|  |  |
| --- | --- |
|  | |
| Project Software Implementation and Improvement |
| SWEN 772, Complex Class Detector | |
| Silva Matti| Pavithra Sathiyanarayanan | Alexander Bogart | Avezou Petit Frere |

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Revision Number | Author | Modification | Date |
| 1.0 | Pavithra | 1, 2.0, 3.1, 3.2, 3.3 | 11/29/2016 |
|  | Silva | 3.1 | 11/29/2016 |
| 1.1 | Alex | 1.3, 3.4 | 12/02/2016 |
|  | Silva | 3.2, 3.3, 2.1 | 12/02/2016 |
|  | Pavithra | 2.1 | 12/02/2016 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[Revision History](#_gjdgxs)

[1. Deployment Requirements](#_30j0zll)

[1.1 Hardware Requirements](#_1fob9te)

[1.2 Software Requirements](#_3znysh7)

[1.3 Setup and Deployment Instructions](#_2et92p0)

[2. Requirements Traceability Matrix](#_tyjcwt)

[2.1 Functional Tests](#_3dy6vkm)

[2.1.1 Test X](#_1t3h5sf)

[3. Challenges and Future Work](#_4d34og8)

[3.1 Lessons Learned](#_2s8eyo1)

[3.2 Open Issues](#_17dp8vu)

[3.3 Recommendations](#_3rdcrjn)

# 1. Deployment Requirements

## Hardware Requirements

Any functioning computer

## Software Requirements

**OS -** Windows/ Linux

**IDE -** Eclipse Neon (Open source)

**Plug-in -** Metrics3 (Open source)

**Version Control -** Git (Open source)

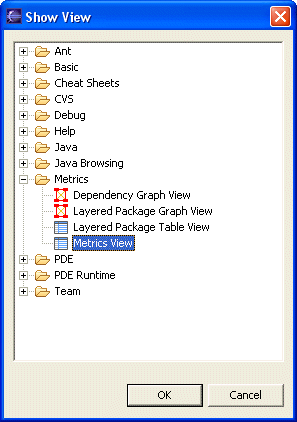
**Technology -** Java

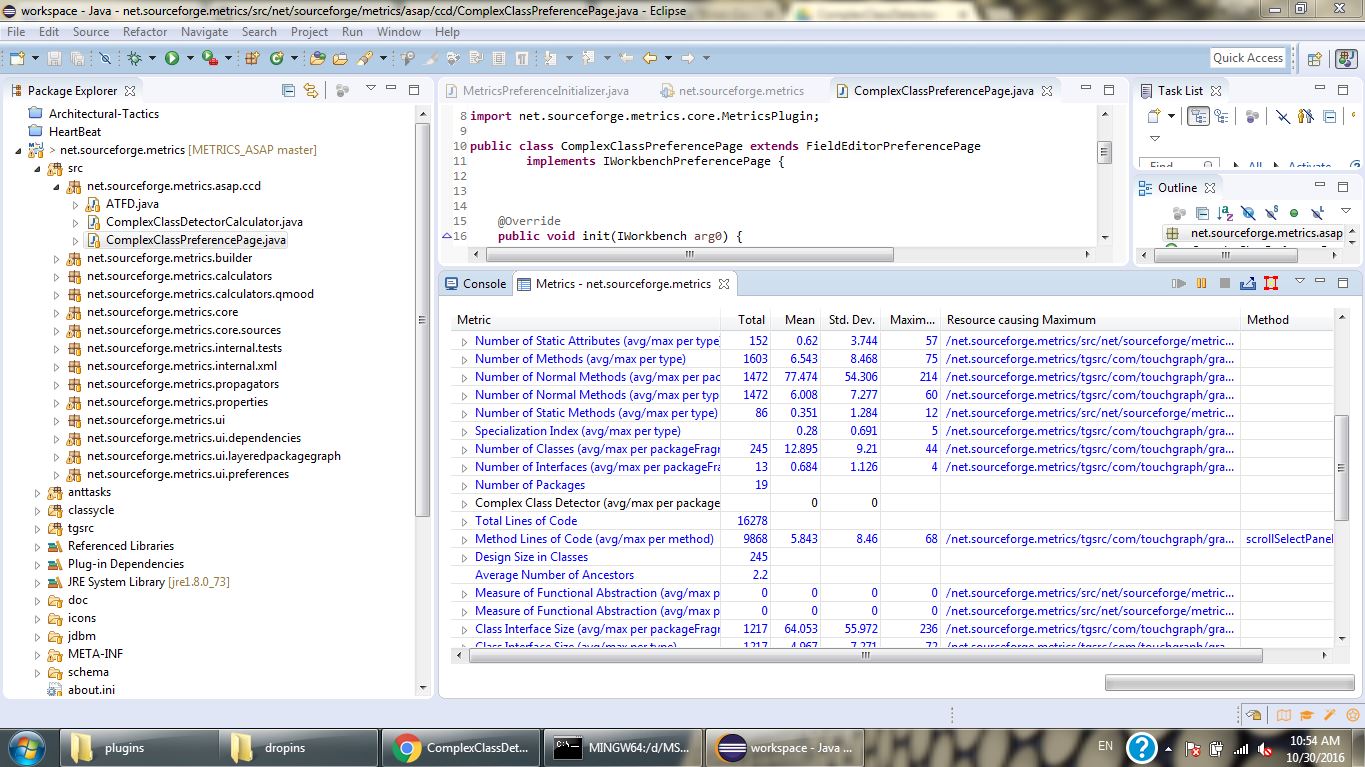
## Setup and Deployment Instructions

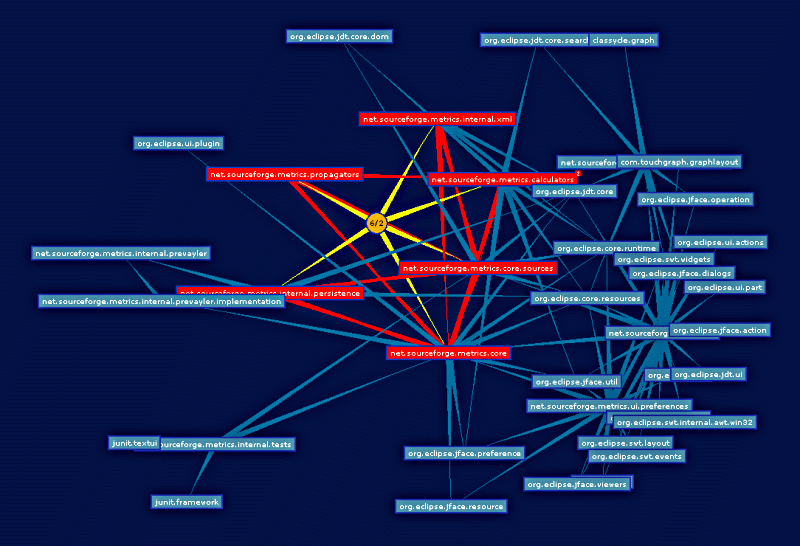
There were difficulties exporting a consistently functional .jar file. The following setup instructions detail how to run the source code itself as a plugin with Eclipse.

Video of setup: <https://youtu.be/rxcX3OI9tQ0>

Video of use: <https://youtu.be/ti18OABi80c>

1. Install Eclipse <https://eclipse.org/downloads/>
2. Open Eclipse.
3. Import the swen772\_mmetrics3 folder using File 🡪 Import 🡪 General 🡪 File System
4. If the project has build errors, you likely need to install the “Eclipse Plug-in Development Environment” and restart Eclipse.
5. Now, open swen772\_mmetrics3 🡪 net.sourceforge.metrics 🡪 plugin.xml.
   1. You should see a tab called “Overview.” Select it if it isn’t already.
   2. Under the “Testing” header, select “Launch an Eclipse Application”
6. Now, you should be in a new Eclipse window. Import the projects you would like to analyze here. They will remain even if you restart this instance of Eclipse.
7. Select the project for which Metrics have to be calculated; right-click it and choose Properties
8. In the Properties window, select Metrics from the left pane
9. Check the Enable Metrics checkbox and click OK
10. From the Menu Bar, Click Window → Show View → Other → Metrics → Metrics View and click OK
11. From the Menu Bar, Click Project → Clean
    1. Metric values will be displayed, but these may not be accurate depending if the project has been changed, another project has been cleaned in the interim, etc
12. Clean the project a second time.
    1. Now the values are accurate
13. The CCD table will display the “complexity rating” for a given class. A higher complexity rating indicates a greater need for refactoring.
    1. A developer may enable a flag in CCDCalculator.java to limit this value to 3, with 1 being low, 2 being medium, and 3 being high.
14. The CCD1 table will indicate the number of metric “rules” that have been broken by each class. The sum of these rules’ weighted complexity values is what is displayed in the CCD table.
15. In order to see graphic view, choose graphic view from the Show View options
16. icons





# 2. Requirements Traceability Matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Functional Requirement** | **System Component(s)** | **In Design?** | **Implemented?** |
| The system shall utilize a given set of metrics to identify complex classes in a given code base. | Complex Class Detector | Yes | Yes |
| The system shall take top 25 or bottom 25 percentile of the classes to identify the most complex classes relative to the other classes | Complex Class Detector | Yes | Yes |
| Add CCD as one of the metrics to the metrics list | Metrics Plugin | Yes | Yes |
| Group various Metrics in pairs to identify the complex class (Perform all combinations) | Complex Class Detector | Yes | Yes |

## Functional Tests

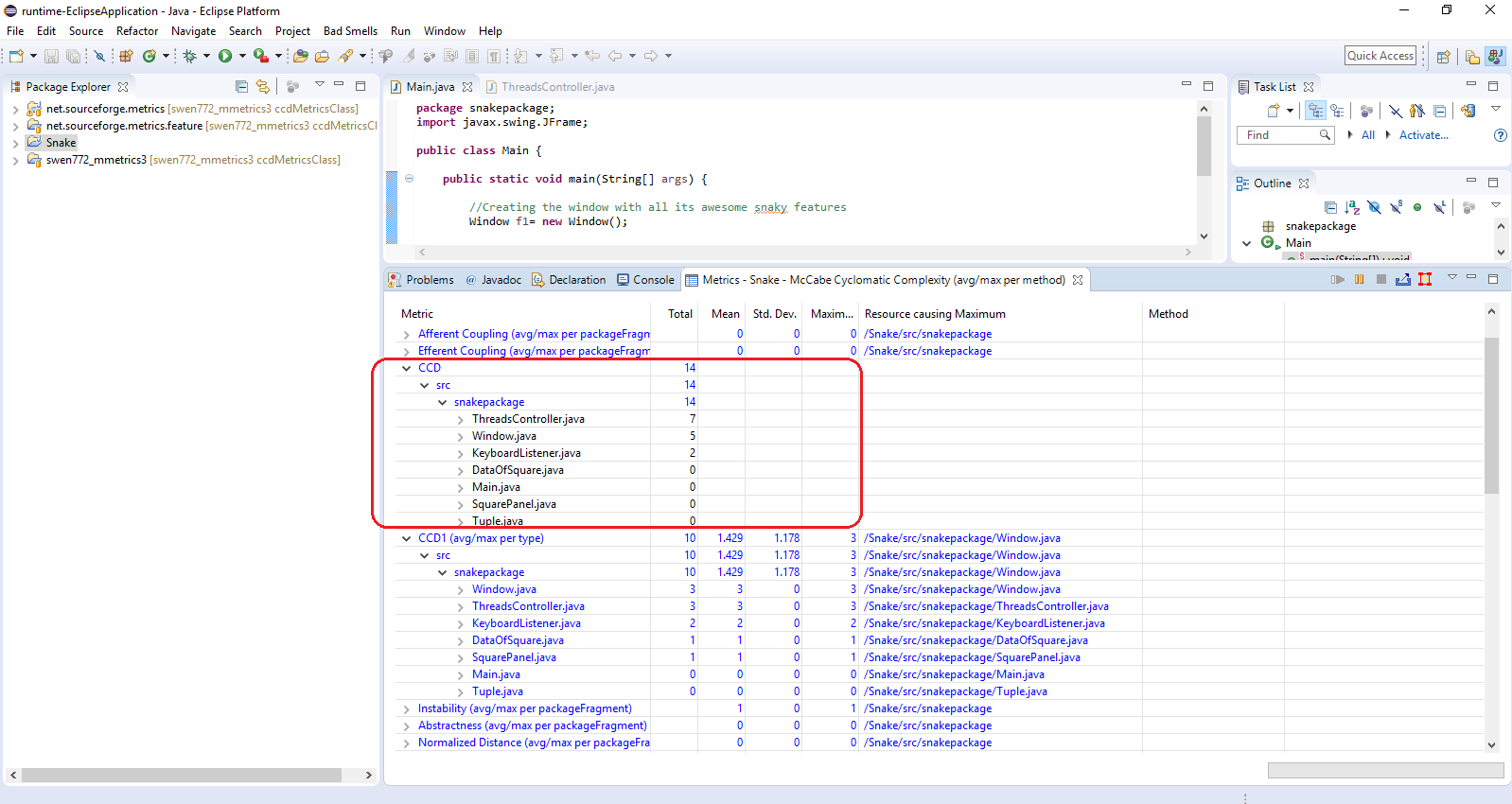
These are black-box tests to show how the software fulfills the needed operations. These operations can be triggered by one or multiple scenarios that will be detailed in the subsections below:

### Test 1

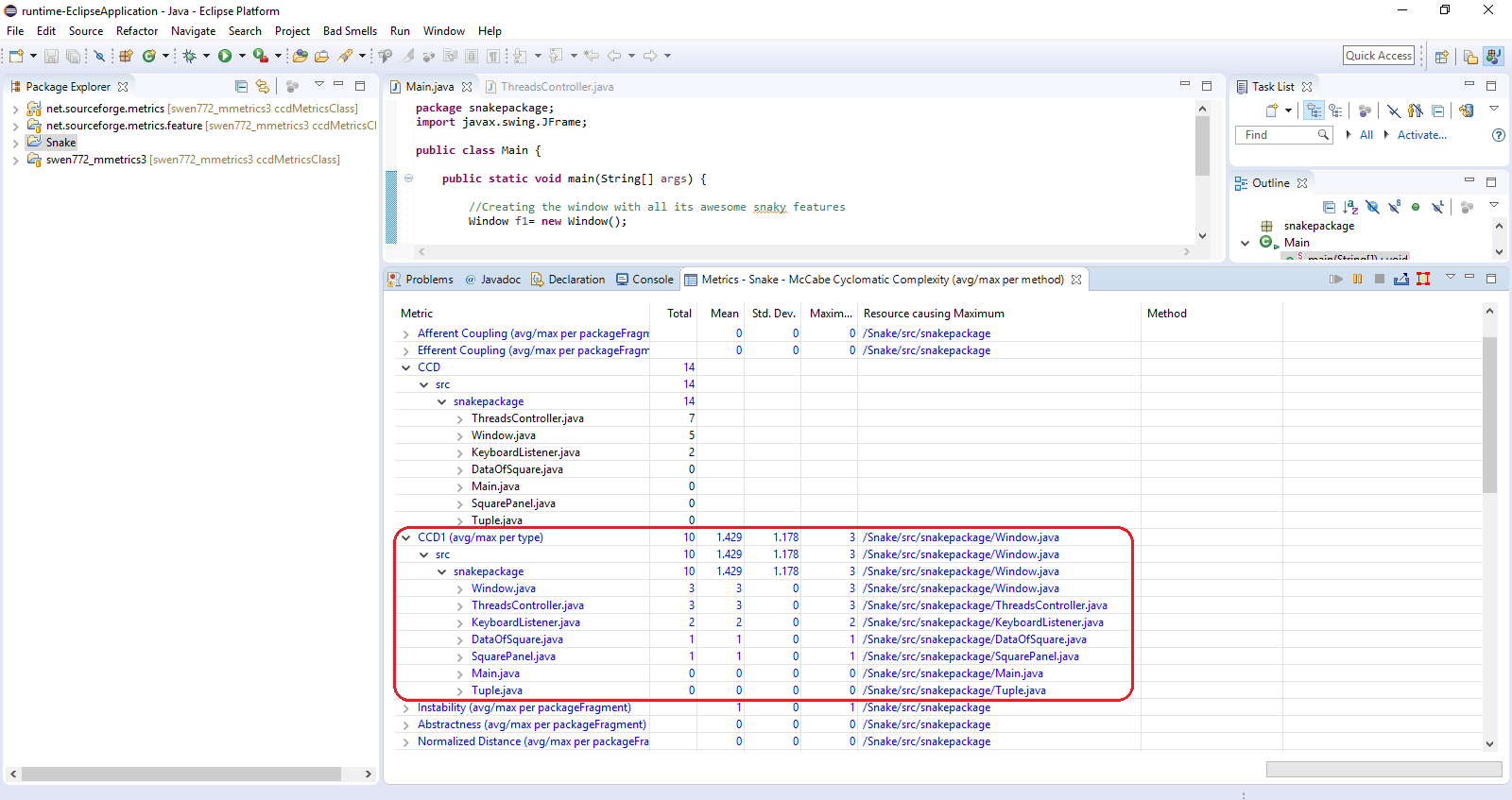
* + - 1. *Input*

The source code of the project for which complex classes needs to be identified.

* + - 1. *Output*
* Class list along with the number of violations
* Severity of the complex class (level - 1, 2, 3)
  + - 1. *Screenshot*



* + 1. **Test 2**
       1. *Input*The source code of the project for which complex classes needs to be identified.
       2. *Output*
* Class list along with the number of violations
* Number of violated metric rules (No more than 5, as there are 5 metrics)
  + - 1. *Screenshot*



# 3. Challenges and Future Work

## Lessons Learned

1. The main challenge was building upon an existing plugin that had numerous constraints while implementing new features. We still prefered building upon this plugin rather than starting from scratch due to time constraints and the fact that the plugin was already providing the metrics needed to calculate CCD.
2. The main difficulty was with displaying the results of CCD in the Metrics View that already exists and has a specific layout.
3. Improper documentation regarding the flow or structure of metrics3 impeded development. We mitigated this by manually going through the source code and understanding the flow of the plug-in with the help of break points.
4. The initial definition for identifying a God Class comprised of ATFD, WMC and TCC metrics. But metrics3 didn’t provide the value for ATFD. Since there is no concrete definition for identifying complex classes, we defined a high level definition which comprises of six metrics to identify the complex class, and hey are WMC, TCC (or CAM), DCC (Afferent and efferent coupling), LCOM, DIT and NOM.
5. CCD Metric is different from other metrics in such a way that other metrics returns a value where as CCD returns set of classes. Dealing with the retrieval of classes was not straightforward and we leveraged the Map and List data structure to fetch every class name along with its metrics and its values.

## Open Issues

1. In order to identify if a class is a complex class or not, we planned on grouping the metrics by 2, by 3, etc. But in this demo, we were not able to perform group by 3 as the computational time is really high and considering a three dimensional approach for grouping is left for future task.
2. Showing the CCD graphically is also an open issue as it is complicated to examine and comprehend the dependencies within the limited time.
3. When performing one clean (is needed to calculate all the metrics in Metrics3) CCD will not give accurate values as Metrics3 calculates class metrics before all classes added to collection. This was solved with static collection of class metrics retained after clean. This led to the need for performing clean twice to get the actual CCD results.While this is a good solution, finding a way to get results with the first clean would be something to consider later.

## Recommendations

1. Adding more relevant metrics to uncover the complex classes.
2. Adding more interactive view than displaying the list of classes.
3. Adding severity of complex classes (Color coding).
4. Group metrics by 3 or more to identify Complex Classes.

# 3.4 Future Scenarios

***Choose a refactoring technique for a detected complex class***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

***The CCD has identified at least one complex class in a code base***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

|  |
| --- |
| **Flow of Events** |
| 1. The user selects a complex class and selects its prompt to display suggestions for refactoring 2. The system displays to the user the list of available refactoring techniques,    1. The techniques are ranked by applicability based on the metrics for that particular class    2. The system displays a contextualized explanation for why the technique is applicable 3. (optional) The user selects the prompt to display the graphical view of the metrics for the selected class |

## 