William Cosulich

Lauren Medley

Erica Muschelli

Jose Quinchuela

CoSC 603

FreeCol tactital test plan

Table of Contents

Introduction

Features that will be tested

Approach to Testing/Improving Software

1. Activities
2. Criteria
3. Testing tools

Criteria for pass/fail

Introduction

Features that will be tested

Approach to Testing/Improving Software

1. Activities
2. Criteria
3. Testing tools

Criteria for pass/fail

Test Deliverables/status communication documents

1. Test plan
2. Test design specifications
3. Test case specifications
4. Test process
5. Test logs.
6. Test trouble reports
7. Test summary reports
8. Test input data and test output data(or where they are located)

Environmental Needs

1. Hardware
2. Software
3. Any other software/supplies needed to run the test.
4. Mode of usage (stand-alone, web based)
5. Test tools needed.
6. Any other testing needs. (publications)

Responsibilities

Schedule

**Introduction**

FreeCol is an Open Source version of Colonization. Colonization is heavily based on Civilization which some consider to be the best turn-based strategy game for the PC in the history of mankind. FreeCol differs from the original game in two regards: it supports multiplayer games and uses an isometric map. At some point in the future, the game might also add support for rectangular tiles similar to those used in the original game. Our team will be focusing on several suggested packages: client, control, gui and others. Our team will select and prioritize 20 classes to improve the quality of the code as well as to include test cases within the packages above mentioned. The test plan will help the team to report on refactoring activities, test deliverables, test tools and also identify overall team responsibilities.

**Features to be tested.**

The team decided on the 20 classes listed below which were prioritized according to severe bugs found, high code violations found, high code complexity, methods too long, styling and documentation and code coverage.

|  |  |  |
| --- | --- | --- |
| **Feature Number** | **Package Name** | **Class Name** |
| **1** | net.sf.freecol.metaserver | MetaServer |
| **2** | net.sf.freecol.common.model | Limit |
| **3** | net.sf.freecol.common.option | RangeOption |
| **4** | net.sf.freecol.common.option | SelectOption |
| **5** | net.sf.freecol.common.server.model | Unit |
| **6** | net.sf.freecol.server.ai | REFAIPlayer |
| **7** | net.sf.freecol.server.model | ServerUnit |
| **8** | net.sf.freecol.server.model | ServerColony |
| **9** | net.sf.freecol.client.gui.panel | DragListener |
| **10** | net.sf.freecol.client.gui.panel | BuildingDetailPanel |
| **11** | net.sf.freecol.client.gui.panel | BuildingToolTip |
| **12** | net.sf.freecol | FreeCol |
| **13** | net.sf.freecol.client | ClientOptions |
| **14** | net.sf.freecol.client.control | InGameController |
| **15** | net.sf.freecol.client.gui | FontLibrary |
| **16** | net.sf.freecol.client.gui | ChoiceItem |
| **17** | net.sf.freecol.client.gui | FrameMontionListener |
| **18** | net.sf.freecol.client.gui | TerrainCursor |
| **19** | net.sf.freecol.common | FreeColSeed |
| **20** | net.sf.freecol.client.common | ServerInfo |
|  |  |  |

**Approach to Testing**

The team’s approach to testing was prioritized testing according to severe bugs found, high code violations found, high code complexity, methods too long, styling issues and documentation and code coverage. We used the following tools to help prioritize what classes would be tested. : FindBugs, PMD, Google Code Pro, JDeodorant, CheckStyle, JAutoDoc, EclEmma, and JUnit Testing.

Also the team added code coverage to classes other than the 20 that we prioritized that did had of 0% coverage

Feature1: MetaServer

Activities: Fix bug that was found, fix high code violation, lowered the Cyclomatic Complexity, Refactored Methods that were too long, Add JDoc documentation for the class.

Test Tools: FindBugs, PMD, JDeorderant, Google Code Pro, Eclipse refactoring tools, JAutoDoc

Feature2: Limit

Activities: Fix bug that was found, fix high code violation, lower the Cyclomatic Complexity, Refactored Methods that were too long, Add JDoc documentation for the class.

Test Tools: FindBugs, PMD, JDeorderant, Google Code Pro, Eclipse refactoring tools, JAutoDoc

Feature3: RangeOption

Activities: Fix bug that was found, fix high code violation, Add JDoc documentation for the class.

Test Tools: FindBugs, PMD, JAutoDoc

Feature4: SelectOption

Activities: Fix bug that was found, fix high code violation, Add JDoc documentation for the class.

Test Tools: FindBugs, PMD, JAutoDoc

Feature5: Unit

Activities: Fix high code violation, Add JDoc documentation for the class.

Test Tools: PMD, JAutoDoc

Feature6: REFAIPlayer

Activities: Fix high code violation, lower the Cyclomatic Complexity, Add JDoc documentation for the class.

Test Tools: PMD, CodePro, Eclipse refactoring tools, JAutoDoc

Feature7: ServerUnit

Activities: Fix high code violation, lower the Cyclomatic Complexity, Add JDoc documentation for the class.

Test Tools: PMD, CodePro, Eclipse refactoring tools, JAutoDoc

Feature8: Draglistener

Activities: Lower the Cyclomatic Complexity, Fix high code violation, Refactor Method Too Large, Add JDoc documentation for the class.

Test Tools: CodePro, Eclipse refactoring tools, JDeodorant, JAutoDoc

Feature9: ServerColony

Activities: Fix high code violation, Add JDoc documentation for the class.

Test Tools: PMD, JAutoDoc

Feature10: BuildingDetailPanel

Activities: Lower the Cyclomatic Complexity, Add JDoc documentation for the class.

Test Tools: CodePro, Eclipse refactoring tools, JAutoDoc

Feature10: BuildingToolTip

Activities: Lower the Cyclomatic Complexity, Add JDoc documentation for the class.

Test Tools: CodePro, Eclipse refactoring tools, JAutoDoc

Feature12: FreeCol

Activities: Add JUnit Test Cases, Clean up style issues, Add JDoc documentation for the class.

Test Tools: CodePro, JUnit, Eclipse refactoring tools, CheckStyle, Eclipse Formatter tool, JAutoDoc

Feature13: ClientOption

Activities: Add JUnit Test Cases, Clean up style issues, Add JDoc documentation for the class.

Test Tools: CodePro, JUnit, Eclipse refactoring tools, CheckStyle, Eclipse Formatter tool, JAutoDoc

Feature14: InGameController

Activities: Add JUnit Test Cases, Clean up style issues, Add JDoc documentation for the class.

Test Tools: CodePro, JUnit, Eclipse refactoring tools, CheckStyle, Eclipse Formatter tool, JAutoDoc

Feature15: FontLibrary

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

Feature16: ChoiceItem

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

Feature17: FrameMontionListener

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

Feature18: TerrainCursor

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

Feature19: FreeColSeed

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

Feature20: ServerInfo

Activities: Add JUnit Test Cases and bring up the code coverage, Add JDoc documentation for the class

Test Tools: CodePro, JUnit, EclEmma, JAutoDocs

**Extra Features: All activities for extra features:**

NationSummary.java

Operand.java

Limit.java

IndianNationType.java

Disaster.java

NationOptions.java

Nation.java

NationType.java

BuidableType.java

AbstractUnit.java

ResourceType.java

AbstractGoods.java

Stance.java

Tension.java

Activities: Use eclEmma to improve code coverage and JUnit to generate test cases.

Test Tools: Eclipse refactoring tools, JUnit, Eclmma

**Criteria for Pass/Fail**

The criteria for passing was based on a number of items that also depended on what type of issues that were being addressed on the classes. Not all criteria for pass/fail was used for each class as not tests/ improvements were performed on all classes used. For the classes with bugs, criteria for passing was to removal of the severe bugs found while criteria for failure was not fixing the bugs. For the classes with high code violations, the criteria for passing removal of the code violation while the criteria for failure was to leave in the violations, For classes that we were lowering the Cyclomatic Complexity for, the criteria for passing was to have Cyclomatic Complexity be reduced at all while we did not have any failure criteria as we found that this could not apply to all class as their Cyclomatic Complexity was low to begin. For classes that had methods that were too long, the criteria for passing was the making their methods were refactored into smaller ones. Their criteria for failure was not completely refactoring their methods until their no-longer refactorable. Adding documentation pass or fail criteria was more of the we added the JDoc for the class or we didn’t. For classes that we added JUnit test cases for, the passing criteria was that we made sure that all the test cases passed while the criteria for failure was that some of the classes failed. For classes that we had added code coverage for the pass criteria was that increased the coverage to over 50% while the failure criteria was that the coverage was less than 50%.

Feature1: MetaServer

Pass Criteria: Fixed bug found, high code violation fixed, lowered the Cyclomatic Complexity by any percent, refactored all Methods that were too long, added JDoc documentation.

Failure Criteria: Bug found not fixed, high code violation not fixed, left methods too long, didn’t add any JDoc documentation.

Feature2: Limit

Pass Criteria: Fixed bug found, high code violation fixed, lowered the Cyclomatic Complexity by any percent, refactored all Methods that were too long, added JDoc documentation.

Failure Criteria: Bug found not fixed, high code violation not fixed, left methods too long, didn’t add any JDoc documentation.

Feature3: RangeOption

Pass Criteria: Fixed bug found, high code violation fixed, added JDoc documentation.

Failure Criteria: Bug found not fixed, high code violation not fixed, didn’t add any JDoc documentation.

Feature4: SelectOption

Pass Criteria: Fixed bug found, high code violation fixed, added JDoc documentation.

Failure Criteria: Bug found not fixed, high code violation not fixed , didn’t add any JDoc documentation.

Feature5: Unit

Pass Criteria: High code violation fixed, added JDoc documentation.

Failure Criteria: high code violation not fixed, left methods too long, didn’t add any JDoc documentation.

Feature6: REFAIPlayer

Pass Criteria: High code violation fixed, lowered the Cyclomatic Complexity by any percent, added JDoc documentation.

Failure Criteria: High code violation not fixed, didn’t add any JDoc documentation.

Feature7: ServerUnit

Pass Criteria: high code violation fixed, lowered the Cyclomatic Complexity by any percent, added JDoc documentation.

Failure Criteria: high code violation not fixed, didn’t add any JDoc documentation.

Feature8: Draglistener

Pass Criteria: high code violation fixed, lowered the Cyclomatic Complexity by any percent, refactored all Methods that were too long, added JDoc documentation.

Failure Criteria: high code violation not fixed, left methods too long, didn’t add any JDoc documentation.

Feature9: ServerColony

Pass Criteria: high code violation fixedadded JDoc documentation.

Failure Criteria: high code violation not fixed, didn’t add any JDoc documentation.

Activities: Fix high code violation, Add JDoc documentation for the class.

Test Tools: PMD, JAutoDoc

Feature10: BuildingDetailPanel

Pass Criteria: lowered the Cyclomatic Complexity by any percent, added JDoc documentation.

Failure Criteria: didn’t add any JDoc documentation.

Feature10: BuildingToolTip

Pass Criteria: lowered the Cyclomatic Complexity by any percent, added JDoc documentation.

Failure Criteria: didn’t add any JDoc documentation.

Feature12: FreeCol

Pass Criteria: All test cases passed, Cleaned style issues, added JDoc documentation.

Failure Criteria: Some test cases failed, Leave style issues found, didn’t add any JDoc documentation.

Activities: Add JUnit Test Cases, Clean up style issues, Add JDoc documentation for the class.

Feature13: ClientOption

Pass Criteria: All test cases passed, Cleaned style issues, added JDoc documentation.

Failure Criteria: Some test cases failed, Leave style issues found, didn’t add any JDoc documentation.

Feature14: InGameController

Pass Criteria: All test cases passed, Cleaned style issues, added JDoc documentation.

Failure Criteria: Some test cases failed, Leave style issues found, didn’t add any JDoc documentation.

Feature15: FontLibrary

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

Feature16: ChoiceItem

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

Feature17: FrameMontionListener

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

Feature18: TerrainCursor

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

Feature19: FreeColSeed

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

Feature20: ServerInfo

Pass Criteria: All test cases passed, increased coverage to over 50%, added JDoc documentation.

Failure Criteria: Some test cases failed, coverage less than 50%, didn’t add any JDoc documentation.

**Test Deliverables**

The following items will be turned in upon completion of software improvements:

1. **Test plan**

Improved and complete FreeCol test plan.

1. **Test process**

Statement coverage report

1. **Test logs.**

Screenshots (coverage reports)

1. **Test trouble reports**

Test trouble reports are handled within GitHub in the issue generation feature of GitHub.

1. **Test summary reports**

Summary reports. We can have summary of how we tested the code. See the Schedule part for this.

1. **Test input data and test output data(or where they are located)**

See criteria for pass/fail that refer to the test input and output data.

**Environmental Needs**

1. **Hardware**

Dell laptop inspiron 1525 series

Dell laptop

HP laptop

Mac laptop

1. **Software**

Eclipse IDE Mars, Luna,

Java rei 1.8.0

1. **Any other software/supplies needed to run the test.**

Eclipse

Java Version 7 and above

Windows Vista and above

Mac O.S

GitHub

1. **Mode of usage (stand-alone, web based)**

Stand alone.

1. **Test tools needed.**

Google Code Pro Metrics

Code Pro Analytix

FindBugs

CheckStyle

PMD

JUnit

EclEmma

JAutoDocs,

Eclipse Formatter tool,

Eclipse Refactor tool,

1. **Any other testing needs. (publications)**

FreeCol docs, FreeCol FQA, FreeCol test cases.

**Responsibilities**

**Identify the groups responsible for all aspects of testing and correcting problems**

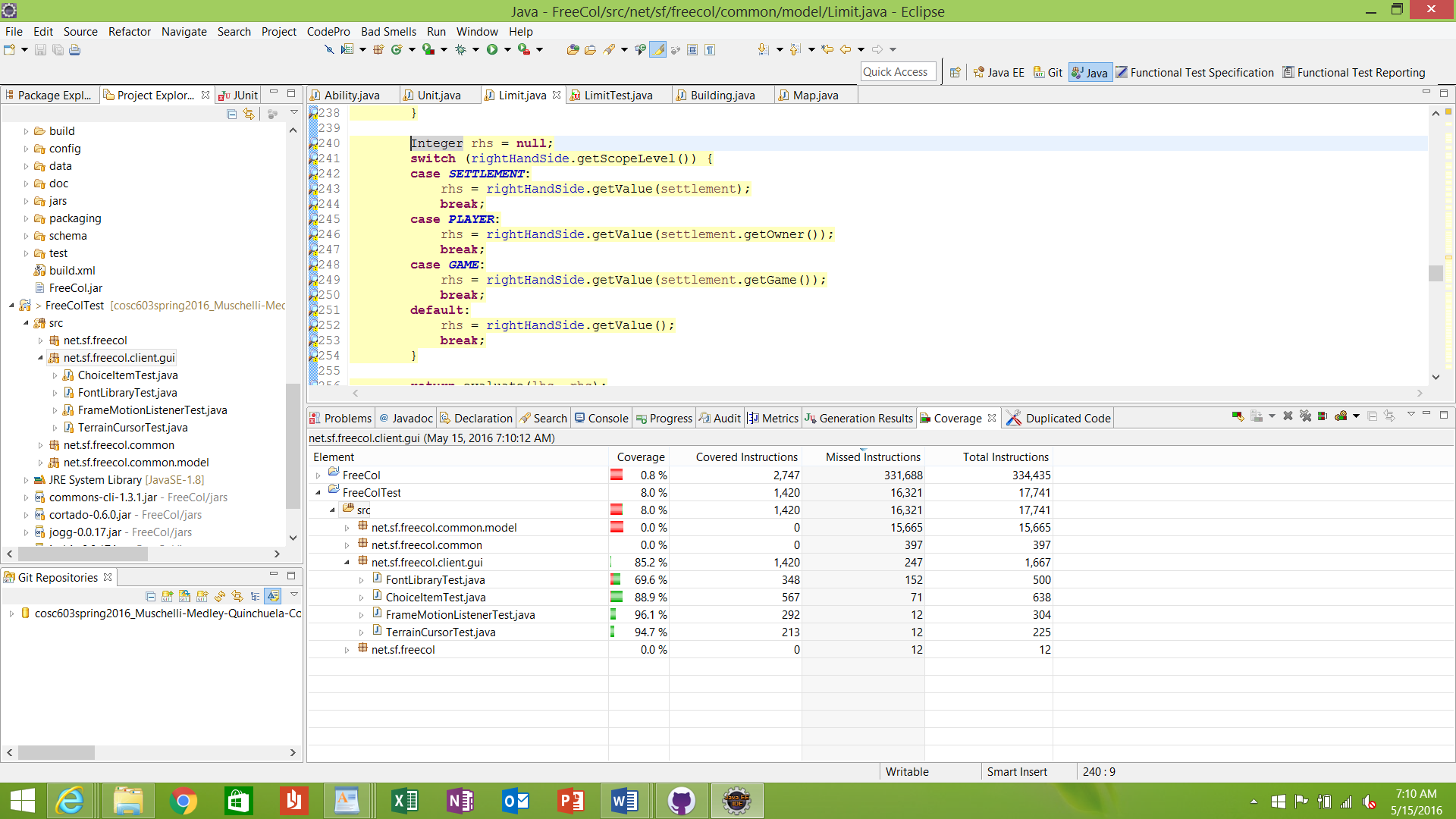
**Erica:** Setting up the repository and uploading the project on the repository for all to use.

**Lauren/Erica:**

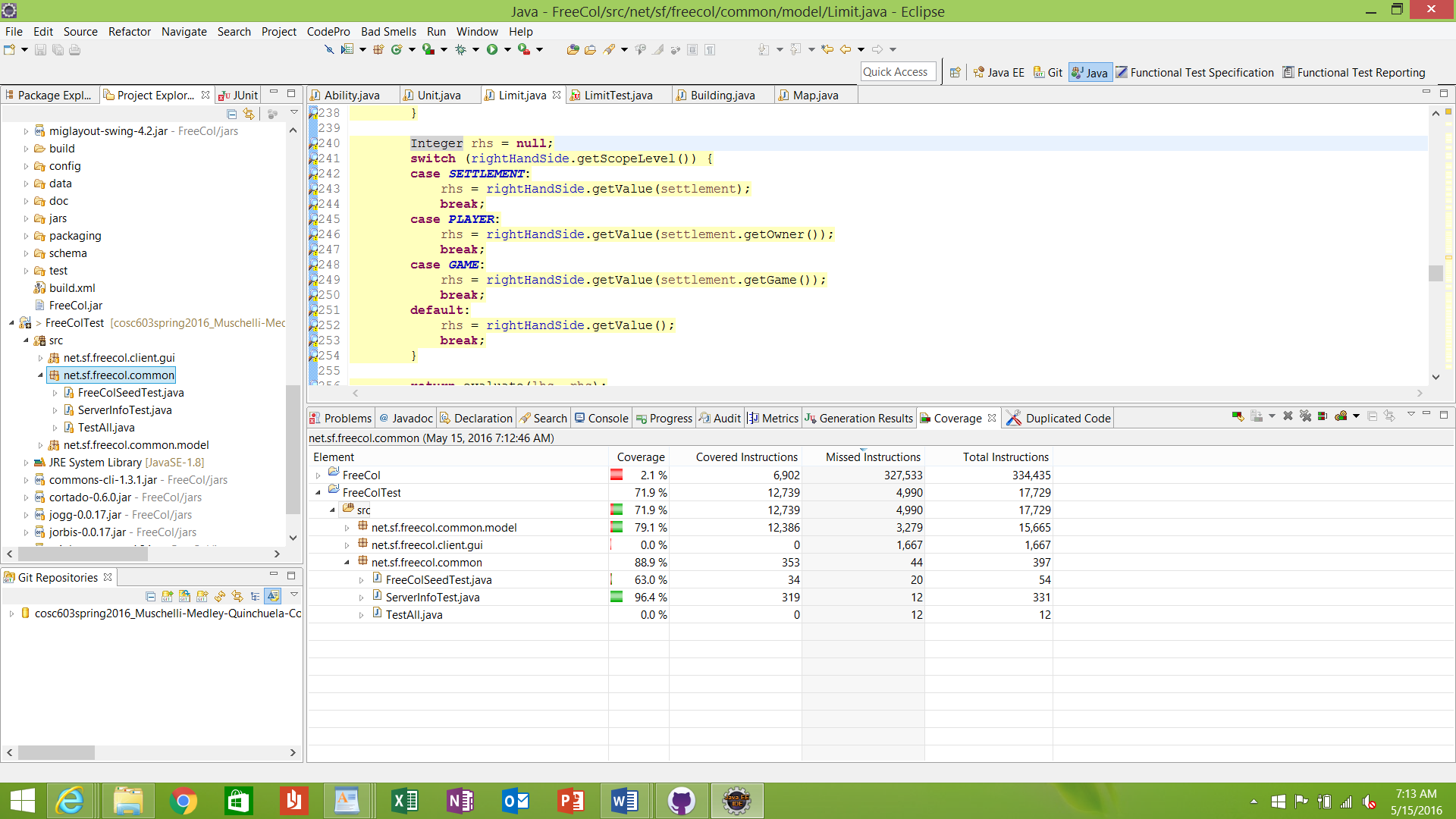
Erica with assistance from Lauren work on the majority of the JUnit test cases that were created for the project. They also worked to increase the coverage for a number of classes. They selected files where coverage was 0.0% to show code coverage improvement when developing our test cases. Some GUI files were selected but they felt that those classes contained within the “common” files were a bit more significant since they seemed to be used the most for our test case development.

Test cases created to improve branch code coverage for the following:

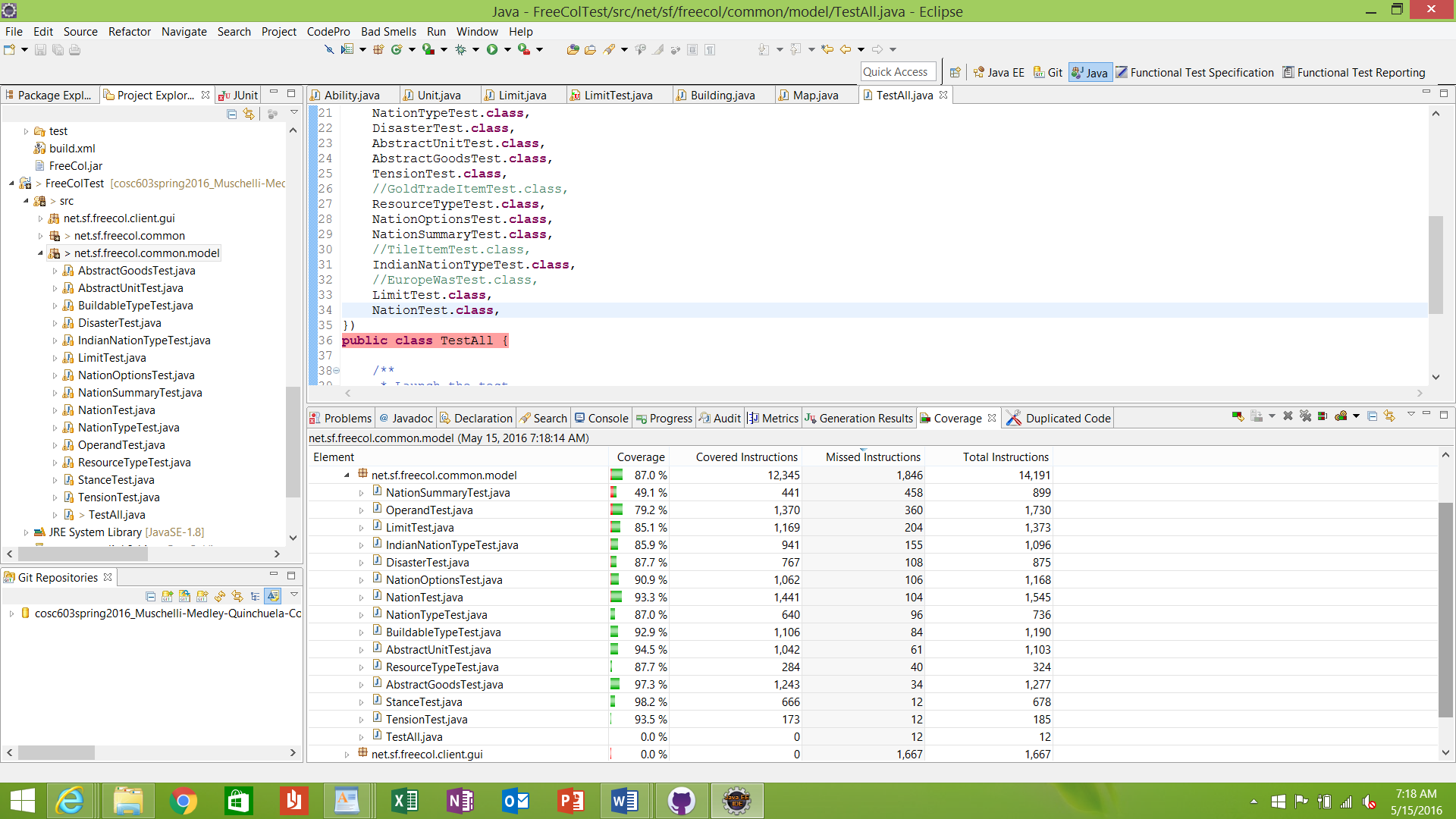
1. Net.sf.freecol.client.gui/ChoiceItem.java: Achieved 88.9% branch coverage.
2. Net.sf.freecol.client.gui/FontLibrary.java: Achieved 69.6.% branch coverage.
3. net.sf.freecol.client.gui/FrameMotionListener.java: Achieved 96.1% branch coverage.
4. net.sf.freecol.client.gui/TerrainCursor.java: Achieved 94.7% branch coverage.



1. net.sf.freecol.common/FreeColSeed.java: Achieved 63.0% branch coverage.
2. Net.sf.freecol.common/ServerInfo.java: Achieved 96.4% branch coverage.

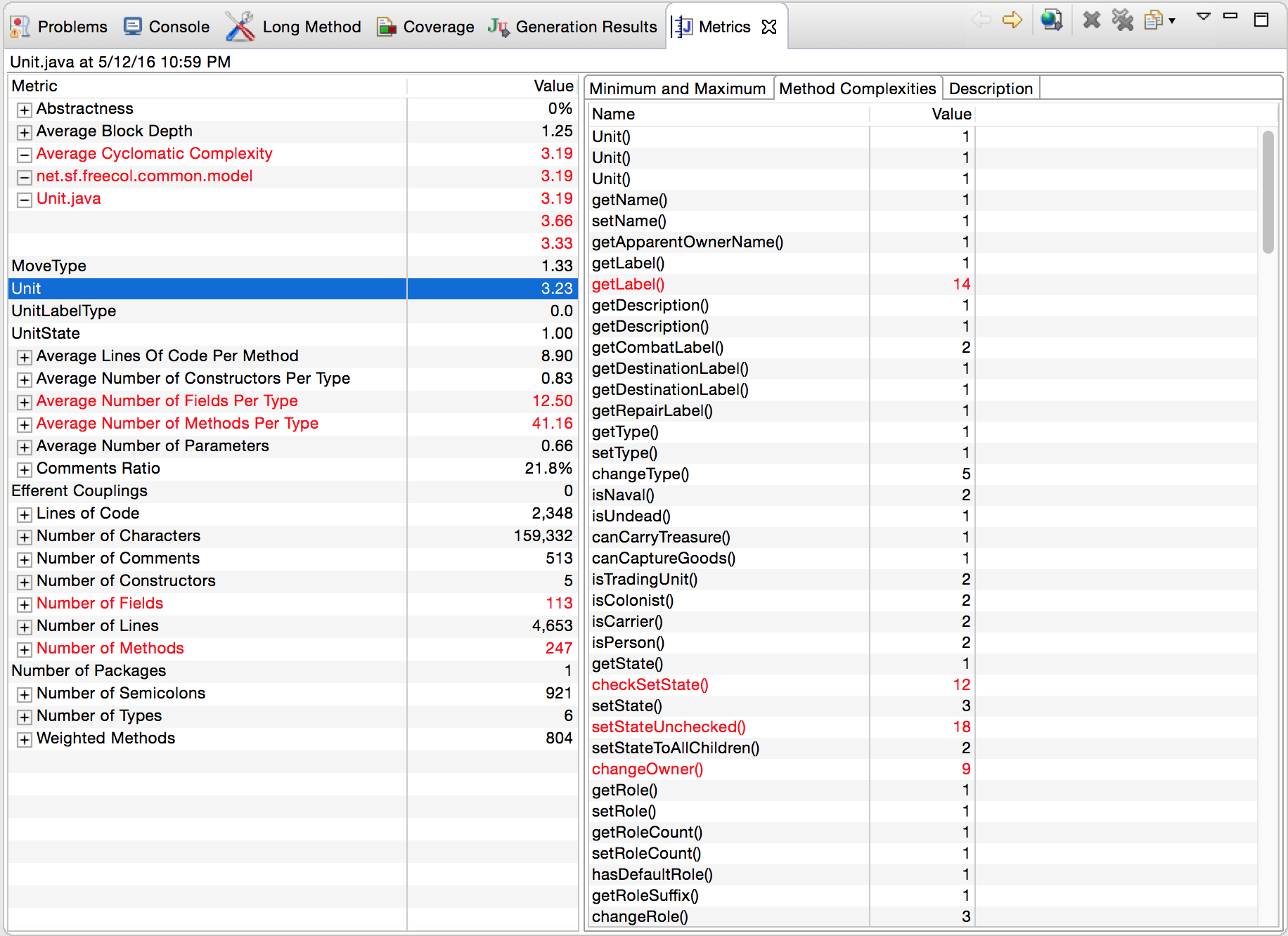
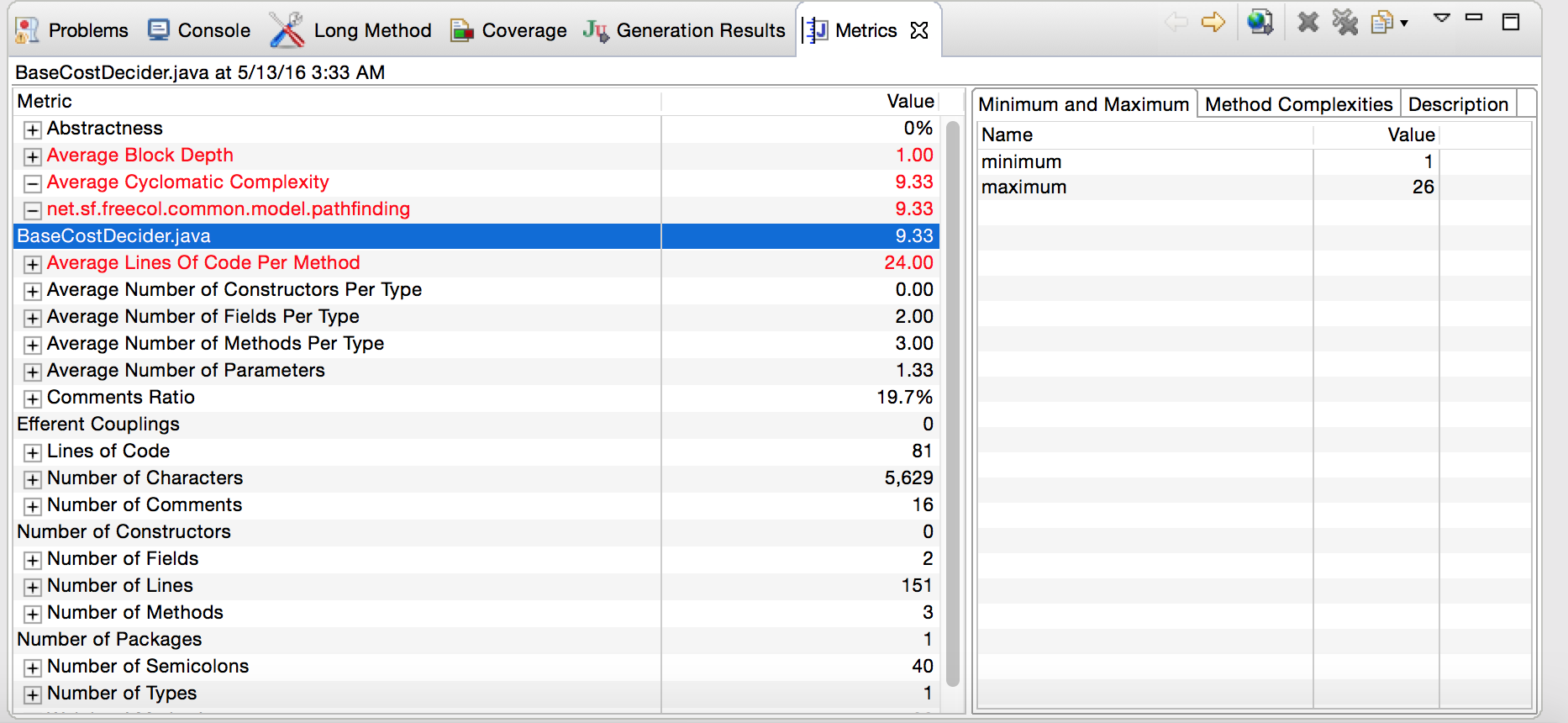
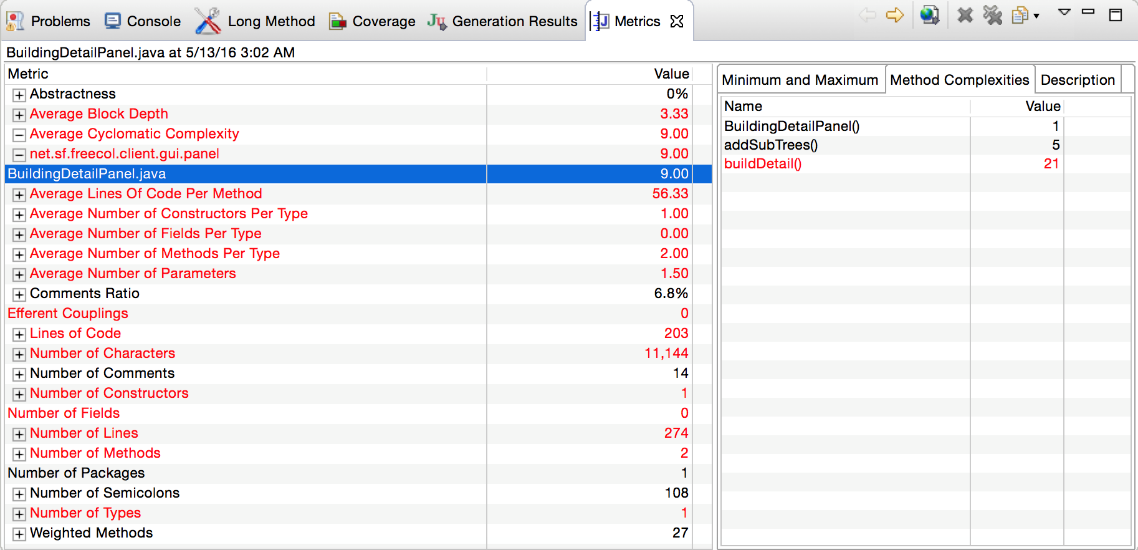
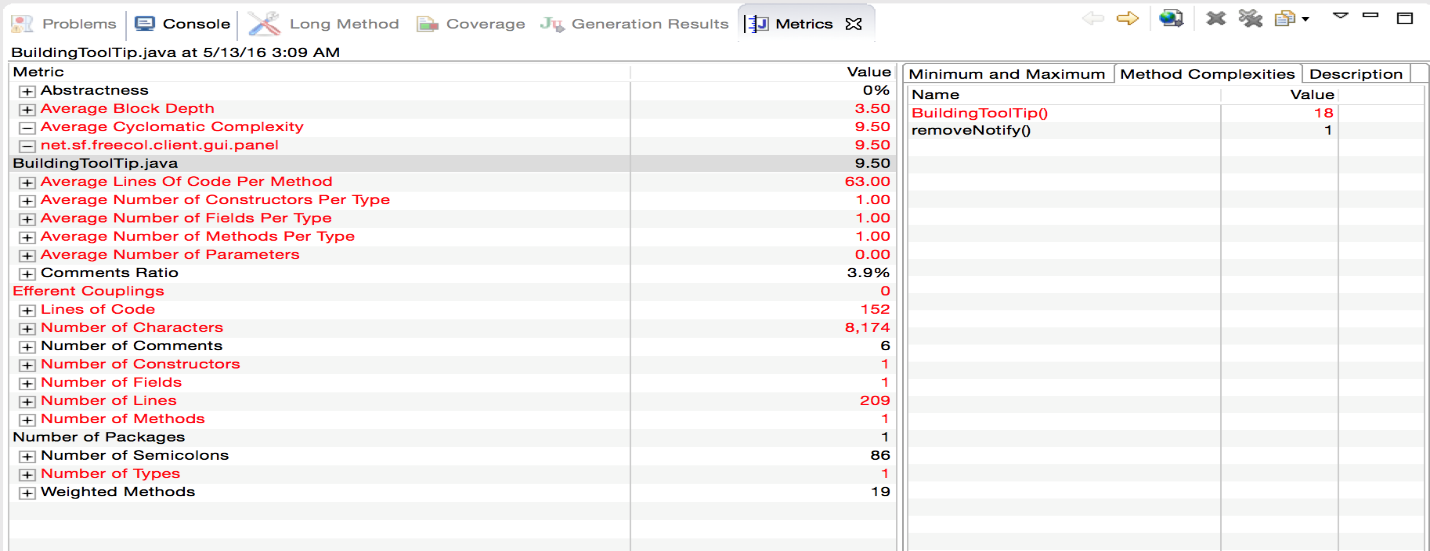
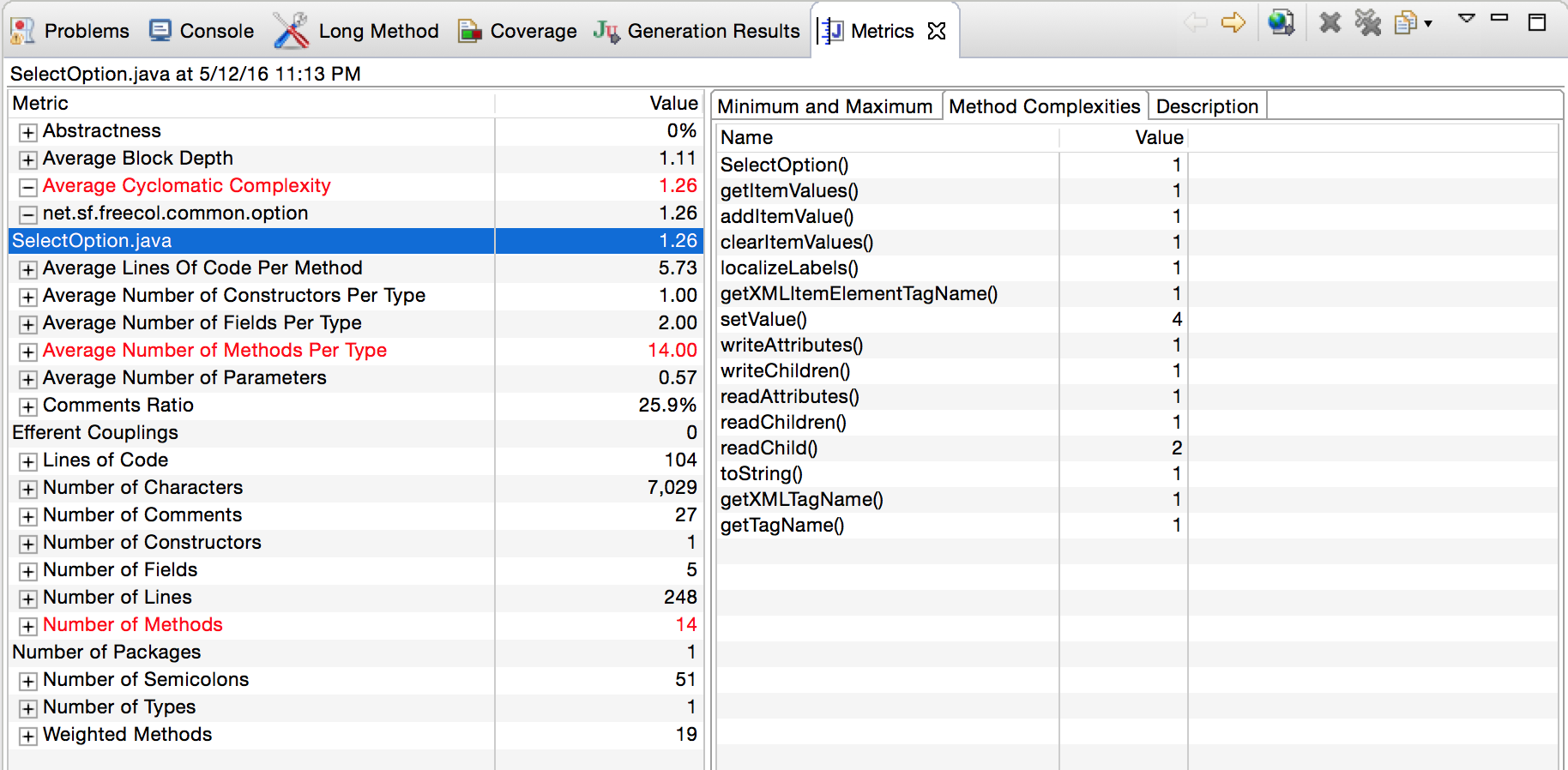
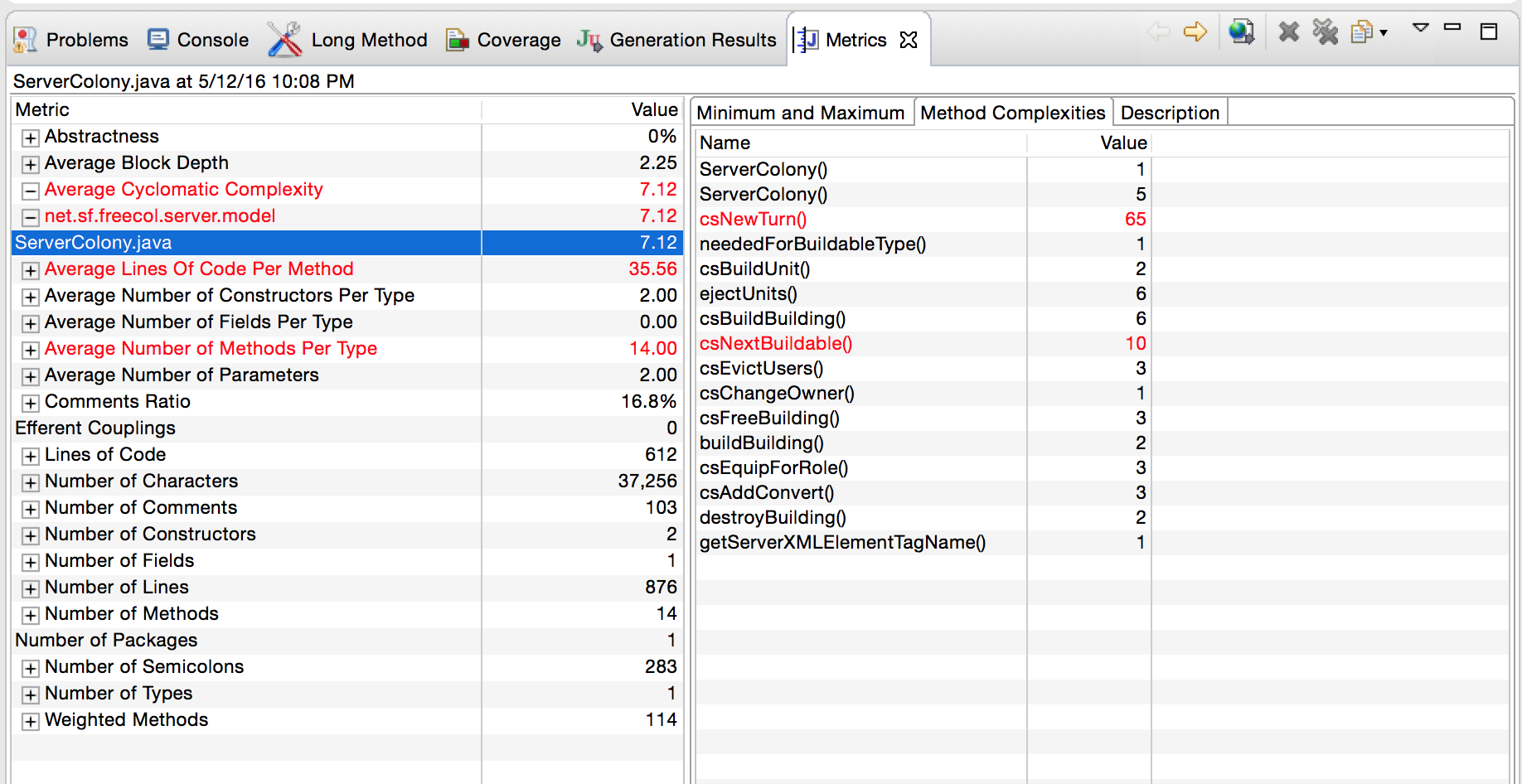
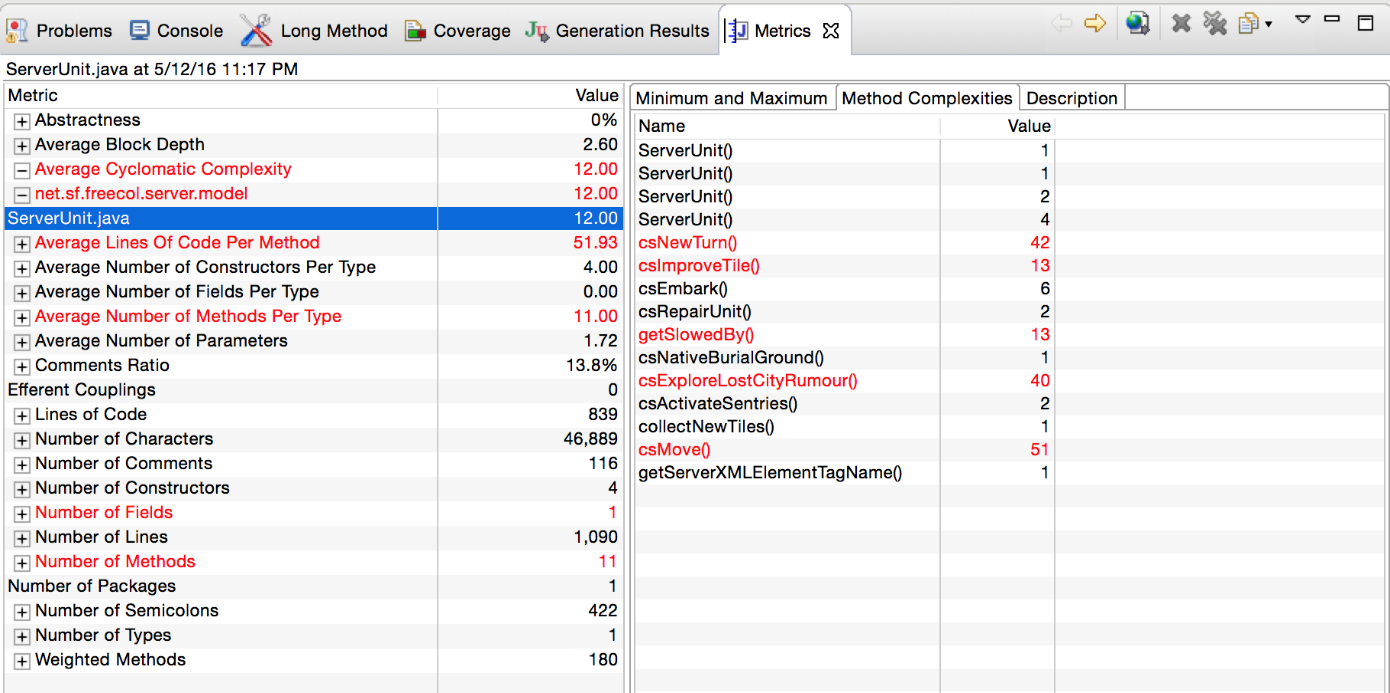


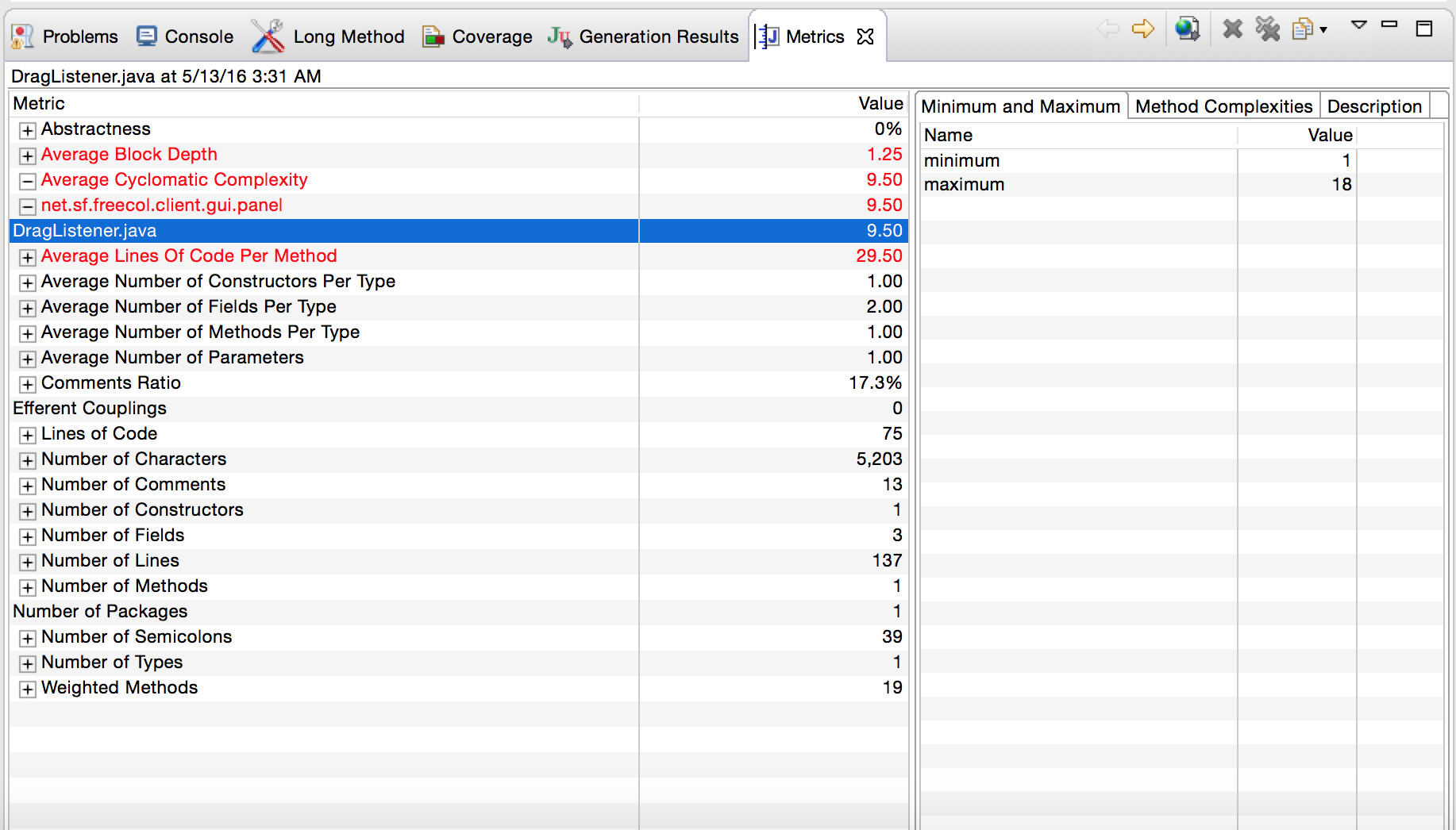
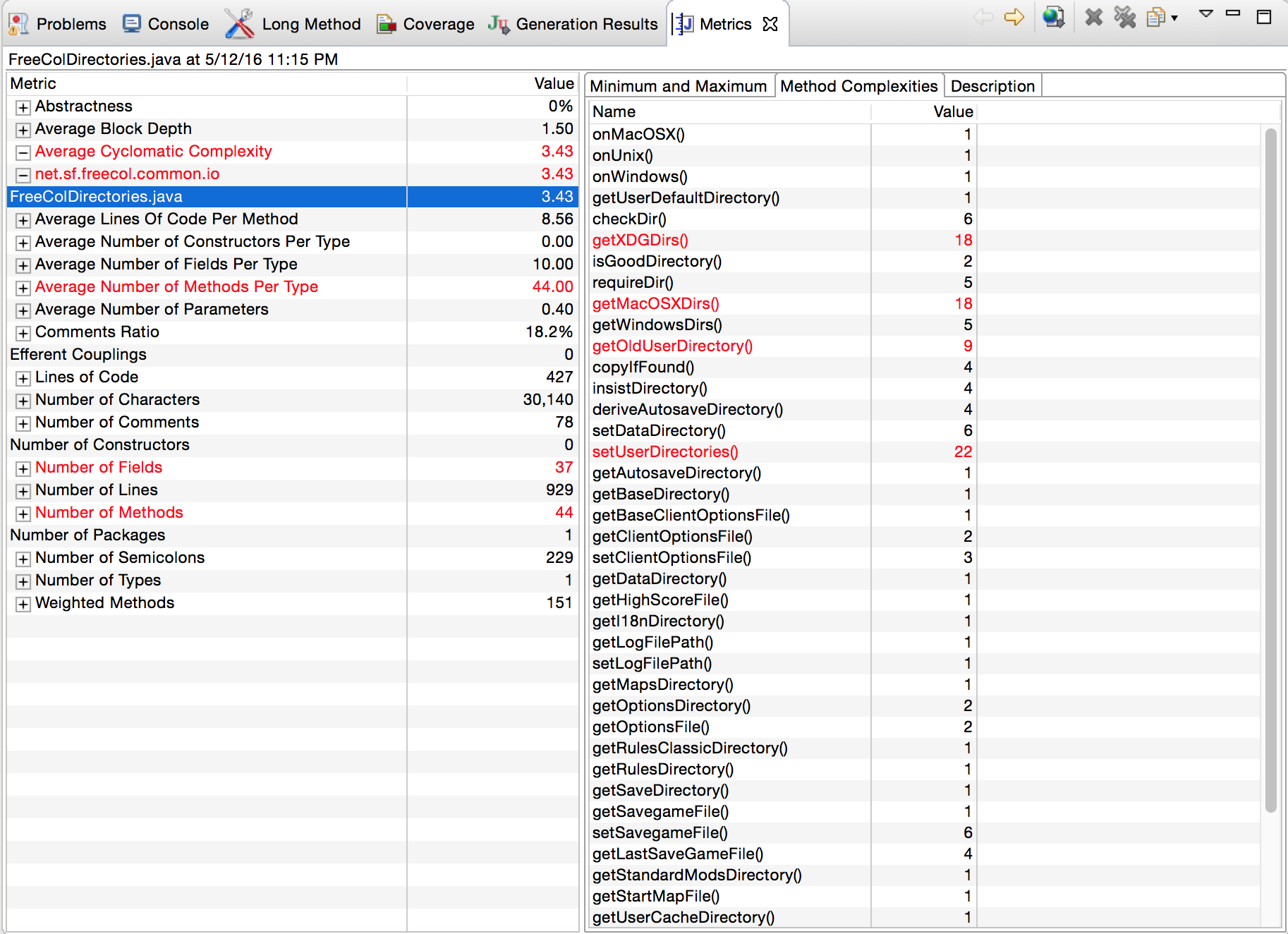
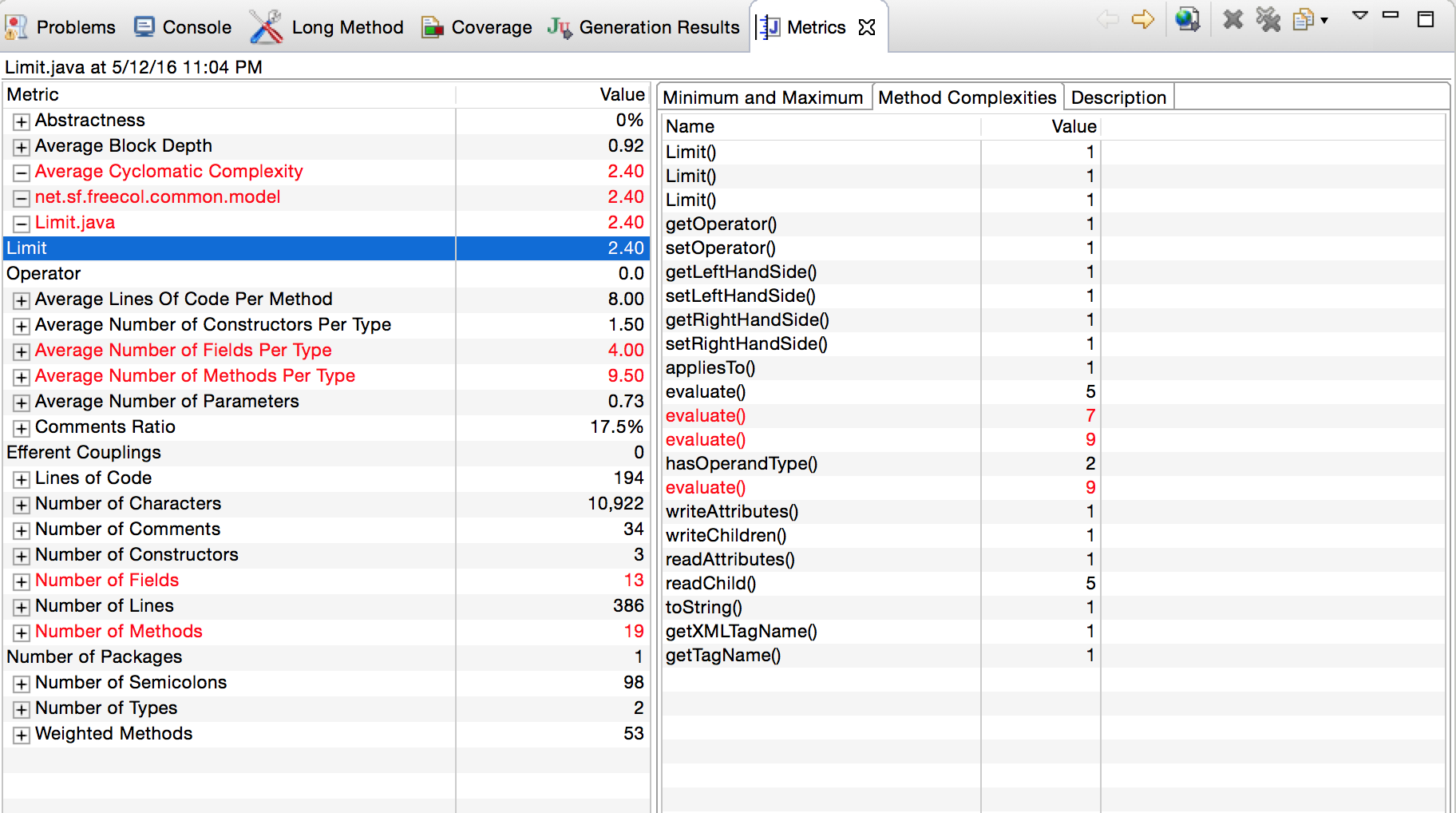
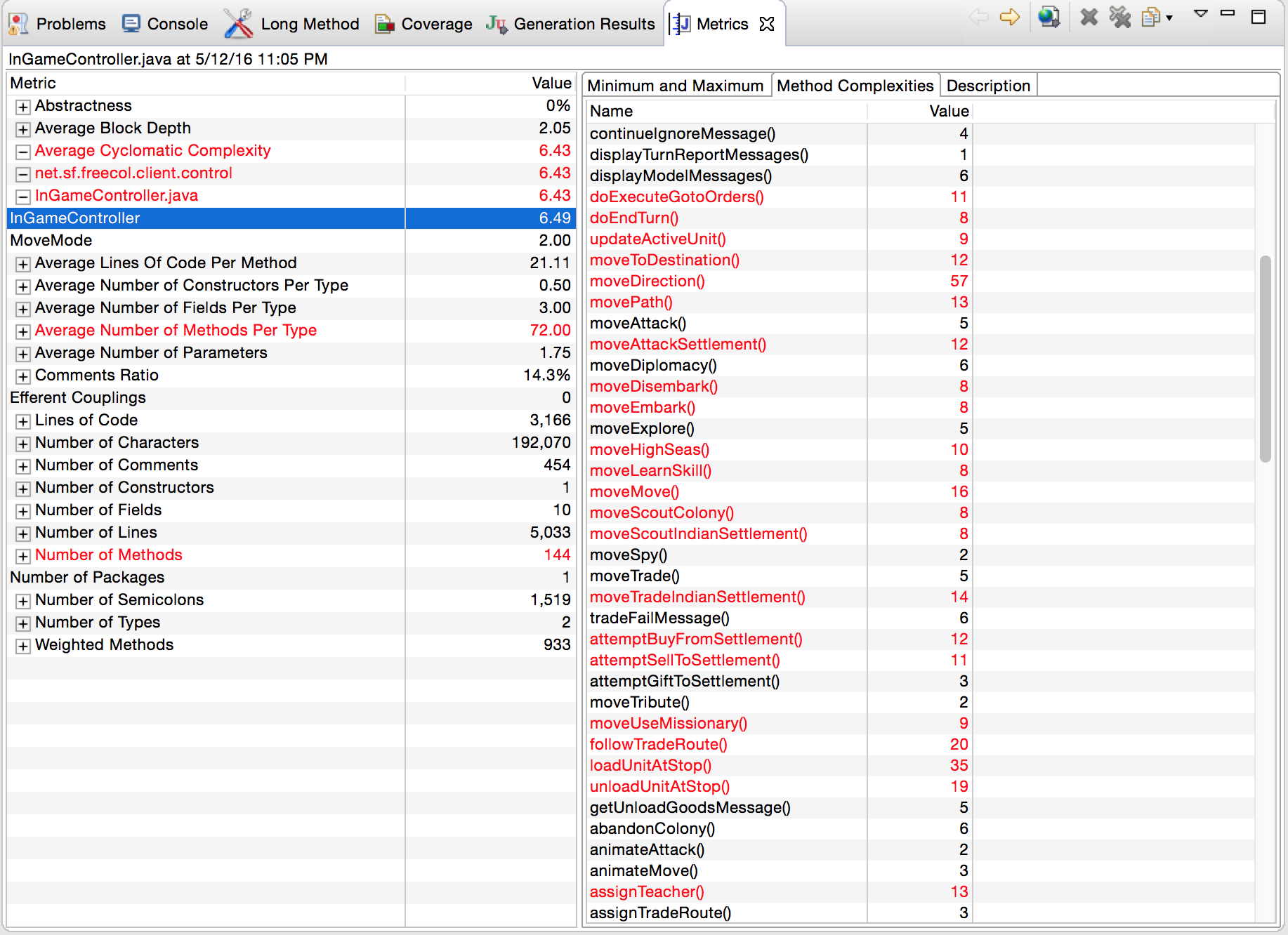
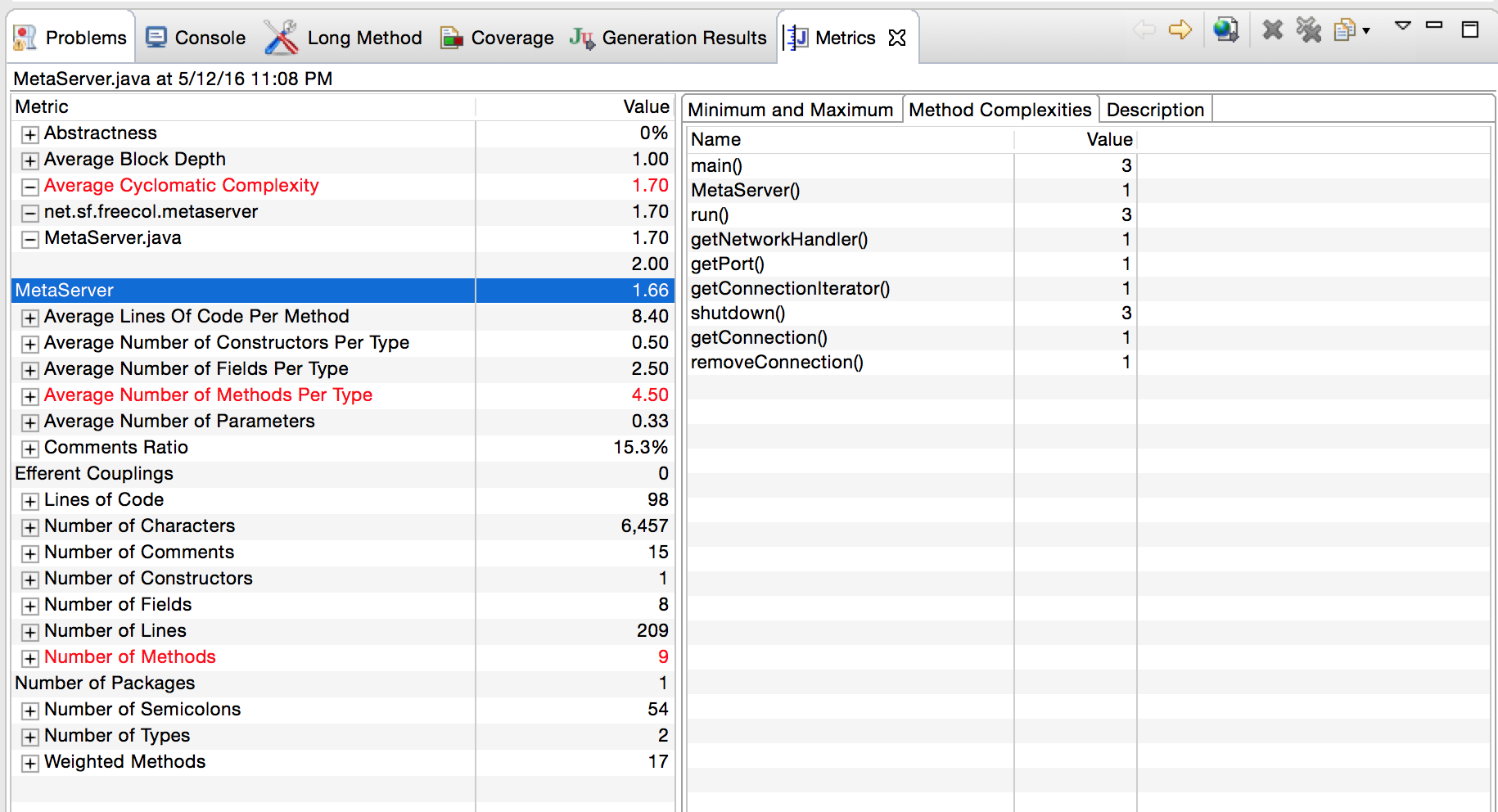
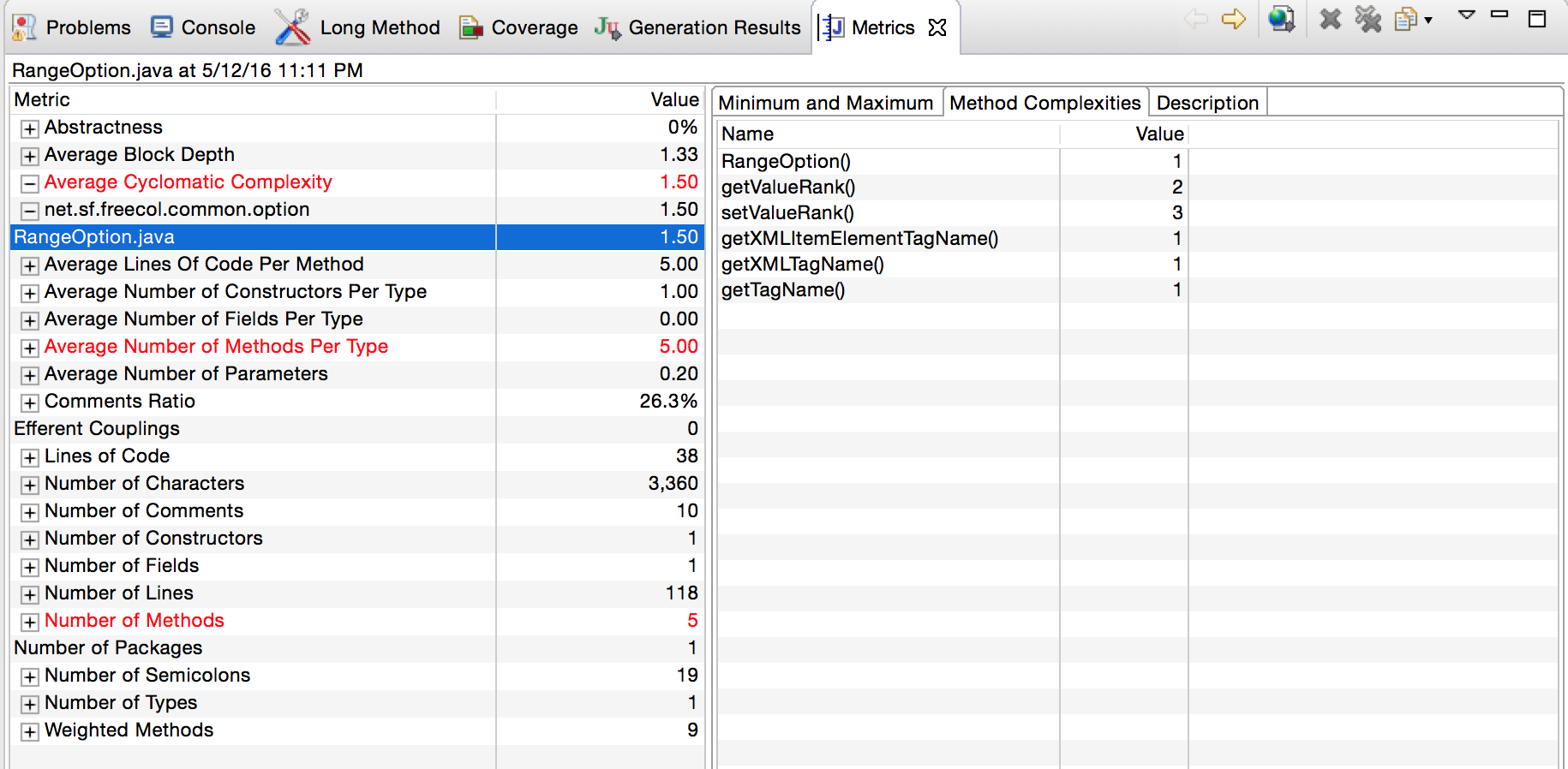
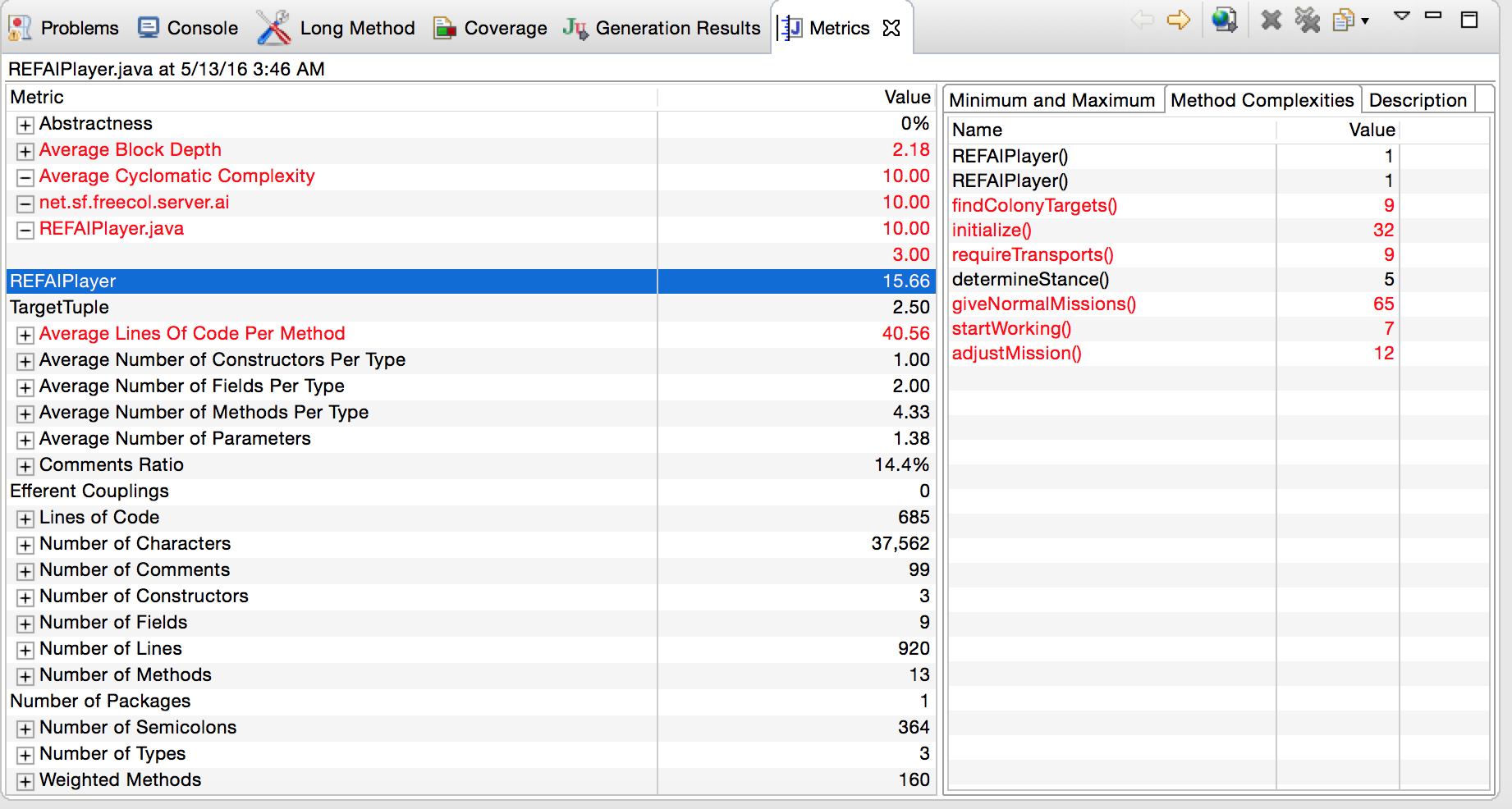
1. Net.sf.freecol.common.model/NationSummary.java: Achieved 49.1% code coverage.
2. Net.sf.freecol.common.model/Operand.java: Achieved 79.2% code coverage.
3. Net.sf.freecol.common.model/Limit.java: Achieved 85.1% code coverage.
4. Net.sf.freecol.common.model/IndianNationType.java: Achieved 85.9% code covereage.
5. Net.sf.freecol.common.model/Disaster.java: Achieved 87.7% code coverage.
6. Net.sf.freecol.common.model/NationOptions.java: Achieved 90.9% code coverage.
7. Net.sf.freecol.common.model/Nation.java: Achieved 93.3% code coverage.
8. Net.sf.freecol.common.model/NationType.java: Achieved 87.0% code coverage.
9. Net.sf.freecol.common.model/BuidableType.java: Achieved 92.9% code coverage.
10. Net.sf.freecol.common.model/AbstractUnit.java: Achieved 94.5% code coverage.
11. Net.sf.freecol.common.model/ResourceType.java: Achieved 87.47% code coverage.
12. Net.sf.freecol.common.model/AbstractGoods.java: Achieved 94.5% code coverage.
13. Net.sf.freecol.common.model/Stance.java: Achieved 98.2% code coverage.
14. Net.sf.freecol.common.model/Tension.java: Achieved 93.5% code coverage.



**Will:**

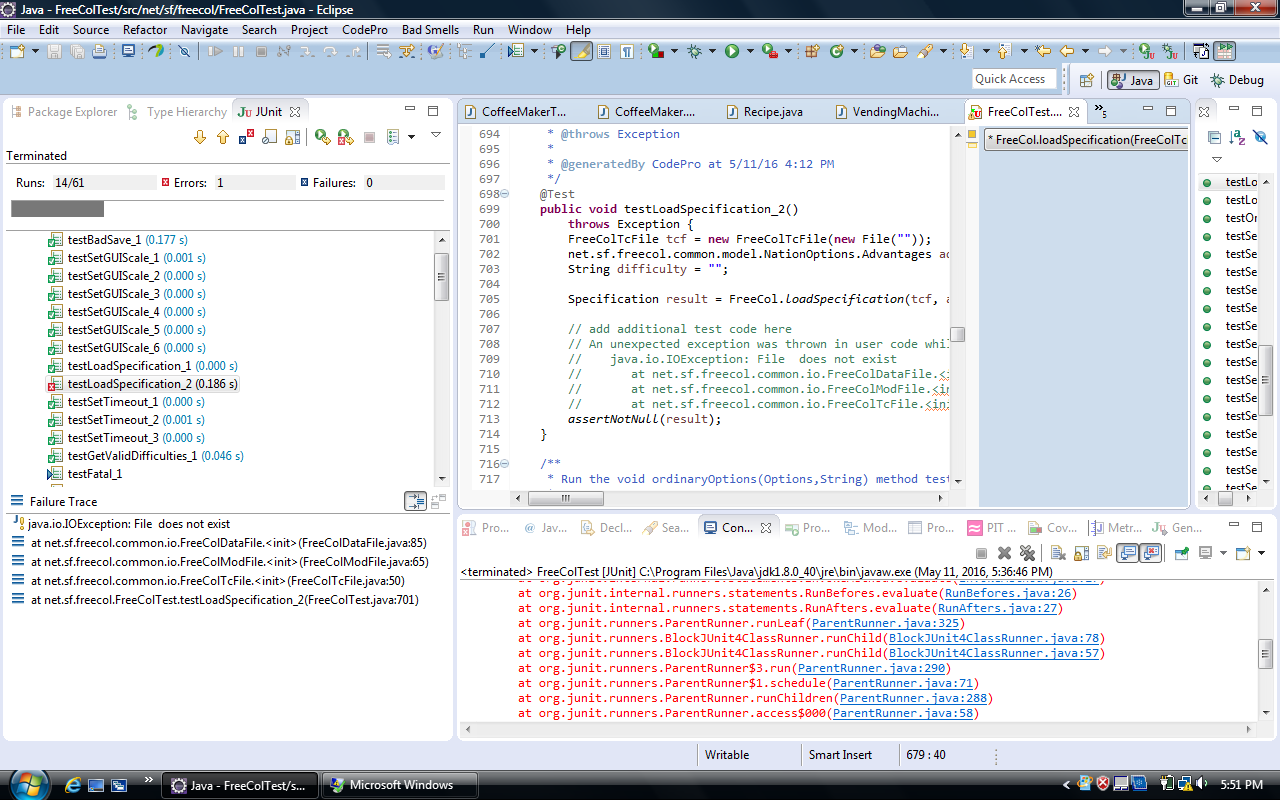
Will’s responsibility is to fix all the severe bugs that were found with the FindBug tool as well as all the high code violations found with the PMD tool for the classes picked. He is to also refactor method using JDeodorant, and Eclipse Refactoring tools where possible. He is to lower the cyclomatic complexity on the classes that it is possible for. He is also to generate all the JDocs documentation for all the classes.

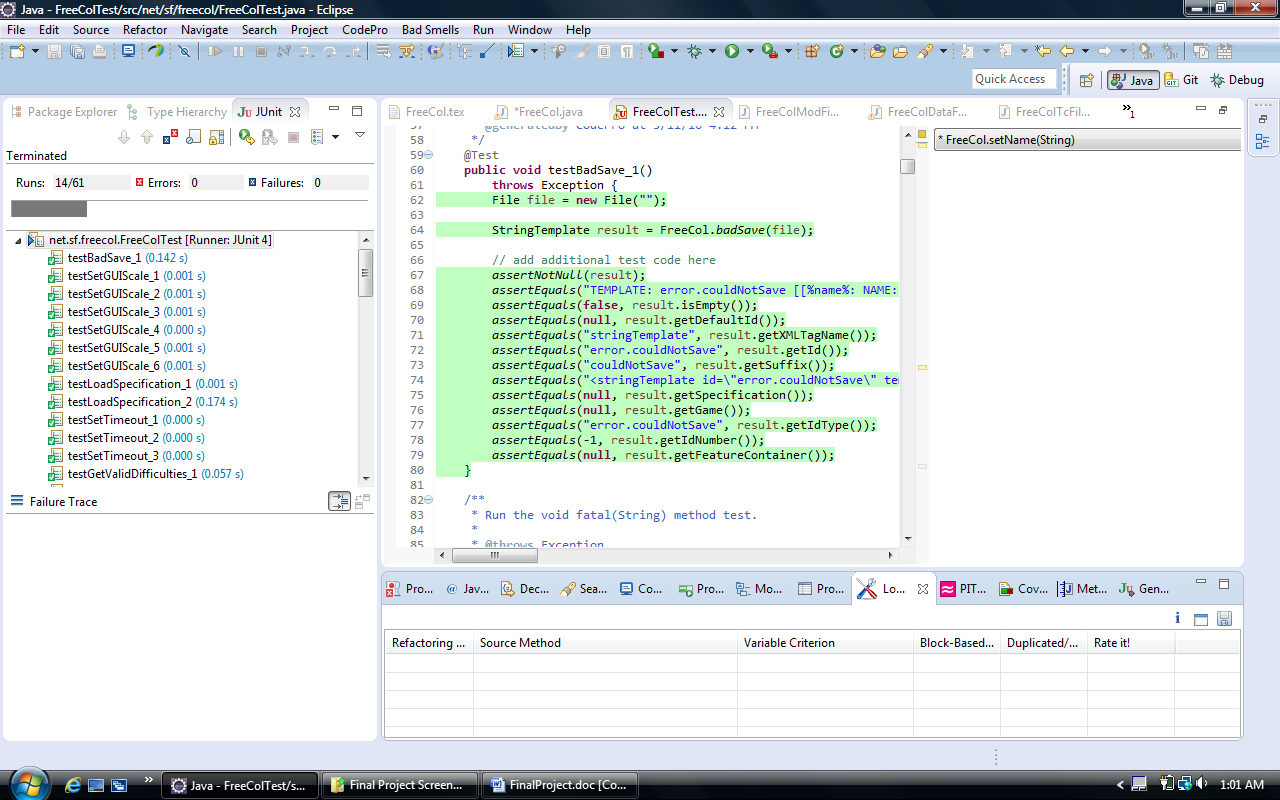
 



**Jose:**

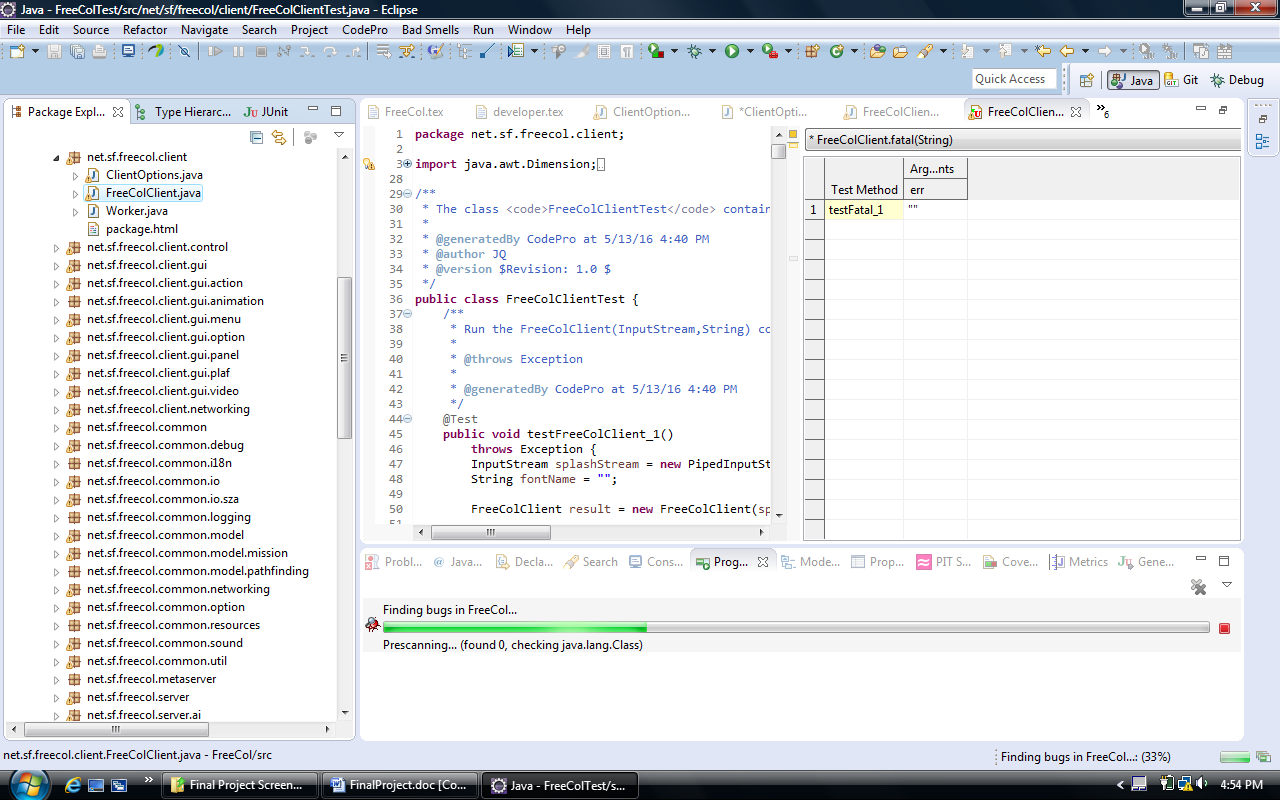
Jose is to assist with refactoring methods using JDeodorant, and Eclipse Refactoring tools, create JUnit test cases, fix style issues found with CheckStyle

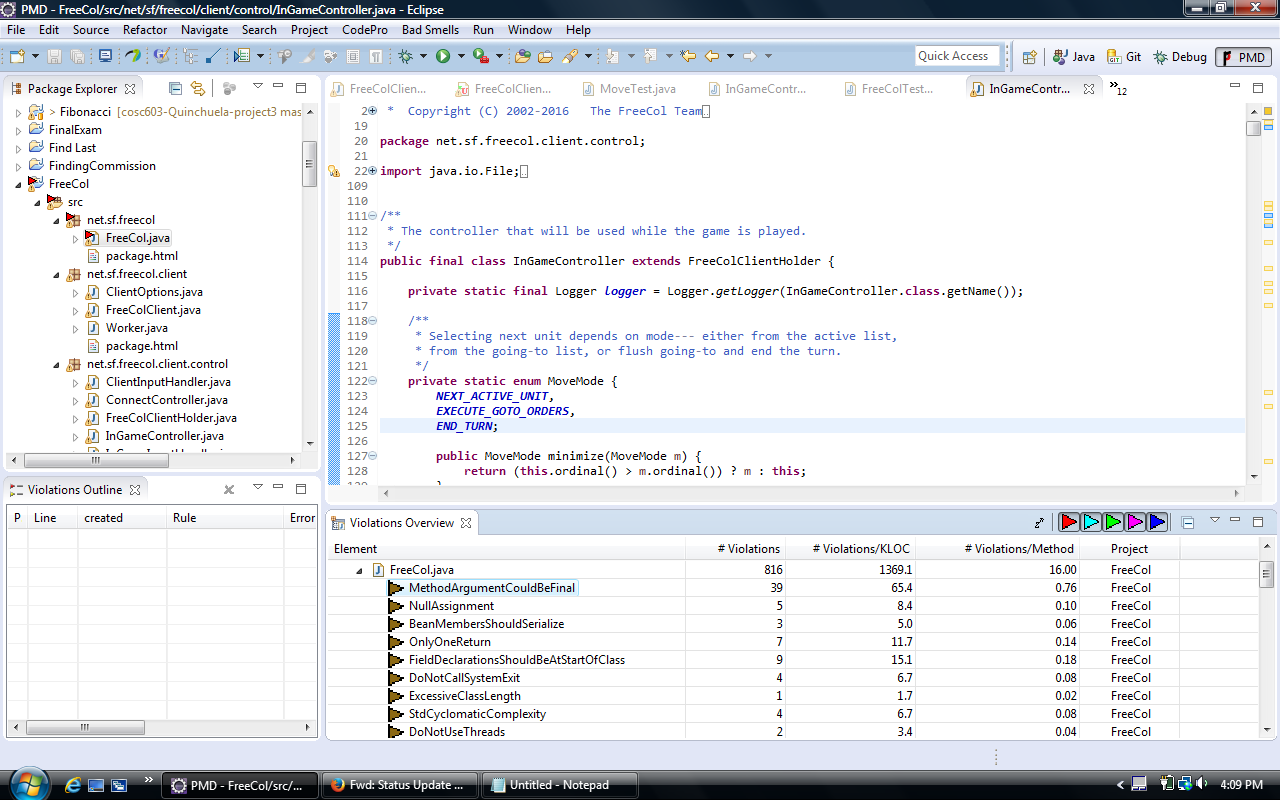




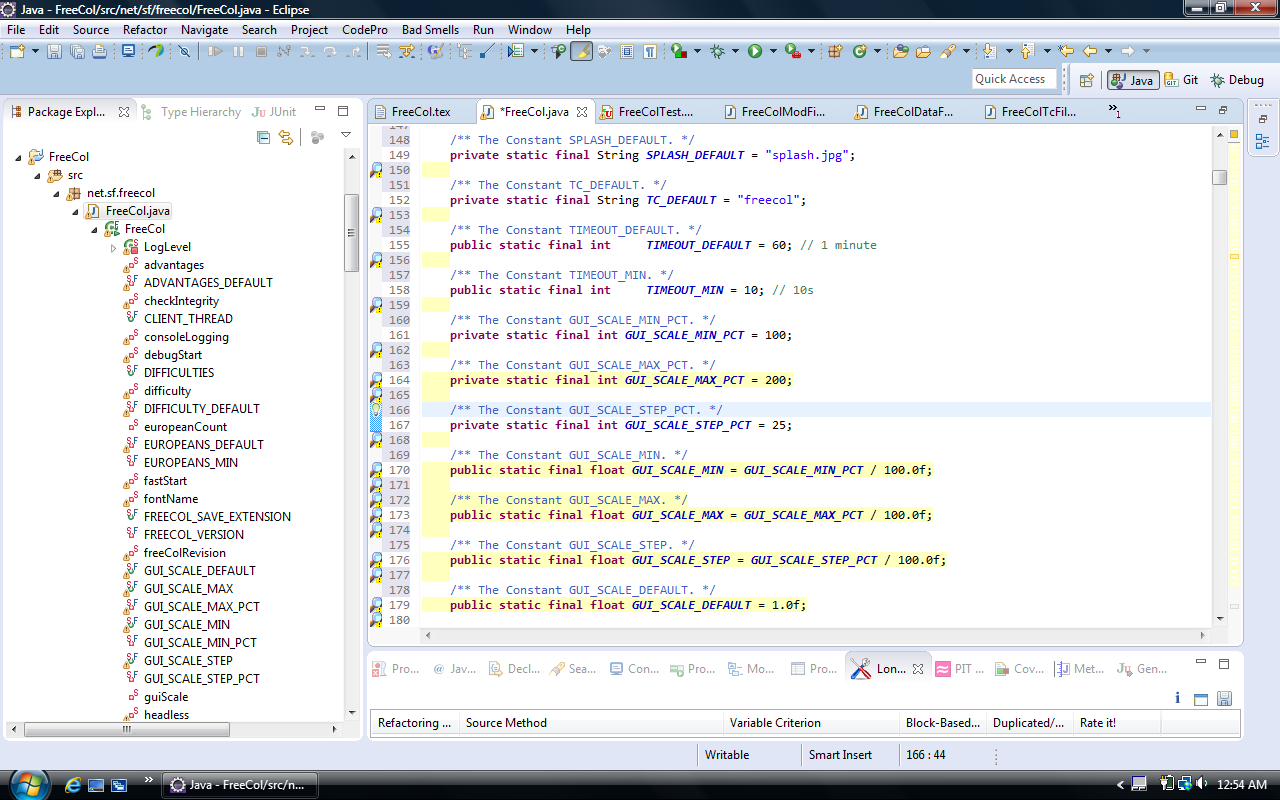
I had questions where it asked me to fix a null value but I left it there because it was part of a variable assignation.

No bugs were found with FindBugs

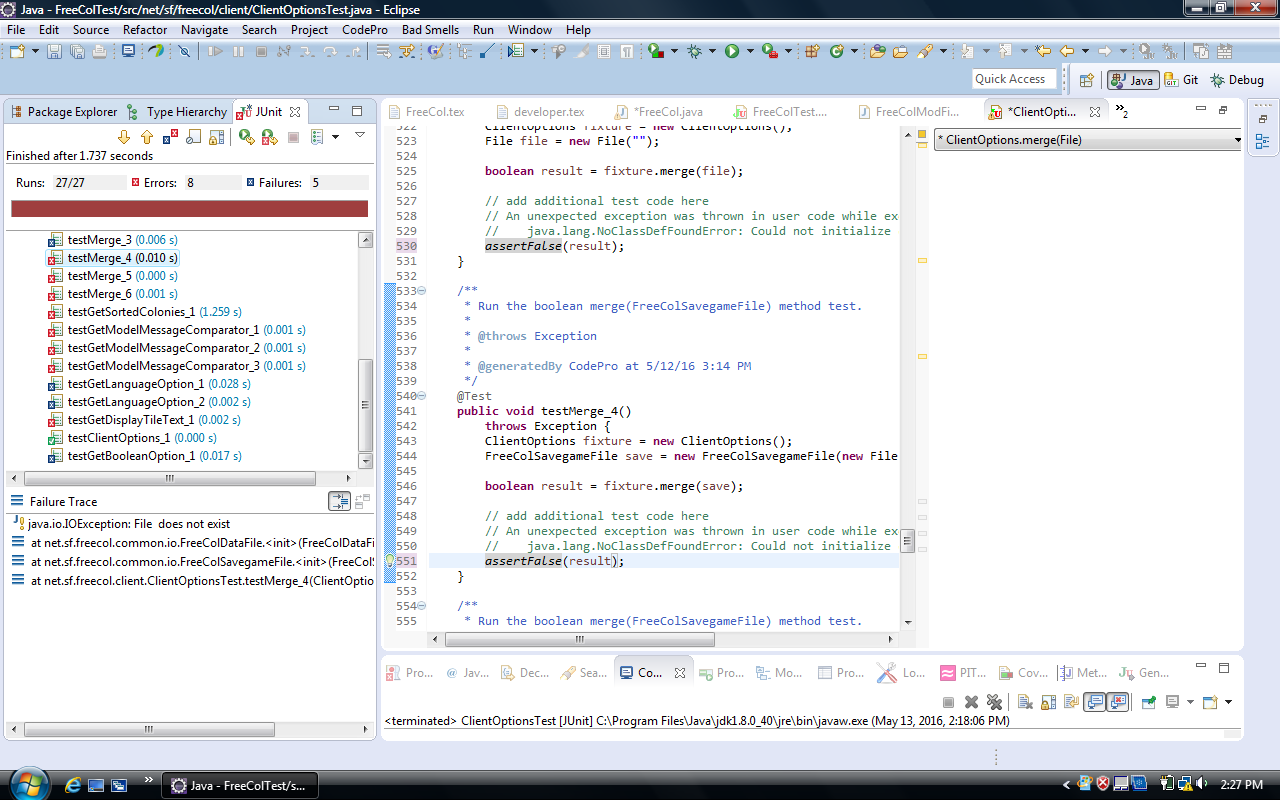


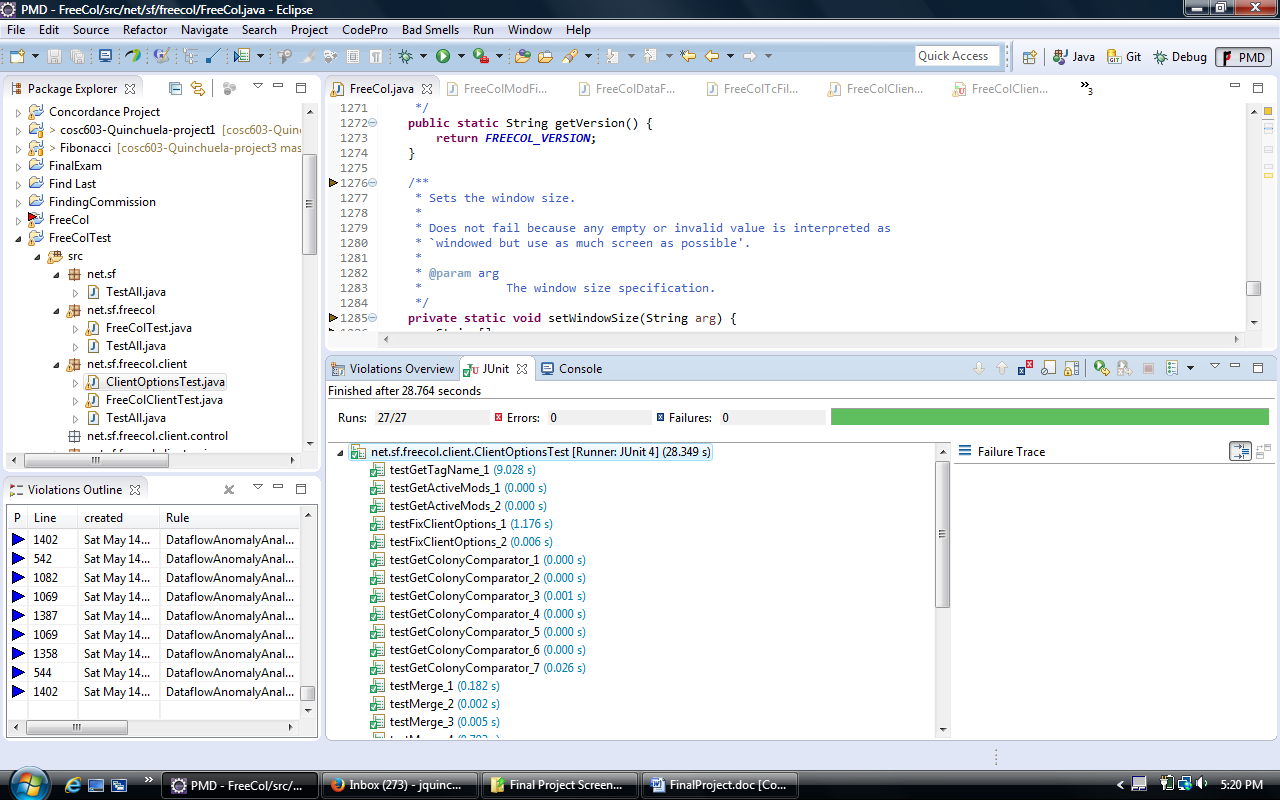


Checklist used. I used Eclipse Formatter tool to clean up most of the warnings.

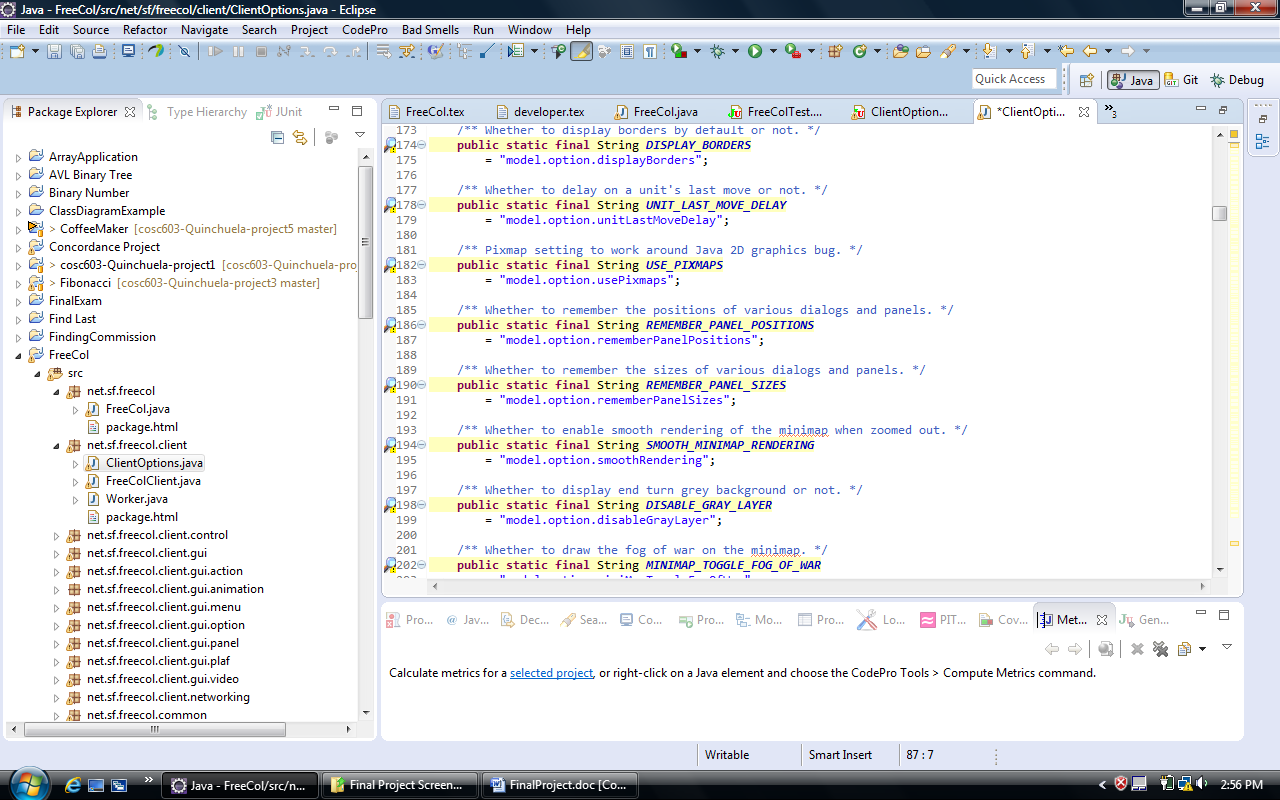


More test cases:





Checklist used. I used Eclipse Formatter tool to clean up most of the warnings.



**Identify the groups responsible for the environmental needs above.**

Everyone on the team would be responsible for their own environmental needs above cited.

**Software Project Schedule**



