodus

Failo pavadinimas	GameController.java
Kodo fragmentas	
<pre> 874 if(GameData.GameLive) { 875 if(GameData.PlayerTurn) { 876 if (callerGridIndex != 0 && refShipButton != null && refShipButton.getType() == 7) { 877 if (seaButton.getBackground() != GameConstants.buttonShotSurface && seaButton.getBackground() != GameConstants.buttonShotColor && seaButton.getBackground() != GameConstants.buttonShotSurface) 878 ((GameData.shootType.equals(GameConstants.superShoot) && GameData.RoundShotSuper == GameData.RoundId <= 0) (GameData.shootType.equals(GameConstants.underWaterShoot) && GameData.RoundShotUnderWater == GameData.RoundId <= 0)) </pre>	
Kodo eilutė faile	877,878
Klaidos kategorija	Kodo formatavimas
Klaidos apibūdinimas	877 eilutėje if formuluotė netelpa į ekraną
Galimas sprendimas	Suskaidyti į kelias eilutes

Failo pavadinimas	ClientShip.java
Kodo fragmentas	
<pre> 196 else if (shipPart.getX() == shot.getX() && shipPart.getY() == shot.getY() && shipPart.getPart().getShipType() == ShipType.SUBMARINE && shot. 197 198 shipHits.add(shot); 199 200 } 201 else if (shipPart.getX() == shot.getX() && shipPart.getY() == shot.getY() && shipPart.getPart().getShipType() == ShipType.SUBMARINE && (shot 202 { 203 if (shipPart.getHitOnce() && shield) { </pre>	

Kodo eilutė faile	Kodo formatavimas
Klaidos kategorija	197,201
Klaidos apibūdinimas	if formuluotė netelpa į ekraną
Galimas sprendimas	Suskaidyti į kelias eilutes

Failo pavadinimas	ClientShipPartBaseShip.java ir ClientShipPartDamaged.java
<p>Kodo fragmentas</p> <pre> 24 Icon oldIcon = new ImageIcon(shipPartImage); 25 BufferedImage bi = new BufferedImage(26 oldIcon.getIconWidth(), 27 oldIcon.getIconHeight(), 28 BufferedImage.TYPE_INT_ARGB); 29 Graphics g = bi.createGraphics(); 30 oldIcon.paintIcon(c: null, g, x: 0, y: 0); 31 g.dispose(); 32 shipPartImage = rotateImage(bi, this.getPart().getAngle()); </pre>	
Kodo eilutė faile	24-32
Klaidos kategorija	Ne funkciniai reikalavimai
Klaidos apibūdinimas	Pasikartojantis kodo fragmentas
Galimas sprendimas	Iškelti logiką į atskirą klasę

Failo pavadinimas	ClientShipPartArmour.java
<p>Kodo fragmentas</p> <pre> 15 public ClientShipPartArmour(ClientShipPartInterface decoratedShipPart) { 16 super(decoratedShipPart); 17 try { 18 shipPartImage = ImageIO.read(new FileInputStream(GameConstants.shipShieldImage)); 19 if (shipPartImage != null) { 20 shipPartImage = shipPartImage.getScaledInstance(ComponentsSizeConstants.gridButtonWidth, ComponentsSizeConstants.gridButtonHeight, Image.SCALE_DEFAULT); 21 } 22 } catch (Exception e) { 23 System.out.println(e.toString()); 24 } 25 } </pre>	
Kodo eilutė faile	22-24
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Nesusidorėjimas su iškilusia klaida
Galimas sprendimas	Perduoti klaidą atsakingam padaliniui arba susidoroti su ja iškart

Failo pavadinimas	Shoot.java
<pre> 52 // error = false; 53 // resultSet = statement.executeQuery("SELECT COUNT(id) as cnt FROM shoots WHERE fk_player_id = " + playerId + " AND x = " + sp 54 // resultSet.next(); 55 // Integer shootCoordinateCheck = resultSet.getInt("cnt"); 56 // if(!error && shootCoordinateCheck > 0) { 57 // error = true; 58 // } 59 // 60 // System.out.println(coordinates.getString(i)); 61 // if(!error) { 62 resultSet = statement.executeQuery("SELECT ship_parts.id as part_id, ships.id as ship_id, ship_parts.armour as armc </pre>	
Kodo eilutė faile	52
Klaidos kategorija	Vizualinė
Klaidos apibūdinimas	Paliktas užkomentuotas kodas
Galimas sprendimas	Ištrinti komentarus

Failo pavadinimas	Log.java, Shoot.java, Message.java ir kt.
-------------------	---

<pre> 1 package server.Entities; 2 3 import ... 4 5 6 7 8 9 public class Log implements Entity { 10 @Override 11 public JSONObject find(JSONObject json) { return null; } 12 13 14 15 @Override 16 public void update(JSONObject json) { 17 18 } 19 20 @Override 21 public void remove(JSONObject json) { 22 23 } 24 25 @Override </pre>	
Kodo eilutė faile	-
Klaidos kategorija	Architektūrinė
Klaidos apibūdinimas	Kiekviena klasė implementuoja Entity interface, tačiau tas interface turi kelis metodus ir tos klasės nevisus metodus realizuoja.
Galimas sprendimas	Panaikinti interface nes galimai yra perteklinis

Failo pavadinimas	Game.java
<pre> 83 resultSet = statement.executeQuery(sql: "SELECT COUNT(DISTINCT fk_player_id) as playerCnt FROM ship 84 "INNER JOIN players ON players.id = ships.fk_player_id " + 85 "WHERE fk_game_id = " + gameId); 86 resultSet.next(); 87 if (resultSet.getInt(columnLabel: "playerCnt") < 2) { 88 jsonObject.append("GameStatus", "Waiting ships"); 89 jsonObject.append("RoundNumber", 0); 90 return jsonObject; 91 } 92 93 resultSet = statement.executeQuery(sql: "SELECT * FROM games WHERE id=" + gameId); 94 resultSet.next(); 95 int currentRound = resultSet.getInt(columnLabel: "round"); 96 97 resultSet = statement.executeQuery(sql: "SELECT DISTINCT fk_player_id as playerId, username FROM sh 98 "INNER JOIN ship_parts ON ship_parts.fk_ship = ships.id " + 99 "INNER JOIN players ON players.id = fk_player_id " + 100 "WHERE destroyed = 0 AND fk_game_id = " + gameId); 101 resultSet.next(); </pre>	
Kodo eilutė faile	64-100
Klaidos kategorija	Vizualinis
Klaidos apibūdinimas	Visi SQL yra labai negražiai hard codinti ir blogas skaitomumas
Galimas sprendimas	Išsikelti visus SQL į atskirą klasę

Failo pavadinimas	EntityMaker.java
	<pre> 110 public JSONObject createMessage(JSONObject json) { return this.messages.create(json); 114 115 public JSONObject getMessages(JSONObject json) { return this.messages.find(json); } 119 120 public void setGames(Entity games) { this.games = games; } 123 124 public void setPlayers(Entity players) { this.players = players; } 127 128 public void setShips(Entity ships) { this.ships = ships; } 131 132 public void setShoots(Entity shoots) { this.shoots = shoots; } 135 136 public void setMessages(Entity messages) { this.messages = messages; } 139 } 140 </pre>
Kodo eilutė faile	110-136 eilutės
Klaidos kategorija	Architektūrinis
Klaidos apibūdinimas	Visi setterių metodai yra naudingi tik testavimui, kad užsimockinti objektus, kitose vietose šie metodai iš viso nėra naudojami.
Galimas sprendimas	Susikurti atskirą klasę kuri paveldėtų šią klasę ir galbūt į ją susikelti tuos metodus

Failo pavadinimas	Context.java
	<pre> 6 public class Context { 7 private Strategy strategy; 8 9 public Context(Strategy strategy) { </pre>
Kodo eilutė faile	7
Klaidos kategorija	Sintaksinė
Klaidos apibūdinimas	Strategy kintamasis yra rekomenduojama, kad būtų final
Galimas sprendimas	Pakeisti property į final

Failo pavadinimas	GameController.java
Kodo fragmentas	
	<pre> 874 if (GameData.GameLive) { 875 if (GameData.PlayerTurn) { 876 if (callerGridIndex != 0 && refShipButton != null && refShipButton.getType() == 7) { 877 if (seaButton.getBackground() != GameConstants.buttonShotSurface && seaButton.getBackground() != GameConstants.buttonShotSu 878 ((GameData.shootType.equals(GameConstants.superShoot) && GameData.RoundShootSuper - GameData.RoundId <= 0) (GameData.shootType.equals(GameConstants.underWaterShoot) && GameD 879 String coordinates = String.format("%d,%d", seaButton.getGridX(), seaButton.getGridY())); 880 881 if (GameData.shootType.equals(GameConstants.underWaterShoot)) 882 GameData.UnderWaterUsedInRound = true; 883 884 } 885 } 886 } </pre>
Kodo eilutė faile	874,875
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Bereikalingai išskirtos if formuluotės
Galimas sprendimas	Galima padaryti if formuluote su and kad būtų: if (GameData.GameLive && GameData.PlayerTurn)

Failo pavadinimas	ConnectionManager.java
Kodo fragmentas	
<pre> 185 // TODO: OLD Logic 186 // for (int partId = 0; partId < shipPartList.size(); partId++) { 187 // String coordinate = String.format("%d,%d",shipPartList[partId].getX(), shipPartList[partId].getY()); 188 // shipParts.put(coordinate); 189 // } </pre>	
Kodo eilutė faile	185-189
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Neaiškus komentaras
Galimas sprendimas	Ištrinti nereikalingą komentarą arba jį pataisyti į aiškesnį.

Failo pavadinimas	ClientPlayer.java
Kodo fragmentas	
<pre> 44 public void appendShips(ClientShip ship) { 45 Collection parts = new ShipPartCollection(ship); 46 Iterator iterator = parts.createIterator(); 47 while (iterator.hasNext()) { 48 ClientShipPart shipPart = (ClientShipPart) iterator.next(); 49 int x = shipPart.getX(); 50 int y = shipPart.getY(); 51 52 ClientCoordinate coordinate = map.getCoordinate(x, y); 53 coordinate.setType(ClientCoordinateType.SHIP); 54 } 55 56 // for (ClientShipPart newShip: ship.shipParts) { 57 // int x = newShip.getX(); 58 // int y = newShip.getY(); 59 // 60 // ClientCoordinate coordinate = map.getCoordinate(x, y); 61 // coordinate.setType(ClientCoordinateType.SHIP); 62 // } 63 ships.add(ship); 64 } </pre>	
Kodo eilutė faile	56-62
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Paliktas užkomentuotas ciklas
Galimas sprendimas	Ištrinti šį komentarą

Failo pavadinimas	PlaceShipOnCommand.java
Kodo fragmentas	

	<pre> 13 private ConnectionManager conManager; 14 private SeaButton[][][] seaButtons; 15 private SeaButton[][][] undoSeaButtons; 16 private SeaButton seaButton; 17 private SeaButton undoSeaButton; 18 private ShipButton refShipButton; 19 private ShipButton undoRefShipButton; 20 private ArrayList<ClientShipPartInterface> newShipParts; 21 private ArrayList<ClientShipPartInterface> undoNewShipParts; 22 private String rotation; 23 private String undoRotation; 24 private ClientShip originalShip; 25 private ClientShip undoOriginalShip; 26 private int callerGridIndex; 27 private int undoCallerGridIndex; 28 private int shipLength; 29 private int undoShipLength; 30 private int shipsLeft; 31 private int undoShipsLeft; </pre>	
Kodo eilutė faile	18,23,24	
Klaidos kategorija	Kodo gerosios praktikos	
Klaidos apibūdinimas	Yra aprašyti nenaudojami kintamieji	
Galimas sprendimas	Juos panaikinti	

Failo pavadinimas	MenuPanel.java	
	Kodo fragmentas	
	<pre> 33 // Unused buttons section 34 private ShipButton unusedButton1; 35 private ShipButton unusedButton2; 36 private ShipButton unusedButton3; 37 private ShipButton unusedButton4; 38 private ShipButton unusedButton5; 39 private ShipButton unusedButton6; </pre>	
Kodo eilutė faile	34-39	
Klaidos kategorija	Kodo gerosios praktikos	
Klaidos apibūdinimas	Nekorektiški kintamųjų pavadinimai	
Galimas sprendimas	Refaktorinti pagal į labiau tinkamus pavadinimus	

Failo pavadinimas	MenuPanel.java	
	Kodo fragmentas	

<pre> 373 @Override 374 public void actionPerformed(ActionEvent e) { 375 Object sender = e.getSource(); 376 377 if (sender.equals(createGameButton)) { 378 // Create Game button pressed 379 delegate.createGameButtonPressed(menuPanel: this, createGameButton); 380 }else if (sender.equals(gameLevelOptions)){ 381 // User selected gameLevel 382 delegate.gameLevelSelected(menuPanel: this, gameLevelOptions); 383 }else if (sender.equals(joinGameButton)){ 384 // Join game button pressed 385 delegate.joinGameButtonPressed(menuPanel: this, joinGameButton); 386 }else if (sender.equals(unusedButton1)){ 387 delegate.unusedButtonPressed(menuPanel: this, unusedButton1); 388 unusedButton1.setActive(true); 389 unusedButton2.setActive(false); 390 unusedButton3.setActive(false); 391 unusedButton4.setActive(false); 392 unusedButton5.setActive(false); 393 unusedButton6.setActive(false); </pre>	
Kodo eilutė faile	380, 383, 386
Klaidos kategorija	Kodo formatavimas
Klaidos apibūdinimas	Netolygūs skliaustų atitraukimai tarp sąlygų
Galimas sprendimas	Suvienodinti tarpus

Failo pavadinimas	FloatingShipConfigurator.java
<p>Kodo fragmentas</p> <pre> 7 import client.Controllers.GameController; 8 import client.GameConstants.GameConstants; 9 import client.GameConstants.ShipParameters; 10 import client.Models.ClientFloatingShip; 11 import client.Models.ClientShip; 12 import client.Models.ClientShipPart; 13 import client.Models.Decorator.ClientShipPartArmour; 14 import client.Models.Decorator.ClientShipPartBase; 15 import client.Models.Decorator.ClientShipPartBaseShip; 16 import client.Models.Decorator.ClientShipPartInterface; 17 import client.Models.GameData; 18 import client.Views.SeaButton; 19 20 import javax.imageio.ImageIO; 21 import javax.swing.*; 22 import java.awt.*; 23 import java.io.FileInputStream; 24 import java.io.IOException; 25 import java.util.ArrayList; </pre>	
Kodo eilutė faile	7, 8, 18, 20, 21, 22, 23, 24
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Nenaudojami importai
Galimas sprendimas	Panaikinti nenaudojamus importus

Failo pavadinimas	ShootShipOnCommand.java
-------------------	-------------------------

Kodo fragmentas	
<pre> 3 import client.Models.Bridge.*; 4 import client.Models.ConnectionManager; 5 import client.Models.GameData; 6 import org.json.JSONObject; </pre>	
Kodo eilutė faile	3, 5, 6
Klaidos kategorija	Kodo gerosios praktikos
Klaidos apibūdinimas	Nenaudojami importai
Galimas sprendimas	Panaikinti nenaudojamus importus

5. Statinė kodo analizė naudojant įrankį

Statinės kodo analizės įrankiu yra testuojama aplikacija, kuri yra padaryta su Java programavimo kalba, todėl šiam testavimui naudosime IntelliJ PMD statinės kodo analizės įskiepi, kuris leidžia atlikti šių kategorijų testus:

- Geriausių kodo praktikų
- Kodo stiliaus
- Dizaino
- Dokumentacijos
- Kodo linkusio į klaidas
- Daugelio gijų
- Našumo

Taip pat pasirašysime savo taisyklę, kuri bus tikrins ar „if“ sąlyga turi skliaustus, dėl patogesnio skaitomumo.

Atlikę statinę kodo analizę naudodami IntelliJ PMD įskiepi gavome tokius rezultatus:

Pav. 1 PMD statinės analizės rezultatai



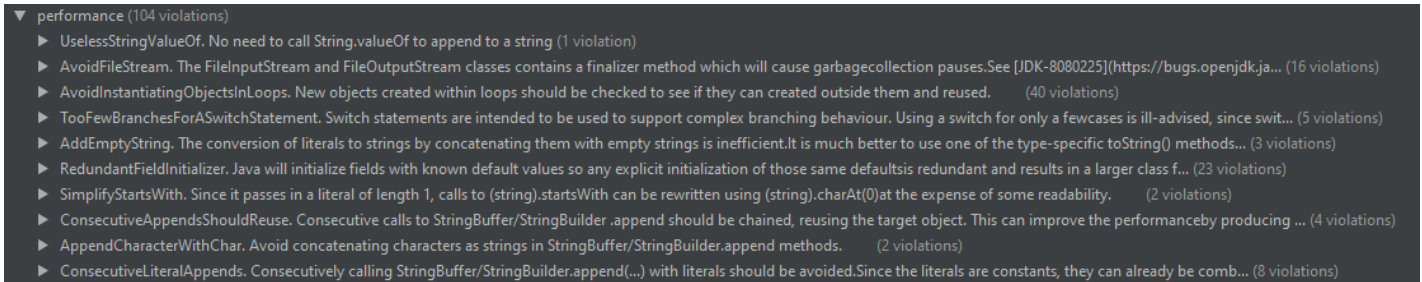
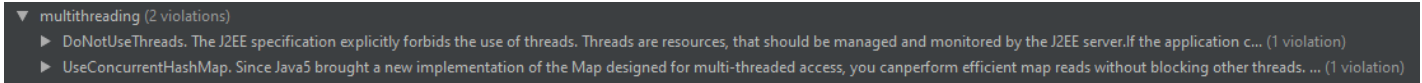
▼ bestpractices (380 violations)	
▶ SwitchStmtsShouldHaveDefault. All switch statements should include a default option to catch any unspecified values. (24 violations)	
▶ PositionLiteralsFirstInComparisons. Position literals first in comparisons, if the second argument is null then NullPointerExceptions can be avoided, they will just return false.... (11 violations)	
▶ UnusedImports. Avoid unused import statements to prevent unwanted dependencies. This rule will also find unused on demand imports, i.e. import com.foo.*. (149 violations)	
▶ SystemPrintln. References to System.out.println are usually intended for debugging purposes and can remain in the codebase even in production code. By using a logger one can e... (23 violations)	
▶ LooseCoupling. The use of implementation types (i.e., HashSet) as object references limits your ability to use alternate implementations in the future as requirements change. When... (41 violations)	
▶ UnusedLocalVariable. Detects when a local variable is declared and/or assigned, but not used. (2 violations)	
▶ UnusedPrivateField. Detects when a private field is declared and/or assigned a value, but not used. (19 violations)	
▶ ArraysStoredDirectly. Constructors and methods receiving arrays should clone objects and store the copy. This prevents future changes from the user from affecting the original ar... (4 violations)	
▶ OneDeclarationPerLine. Java allows the use of several variables declaration of the same type on one line. However, it can lead to quite messy code. This rule looks for several dec... (6 violations)	
▶ AvoidPrintStackTrace. Avoid printStackTrace() (11 violations)	
▶ AvoidReassigningLoopVariables. Reassigning loop variables can lead to hard-to-find bugs. Prevent or limit how these variables can be changed. In foreach-loops, configured by the ... (4 violations)	
▶ AbstractClassWithoutAbstractMethod. The abstract class does not contain any abstract methods. An abstract class suggests an incomplete implementation, which is to be completed by ... (1 violation)	
▶ UnusedPrivateMethod. Unused Private Method detects when a private method is declared but is unused. (9 violations)	
▶ UseVarargs. Java 5 introduced the varargs parameter declaration for methods and constructors. This syntactic sugar provides flexibility for users of these methods and constructor... (1 violation)	
▶ MethodReturnsInternalArray. Exposing internal arrays to the caller violates object encapsulation since elements can be removed or replaced outside of the object that owns it. It ... (1 violation)	
▶ PreserveStackTrace. Throwing a new exception from a catch block without passing the original exception into the new exception will cause the original stack trace to be lost making... (6 violations)	
▶ ForLoopVariableCount. Having a lot of control variables in a 'for' loop makes it harder to see what range of values the loop iterates over. By default this rule allows a regular '... (1 violation)	
▶ CheckResultSet. Always check the return values of navigation methods (next, previous, first, last) of a ResultSet. If the value return is 'false', it should be handled properly. ... (16 violations)	
▶ ForLoopCanBeForeach. Reports loops that can be safely replaced with the foreach syntax. The rule considers loops over lists, arrays and iterators. A loop is safe to replace if it ... (1 violation)	
▶ JUnitTestContainsTooManyAsserts. Unit tests should not contain too many asserts. Many asserts are indicative of a complex test, for which it is harder to verify correctness. Con... (15 violations)	
▶ JUnitAssertionsShouldIncludeMessage. JUnit assertions should include an informative message - i.e., use the three-argument version of assertEquals(), not the two-argument version... (34 violations)	
▶ JUnitTestsShouldIncludeAssert. JUnit tests should include at least one assertion. This makes the tests more robust, and using assert with messages provide the developer a clear... (1 violation)	

▼ codestyle (3324 violations)	
▶ ShortVariable. Fields, local variables, or parameter names that are very short are not helpful to the reader. (268 violations)	
▶ PackageCase. Detects when a package definition contains uppercase characters. (123 violations)	
▶ LongVariable. Fields, formal arguments, or local variable names that are too long can make the code difficult to follow. (103 violations)	
▶ MethodArgumentCouldBeFinal. A method argument that is never re-assigned within the method can be declared final. (857 violations)	
▶ LocalVariableCouldBeFinal. A local variable assigned only once can be declared final. (826 violations)	
▶ UselessParentheses. Useless parentheses should be removed. (69 violations)	
▶ CommentDefaultAccessModifier. To avoid mistakes if we want that an Annotation, Class, Enum, Method, Constructor or Field have a default access modifier we must add a comment at th... (67 violations)	
▶ ConfusingTernary. Avoid negation within an "if" expression with an "else" clause. For example, rephrase: if (x != y) diff() (23 violations)	
▶ OnlyOneReturn. A method should have only one exit point, and that should be the last statement in the method. (309 violations)	
▶ AvoidPrefixingMethodParameters. Prefixing parameters by 'in' or 'out' pollutes the name of the parameters and reduces code readability. To indicate whether or not a parameter will... (3 violations)	
▶ DefaultPackage. Use explicit scoping instead of accidental usage of default package private level. The rule allows methods and fields annotated with Guava's @VisibleForTesting. ... (66 violations)	
▶ DuplicateImports. Duplicate or overlapping import statements should be avoided. (2 violations)	
▶ VariableNamingConventions. A variable naming conventions rule - customize this to your liking. Currently, it checks for final variables that should be fully capitalized and non-f... (186 violations)	
▶ FieldNamingConventions. Configurable naming conventions for field declarations. This rule reports variable declarations which do not match the regex that a... (185 violations)	
▶ ClassNamingConventions. Configurable naming conventions for type declarations. This rule reports type declarations which do not match the regex that applie... (34 violations)	
▶ ShortClassName. Short Classnames with fewer than e.g. five characters are not recommended. (10 violations)	
▶ UnnecessaryLocalBeforeReturn. Avoid the creation of unnecessary local variables (13 violations)	
▶ UnnecessaryConstructor. This rule detects when a constructor is not necessary (8 violations)	
▶ AbstractNaming. Abstract classes should be named 'AbstractXXX'. This rule is deprecated and will be removed with PMD 7.0.0. The rule is replaced by {% rule java/codestyle/ClassNa... (9 violations)	
▶ UnnecessaryModifier. Fields in interfaces and annotations are automatically 'public static final', and methods are 'public abstract'. Classes, interfaces or annotations nested in ... (12 violations)	
▶ AtLeastOneConstructor. Each non-static class should declare at least one constructor. Classes with solely static members are ignored, refer to [UseUtilityClassRule](pmd_rules_jav... (45 violations)	
▶ CallSuperInConstructor. It is a good practice to call super() in a constructor. If super() is not called but another constructor (such as an overloaded constructor) is called, thi... (14 violations)	
▶ UseDiamondOperator. Use the diamond operator to let the type be inferred automatically. With the Diamond operator it is possible to avoid duplication of the type parameters. Inste... (18 violations)	
▶ BooleanGetMethodName. Methods that return boolean results should be named as predicate statements to denote this. i.e. 'isReady()', 'hasValues()', 'canCommit()', 'willFail()', etc... (2 violations)	
▶ MethodNamingConventions. Configurable naming conventions for method declarations. This rule reports method declarations which do not match the regex that a... (2 violations)	
▶ PrematureDeclaration. Checks for variables that are defined before they might be used. A reference is deemed to be premature if it is created right before a block of code that doe... (20 violations)	
▶ UseShortArrayInitializer. When declaring and initializing array fields or variables, it is not necessary to explicitly create a new array using 'new'. Instead one can simply defi... (1 violation)	
▶ IfStmtsMustUseBraces. Avoid using if statements without using braces to surround the code block. If the code formatting or indentation is lost then it becomes difficult to separa... (8 violations)	
▶ ControlStatementBraces. Enforce a policy for braces on control statements. It is recommended to use braces on 'if ... else' statements and loop statements, ... (11 violations)	
▶ IfElseStmtsMustUseBraces. Avoid using if...else statements without using surrounding braces. If the code formatting or indentation is lost then it becomes difficult to separate th... (3 violations)	
▶ LinguisticNaming. This rule finds Linguistic Naming Antipatterns. It checks for fields, that are named, as if they should be boolean but have a different t... (3 violations)	
▶ IdenticalCatchBranches. Identical 'catch' branches use up vertical space and increase the complexity of code without adding functionality. It's better styl... (8 violations)	
▶ FieldDeclarationsShouldBeAtStartOfClass. Fields should be declared at the top of the class, before any method declarations, constructors, initializers or inner classes. (4 violations)	
▶ AvoidProtectedMethodInFinalClassNotExtending. Do not use protected methods in most final classes since they cannot be subclassed. This should only be allowed in final classes that... (1 violation)	
▶ UseUnderscoresInNumericLiterals. Since Java 1.7, numeric literals can use underscores to separate digits. This rule enforces that numeric literals above a ... (1 violation)	
▶ UnnecessaryFullyQualifiedName. Import statements allow the use of non-fully qualified names. The use of a fully qualified name which is covered by an import statement is redundan... (1 violation)	
▶ SuspiciousConstantFieldName. Field names using all uppercase characters - Sun's Java naming conventions indicating constants - should be declared as final. This rule is deprecate... (9 violations)	

▼ design (1416 violations)	
▶ SingularField. Fields whose scopes are limited to just single methods do not rely on the containing object to provide them to other methods. They may be better implemented as local... (21 violations)	
▶ CyclomaticComplexity. The complexity of methods directly affects maintenance costs and readability. Concentrating too much decisional logic in a single method makes its behaviour ... (54 violations)	
▶ AvoidCatchingGenericException. Avoid catching generic exceptions such as NullPointerException, RuntimeException, Exception in try-catch block (44 violations)	
▶ LawOfDemeter. The Law of Demeter is a simple rule, that says "only talk to friends". It helps to reduce coupling between classes or objects. See also the references: Andrew ... (995 violations)	
▶ ExcessiveMethodLength. When methods are excessively long this usually indicates that the method is doing more than its name/signature might suggest. They also become challenging f... (7 violations)	
▶ NcssCount. This rule uses the NCSS (Non-Commenting Source Statements) metric to determine the number of lines of code in a class, method or constructor. NCSS ignores comments, bla... (9 violations)	
▶ NPathComplexity. The NPath complexity of a method is the number of acyclic execution paths through that method. While cyclomatic complexity counts the number of decision points in... (16 violations)	
▶ TooManyMethods. A class with too many methods is probably a good suspect for refactoring, in order to reduce its complexity and find a way to have more fine grained objects. ... (10 violations)	
▶ ModifiedCyclomaticComplexity. Complexity directly affects maintenance costs is determined by the number of decision points in a method plus one for the method entry. The decisio... (35 violations)	
▶ ExcessiveClassLength. Excessive class file lengths are usually indications that the class may be burdened with excessive responsibilities that could be provided by external class... (3 violations)	
▶ TooManyFields. Classes that have too many fields can become unwieldy and could be redesigned to have fewer fields, possibly through grouping related fields in new objects. For ex... (6 violations)	
▶ GodClass. The God Class rule detects the God Class design flaw using metrics. God classes do too many things, are very big and overly complex. They should be split apart to be mor... (7 violations)	
▶ ImmutableField. Identifies private fields whose values never change once object initialization ends either in the declaration of the field or by a constructor. This helps in conv... (108 violations)	
▶ NcssMethodCount. This rule uses the NCSS (Non-Commenting Source Statements) algorithm to determine the number of lines of code for a given method. NCSS ignores comments, and count... (5 violations)	
▶ StdCyclomaticComplexity. Complexity directly affects maintenance costs is determined by the number of decision points in a method plus one for the method entry. The decision poi... (42 violations)	
▶ UseUtilityClass. For classes that only have static methods, consider making them utility classes. Note that this doesn't apply to abstract classes, since their subclasses may well... (16 violations)	
▶ DataClass. Data Classes are simple data holders, which reveal most of their state, and without complex functionality. The lack of functionality may indicate that their behaviour ... (4 violations)	
▶ ExcessiveParameterList. Methods with numerous parameters are a challenge to maintain, especially if most of them share the same data type. These situations usually denote the need ... (1 violation)	
▶ SimplifyBooleanReturns. Avoid unnecessary if-then-else statements when returning a boolean. The result of the conditional test can be returned instead. (1 violation)	
▶ UselessOverridingMethod. The overriding method merely calls the same method defined in a superclass. (1 violation)	
▶ AvoidDeeplyNestedIfStmts. Avoid creating deeply nested if-then statements since they are harder to read and error-prone to maintain. (5 violations)	
▶ ExcessivePublicCount. Classes with large numbers of public methods and attributes require disproportionate testing effort since combinational side effects grow rapidly and increa... (3 violations)	
▶ ExceptionAsFlowControl. Using Exceptions as form of flow control is not recommended as they obscure true exceptions when debugging. Either add the necessary validation or use an a... (8 violations)	
▶ AvoidThrowingNullPointerException. Avoid throwing NullPointerExceptions manually. These are confusing because most people will assume that the virtual machine threw it. To avoid... (4 violations)	
▶ AvoidUncheckedExceptionsInSignatures. A method or constructor should not explicitly declare unchecked exceptions in its 'throws' clause. Java doesn't force the caller to handle an... (2 violations)	
▶ CollapsibleIfStatements. Sometimes a 'null' to a variable (outside of its declaration) is usually bad form. Sometimes, this type of assignment is an indication that the programmer doesn't... (3 violations)	
▶ AvoidThrowingRawExceptionTypes. Avoid throwing certain exception types. Rather than throw a raw RuntimeException, Throwable, Exception, or Error, use a subclassed exception or err... (2 violations)	
▶ SimplifyBooleanExpressions. Avoid unnecessary comparisons in boolean expressions, they serve no purpose and impact readability. (2 violations)	
▶ SignatureDeclareThrowsException. A method/constructor shouldn't explicitly throw the generic java.lang.Exception, since it is unclear which exceptions that can be thrown from the ... (2 violations)	

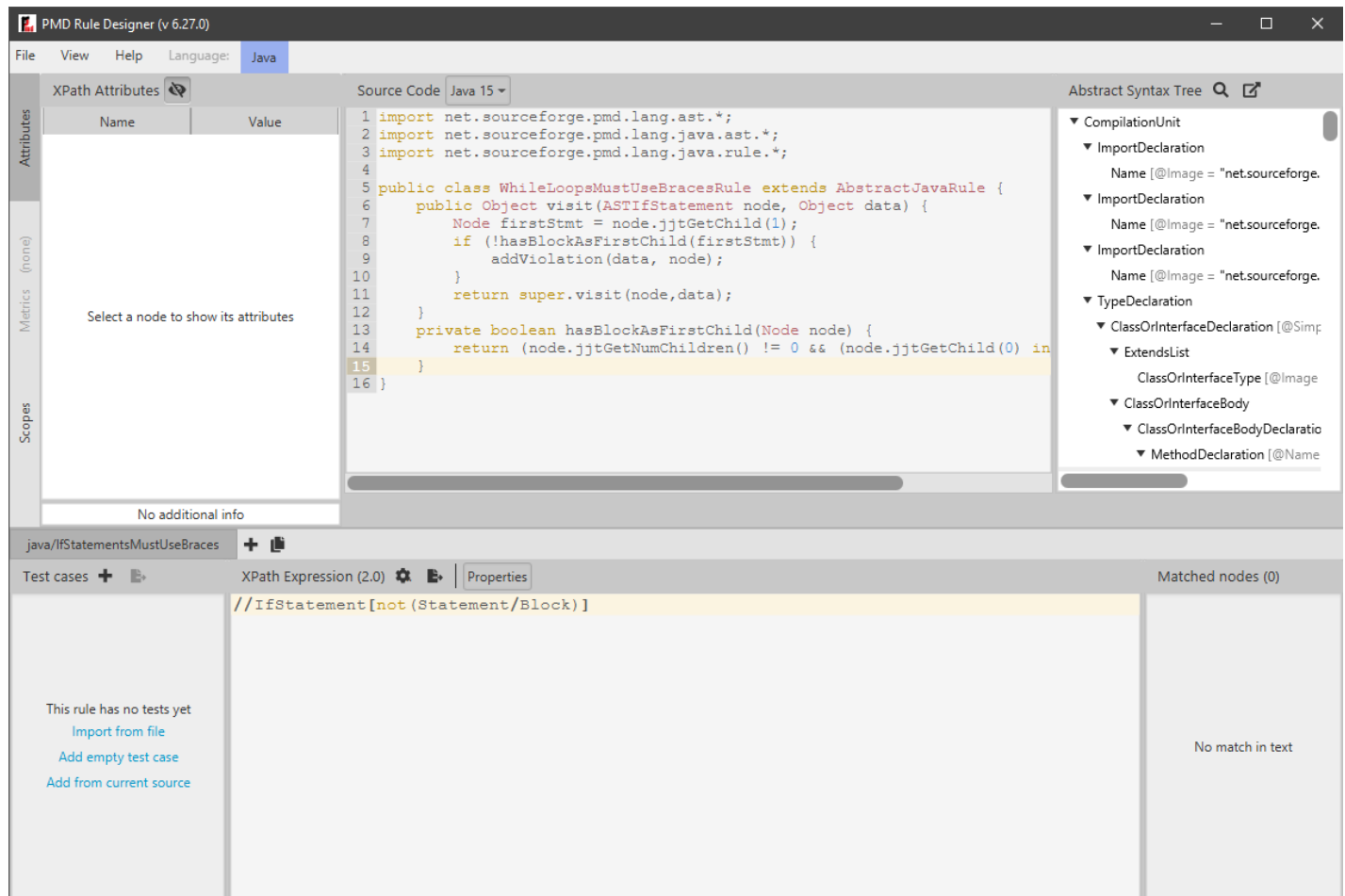
▼ documentation (1280 violations)	
▶ CommentRequired. Denotes whether javadoc (formal) comments are required (or unwanted) for specific language elements. (933 violations)	
▶ CommentSize. Determines whether the dimensions of non-header comments found are within the specified limits. (319 violations)	
▶ UncommentedEmptyConstructor. Uncommented Empty Constructor finds instances where a constructor does not contain statements, but there is no comment. By explicitly commenting empty... (5 violations)	
▶ UncommentedEmptyMethodBody. Uncommented Empty Method Body finds instances where a method body does not contain statements, but there is no comment. By explicitly commenting empty ... (23 violations)	

▼ errorprone (641 violations)	
▶ CompareObjectsWithEquals. Use equals() to compare object references (14 violations)	
▶ AssignmentToNonFinalStatic. Identifies a possible unsafe usage of a static field. (1 violation)	
▶ DataflowAnomalyAnalysis. The dataflow analysis tracks local definitions, undefinitions and references to variables on different paths on the data flow. From those informations ther... (170 violations)	
▶ UseEqualsToCompareStrings. Using '==' or '!=' to compare strings only works if intern version is used on both sides. Use the equals() method instead. (6 violations)	
▶ NullAssignment. Assigning a 'null' to a variable (outside of its declaration) is usually bad form. Sometimes, this type of assignment is an indication that the programmer doesn't... (24 violations)	
▶ BeanMembersShouldSerialize. If a class is a bean, or is referenced by a bean directly or indirectly it needs to be serializable. Member variables need to be marked as transient, ... (226 violations)	
▶ AvoidLiteralsInIfCondition. Avoid using hard-coded literals in conditional statements. By declaring them as static variables or private members with descriptive names maintainabil... (65 violations)	
▶ AvoidDuplicateLiterals. Code containing duplicate String literals can usually be improved by declaring the String as a constant field. (42 violations)	
▶ EmptyIfStmt. Empty If Statement finds instances where a condition is checked but nothing is done about it. (1 violation)	
▶ CloneMethodReturnTypeMustMatchClassName. If a class implements cloneable the return type of the method clone() must be the class name. That way, the caller of the clone method doe... (4 violations)	
▶ ProperCloneImplementation. Object clone() should be implemented with super.clone(). (2 violations)	
▶ AvoidFieldNameMatchingMethodName. It can be confusing to have a field name with the same name as a method. While this is permitted, having information (field) and actions (method... (3 violations)	
▶ ImportFromSamePackage. There is no need to import a type that lives in the same package. (6 violations)	
▶ CloneMethodMustImplementCloneable. The method clone() should only be implemented if the class implements the Cloneable interface with the exception of a final method that only thr... (3 violations)	
▶ AvoidFieldNameMatchingTypeName. It is somewhat confusing to have a field name matching the declaring class name. This probably means that type and/or field names should be chosen ... (1 violation)	
▶ MissingSerialVersionUID. Serializable classes should provide a serialVersionUID field. The serialVersionUID field is also needed for abstract base classes. Each individual class i... (9 violations)	
▶ ConstructorCallsOverridableMethod. Calling overridable methods during construction poses a risk of invoking methods on an incompletely constructed object and can be difficult to d... (18 violations)	
▶ UseLocaleWithCaseConversions. When doing 'String::toLowerCase()/toUpperCase()' conversions, use an explicit locale argument to specify the case transformation rules.... (1 violation)	
▶ AvoidCatchingNPE. Code should never throw NullPointerExceptions under normal circumstances. A catch block may hide the original error, causing other, more subtle problems later ... (2 violations)	
▶ CloneMethodMustBePublic. The java Manual says "By convention, classes that implement this interface should override Object.clone (which is protected) with a public method." ... (2 violations)	
▶ AvoidCatchingThrowable. Catching Throwable errors is not recommended since its scope is very broad. It includes runtime issues such as OutOfMemoryError that should be exposed and... (1 violation)	
▶ AvoidDecimalLiteralsInBigDecimalConstructor. One might assume that the result of "new BigDecimal(0.1)" is exactly equal to 0.1, but it is actually equal to .100000000000000055511... (1 violation)	
▶ EmptyCatchBlock. Empty Catch Block finds instances where an exception is caught, but nothing is done. In most circumstances, this swallows an exception which should either be ac... (8 violations)	
▶ MissingBreakInSwitch. Switch statements without break or return statements for each case option may indicate problematic behaviour. Empty cases are ignored as these indicate an in... (3 violations)	
▶ EqualsNull. Tests for null should not use the equals() method. The '==' operator should be used instead. (2 violations)	
▶ UseProperClassLoader. In J2EE, the getClassLoader() method might not work as expected. Use Thread.currentThread().getContextClassLoader() instead. (1 violation)	
▶ CloneThrowsCloneNotSupportedException. The method clone() should throw a CloneNotSupportedException. (1 violation)	
▶ EmptyStatementNotInLoop. An empty statement (or a semicolon by itself) that is not used as the sole body of a 'for' or 'while' loop is probably a bug. It could also be a double ... (1 violation)	
▶ CloseResource. Ensure that resources (like 'java.sql.Connection', 'java.sql.Statement', and 'java.sql.ResultSet' objects and any subtype of 'java.lang.AutoCloseable') are always c... (22 violations)	
▶ AssignmentInOperand. Avoid assignments in operands (1 violation)	

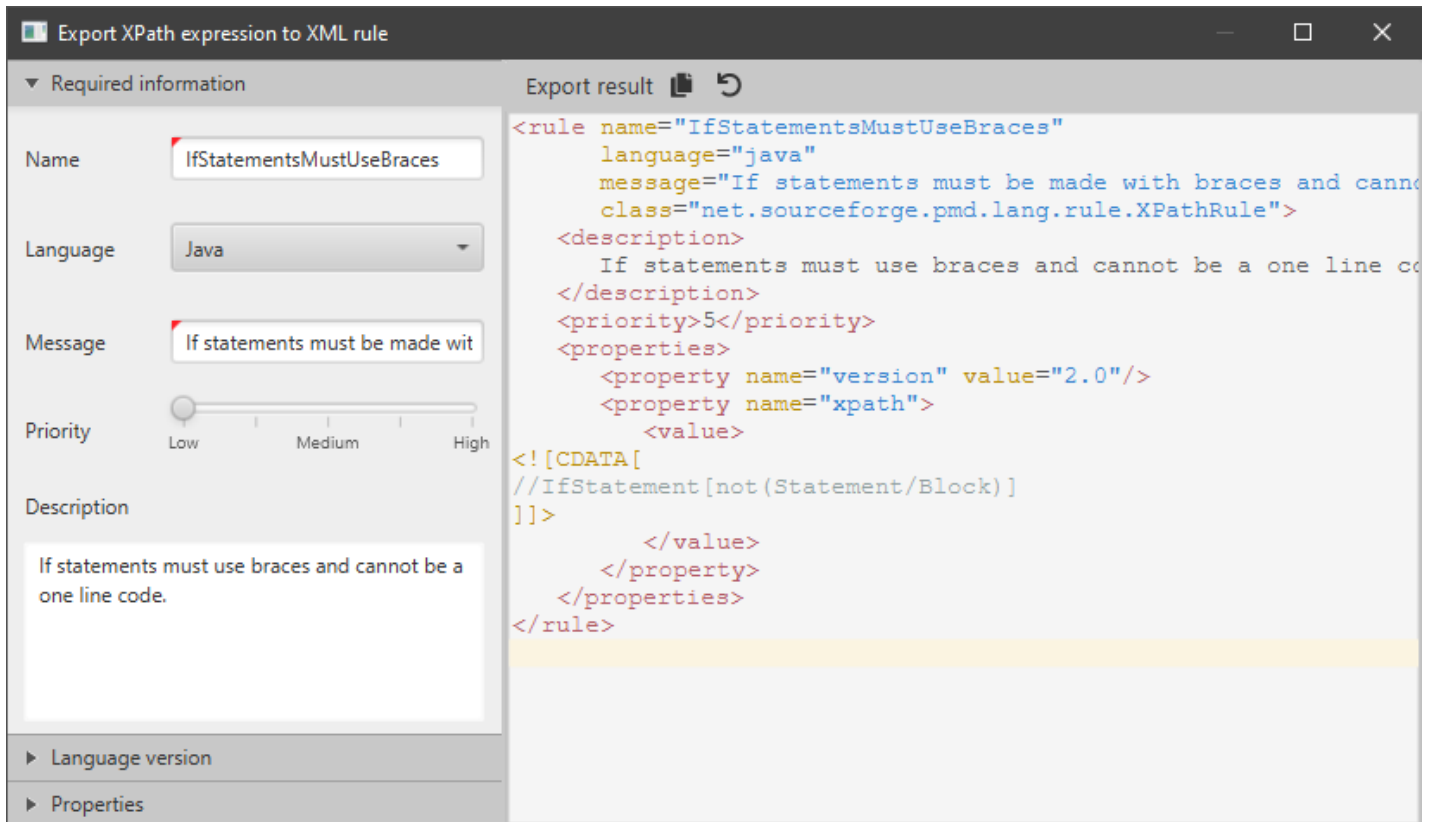


Kaip matome iš rezultatų, yra padaryta daug pažeidimų įvairiose statinės kodo analizės kategorijose. Jeigu šį įrankį būtume naudoję nuo programos kūrimo pradžios, būtume galėję nesunkiai šių klaidų išvengti reguliariai atlikdami statinę kodo analizę, tačiau dabar tai atlikti yra gan sudėtinga, kadangi yra daug programinio kodo kurį reiktų keisti.

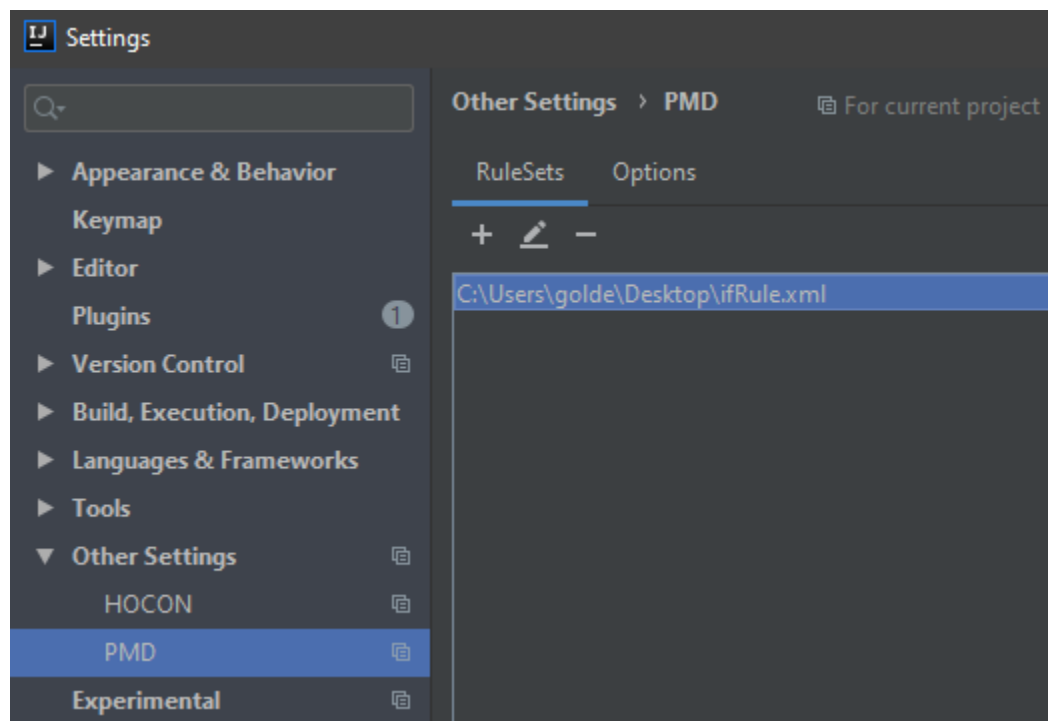
Naudodami PDM Designer sukūrėme savo taisyklę, kuri tikrina ar „if“ sąlyga turi skliaustus ar ne:



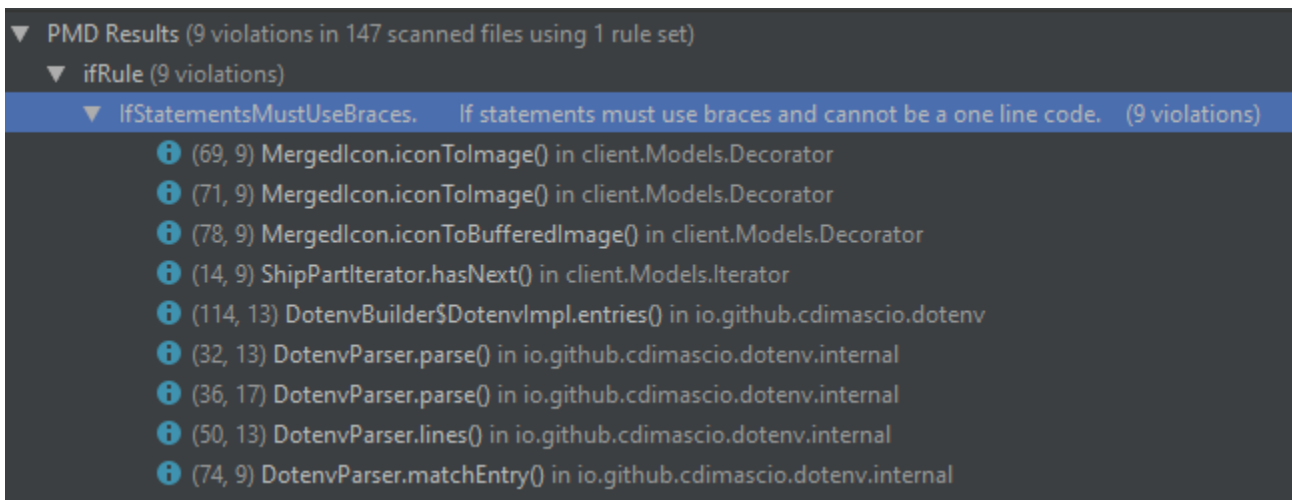
Toliau aprašome taisyklę ir eksportuojame šią taisyklę kaip xml failą:



Pridėjome šią taisyklę IntelliJ:



Paleidome šią taisyklę su PDM ir gavome šiuos rezultatus:



Patikriname programos kode ar taisyklė yra tikrai pažeista:

Lentelė 2 Pažeista mūsų sukurta statinės kodo analizės taisyklė

Failo pavadinimas	MergedIcon.java
Kodo fragmentas	
<pre> 68 @ 69 public static Image iconToImage(Icon icon) { 70 if (icon == null) 71 return null; 72 if (icon instanceof ImageIcon) 73 return ((ImageIcon) icon).getImage(); 74 75 return iconToBufferedImage(icon); 76 } </pre>	
Kodo eilutė faile	69,71
Klaidos kategorija	Kodo formatavimas
Klaidos apibūdinimas	If sąlygoje nėra skliaustelių
Galimas sprendimas	Uždėti skliaustelius šiose if sąlygose

6. Išvados

Pasidarėme kodo analizę tiesiog peržvelgiant kodą, radome klaidų. Taip pat įsidiegėme papildomą IDEA aplinkos pluginą PMD kuris mums leido atlikti statinę kodo analizę. Jos pagalba, paaiškėjo, kad turime daug klaidų. To pasėkoje, suvokėme, kad jau nuo projekto pradžios būtų buvę naudinga ir tikslinga naudoti tokį įrankį kaip PMD, kad šios klaidos būtų išvengtos iškart. Taip pat laboratorinio darbo metu išmokome susikurti papildomų taisyklių PMD įrankiui. .nuo projekto kūrimo pradžios. Taip pat išmokome susikurti papildomų taisyklių, kadangi, taip pat išmokome susikurti papildomų taisyklių.