CS/SE 6356 Software Maintenance, Evolution & Re-engineering
Spring 2023
Assignment 3: Coupling and Cohesion
Team: 05
Members:
Aditya Khandare - ark200000
Deep Gosalia - dxg190036

Code quality software used - *CodeMR*

Metrics used and their respective descriptions:

1. **Cohesion:**
    a. **Lack of Tight Class Cohesion (LTCC)** -
    This method aims to measure the direct and indirect relation between (only) the PUBLIC methods of the given class and how well they are connected. The results resonate well with the generic definition of having high cohesion. That is, classes with a lower LTCC are preferred.
    LTCC = 1 - TCC
    TCC for a class is the ratio of the number of public methods of that class using a set of attributes belonging to that class (let's name the variable A), to
    (A x (A - 1))/2.

    b. **Lack of Cohesion of Methods (LCOM3)** - This is a structural metric that uses the methods of a given class and assesses all the variables, attributes, used/unused and includes all the methods of the class, private public, etc.
    LCOM3 can be calculated using the following formula:
    LCOM3 = (m - sum(mA)/a) / (m-1), where
    m represents the number of procedures (methods) in the class,
    a represents the number of variables (attributes), including shared variables, and
    mA represents the number of methods that access a given variable.
2. **Coupling:**
    a. Coupling between objects (CBO) -
    This is a count of the number of classes that are coupled to a particular class i.e. where the methods of one class call the methods or access the variables of the other. These calls need to be counted in both directions so the CBO of class A is the size of the set of classes that class A references and those classes that reference class A. Since this is a set - each class is counted only once even if the reference operates in both directions i.e. if A references B and B references A, B is only counted once.
    CBO can be calculated as:
    CBO(c) = $|\{d \in C - \{c\}| \text{ uses}(c,d) \vee \text{uses}(d,c)\}|$
    CBO`(c) = $|\{d \in C - (\{c\} \cup \text{Ancestors}(C))|\text{uses}(c,d) \vee \text{uses}(d,c)\}|$

    b. Response for class (RFC) -
    This is the total number of methods that can potentially be executed in response to a message received by an object of a class. This number is the sum of the methods of the class, and all distinct methods are invoked directly within the class methods. Additionally, inherited methods are counted, but overridden methods are not, because only one method of a particular signature will always be available to an object of a given class.
    RFC can be calculated as
    RFC = |RS|

Where

RS = $\{M\} U_{all\ i} \{R_i\}$

$\mathbf{R_i}$ = set of methods called by method i

{M} = set of all methods in the class

<u>Explanation and Analysis:</u>

Cohesion

- **Lack of Tight Class Cohesion (LTCC):**
  Low Cohesion classes:
  - ➢ **TextArea/TextAreaTransferHandler** - 1
  - ➢ **TextArea/CompleteWord** - 1

  Why are these Low cohesion classes? -
  1. On inspecting **Class TextAreaTransferHandler** the methods defined inside this class all pertain to mapping, dragging, and moving the text area. The module in focus is highly concise despite the size of the class. While there are 5 Attributes used in the scope of the class, the Class TextAreaHandler has Low cohesion because the existence of public functions is very less. And the public functions present are less correlated but linked into smaller subtasks and attributes pertaining to them, due to which they share only a handful of common attributes and rather rely on their individual local attributes in the function call. Hence the exhibited cohesion types are **logical and communicational cohesion**.
     <u>Unexhibited</u> Cohesion types -
     - Temporal Cohesion - There is no evidence of time-based function application here. Hence we rule out the possibility of temporal cohesion.
     - Coincidental Cohesion - Since the set of functions is highly correlated, we rule out the possibility of coincidental cohesion.
     - Functional Cohesion - parts of the functions defined in the class are functionally related to each other. But the class provides more than one task service. Hence there is no Functional Cohesion.
     - Procedural - It does not exhibit procedural Cohesion since the sequence doesn't play an important part here
  2. On inspecting **Class CompleteWord**, we find the methods highly correlated to the defined task using the multitude of subtasks. Since the LTCC score of this class is high, this class is not the ideal class for cohesion. This is because the number of public methods in this class is very low. Due to which it is not getting used much apart from the intended purpose. The type of exhibited cohesion is **Logical and Procedural Cohesion.** Logical, since there are methods which are getting overloaded and procedural, since there are methods which get executed based on the sequence and ordering of the structure.
     <u>Unexhibited</u> Cohesion types -
     - Functional - Since there is more than one task at hand being handled by a single class, the functional type is ruled out

- Coincidental - All related methods being implemented are related
- Informational - Methods are related but do not have different entry points in the system.
- Communicational - Although sequence does matter in the given case, modules do not branch in to perform other unrelated tasks.

High Cohesion classes

a. **TextArea/RangeMap** - 0
b. **TextArea/SelectionManager** - 0

Why are these High - cohesion classes? -

1. On inspecting **class Rangemap**, we find that the correlation between the public classes mentioned in the Rangemap is pretty high. They all target only one task in the entire project, that is they work towards the same goal but are using similar yet independent attributes to cater to the function goal which is to perform line handling for a given range of lines on TextArea. Hence they exhibit high cohesion and the type of observed cohesion is **Functional and Informational Cohesion**
   Unexhibited Cohesion types -
   - **Logical** - Methods are linked in a relatable way.
   - **Coincidental** - All nonPublic methods being implemented are related
   - **Communicational** - Since there is not a high amount of modularity displayed this category is ruled out.
   - **Procedural** - Sequence does not matter in the given case.
2. On inspecting **Class SelectionManager**, it is a private class, to reduce its reusability. Given that, this is still an ideal class exhibiting high cohesion because most of the methods defined are short and concise, providing get and set services. However, there are large-scale methods that perform calculations determining selection ranges for given lines and text inside them, which performs math but does not use many class attributes. Hence the methods are relatively linked but independent of the class attributes and are linked in sequence. The type of cohesion observed is **Procedural and Informational and Functional cohesion.**
   Unexhibited Cohesion types -
   - Communicational - Since there is not a high amount of modularity displayed this category is ruled out.
   - Coincidental - All nonPublic methods being implemented are related
   - Communicational - Ordering of methods needed but methods aren't much separated.

How are they different from high/low cohesion classes with respect to each other and in the other metric(s)?

- **With respect to each other-**
  - ➢ High cohesion classes in LTCC are the ones that have well-defined public methods and exhibit strong correlations between them.
  - ➢ It was also observed that in those high cohesion classes, the usage of common variables and attributes was strong and there weren't many unused variables.

    i. LOC for those classes was also short.

- **With respect to another metric (for example “Lack of Cohesion(LCohesion)” metric) -**
    - ➢ All the High Cohesion classes(acc. To LTCC) have a corresponding “Low”/”Low-medium” Lack of Cohesion according to (LCohesion). Which justifies the LTCC metric as both the results coincide indicating that the degree of cohesion is high and detected correctly.
        i. Comparing the high cohesion classes of LTCC with different high cohesion classes in LCohesion, they exhibit a similar design pattern where there are multiple public methods with a high degree of coinciding attributes used over in the class and methods.
    - ➢ We observe stark differences in low cohesion classes (acc. to LTCC) when compared to its corresponding LCohesion values. LCohesion for Textaretransferhandler is low, which indicates that the cohesion is actually high, whereas the LTCC values indicate otherwise.
        i. When comparing the low cohesion classes to other metric worst cohesion classes, the patterns do match. They have almost no public methods or have no common attributes.

- **Lack of Cohesion of Methods (LCOM3):**
  Low Cohesion classes:
    a. **gui/BufferSwitcher**- 1.083
    b. **io/VFSManager**- 0.965

  Why are these low cohesion classes? -
    1. On inspecting **Class BufferSwticher**, this class instantiates the styles stored in the buffer for Textarea. It is noticeable that the methods are majorly getters and setters and updaters of styles present or adding new ones to/from buffer due to which the length is relatively short for methods while it has a decent number of usable attributes defined in the scope of the class. Hence it has **informational cohesion**, and to some extent, **logical cohesion** as well since there are multiple methods overloaded.
       <u>Unexhibited</u> Cohesion types -
        - Communicational - Since there is not a high amount of modularity displayed this category is ruled out.
        - Coincidental - All nonPublic methods being implemented are related
        - Procedural - Ordering of methods needed but methods aren't much separated.
        - Functional - While the methods are short, only a minor amount of classes are exhibiting the same goal.

    2. On inspecting **Class VFSManager,** we observe that this is a large instance class that serves as the backbone of the virtual Filesystem in Jedit itself. It involves at least 8 class variables along with other local method variables and many public methods. We see that on average every method independently uses 2 variables

at the very least for function calls and hence yields a Low LCOM3 score. This indicates that this large instance of a class has many highly correlated methods working inside and hence exhibits **functional, informational, and procedural cohesion.** Procedural since the sequence of the different functions is structured in a way that gets triggered in sequence one after the other.

Unexhibited Cohesion types -

- Communicational - while the sequence is of some importance here, no modules branch into performing minor small and unrelated tasks..
- Temporal - The application of time does not affect the logic much.
- Logical - Not many methods are overloaded

High Cohesion classes

a. **TextArea/ScrollLineCount** - 0
b. **TextArea/StandAloneTextArea** - 0

Why are these high cohesion classes? -

1. On inspecting **Class ScollLineCount**, although it is high cohesion class due to the modularity of the methods in the class, we observe that there are no class variables defined in the global scope, which indicates that the methods are structured to run independently and could be unrelated if need be to some extent. Due to this, the LCOM3 score is 0. It exhibits **coincidental and procedural cohesion**, procedural since the methods are linked via the sequence of their occurrence.

   Unexhibited Cohesion types -

   - Functional - More than one concrete definition of tasks present.
   - Informational - Does not target a shared data structure.
   - Logical - methods are not overall related to each other and do not override each other.
   - Temporal - no effect on the time-based execution.
2. On inspecting the **class StandAloneTextArea**, we again observe a similar instance of no class variables defined in global scope due to which there is 0 LCOM3 score. It exhibits **logical and procedural cohesion** since the methods are overridden and has sequential execution importance as suggested by the naming of the methods in the class.

   Unexhibited Cohesion types -

   - Functional - More than one concrete definition of tasks present.
   - Informational - Does not target a shared data structure.
   - Coincidental - methods are overall related to each other.
   - Temporal - no effect on the time-based execution.

How are they different from high/low cohesion classes with respect to each other and in the other metric(s)?

- **With respect to each other-**

- ➢ High cohesion classes in LCOM are the ones with coinciding attributes being used over the entire scope of the classes and methods.
- ➢ They also have a higher number of methods present in the class as compared to the Low cohesion classes.
    i. LOC for low cohesion classes was also short.

- **With respect to another metric (for example “Lack of Cohesion(LCohesion)” metric) -**
    - ➢ From the High Cohesion classes(acc. To LCOM), Class ScrollLineCount has a corresponding “Low” Lack of Cohesion according to (LCohesion). This justifies the LCOM metric as both results coincide indicating that the degree of cohesion is high and detected correctly. But the class StandAlonetextArea differs from the as the LCohesion corresponding value is “medium-high”.
        i. Comparing the high cohesion classes of LCOM with different high cohesion classes in LCohesion, they exhibit a similar design pattern where there are multiple methods with a high degree of coinciding attributes used over in the class and methods.
    - ➢ All the Low Cohesion classes(acc. To LCOM) have a corresponding “medium”/”high” Lack of Cohesion according to (LCohesion). This justifies the LCOM metric as both results coincide indicating that the degree of cohesion is low and detected correctly.
        i. When comparing the low cohesion classes to other metric worst cohesion classes, the patterns do match. They have almost no common **CLASS** attributes.

Coupling

- **Coupling between objects (CBO) -**
  - ➢ **Tightly coupled (highest coupling):**
    - **org.gjt.sp.jedit.jEdit**: CBO score - 93
    - **org.gjt.sp.jedit.browser.VFSBrowser**: CBO score - 49
  - ➢ **Loosely coupled (lowest coupling):**
    - **org.gjt.sp.jedit.gui.ExtendedGridLayout:** CBO score - 1
    - **org.gjt.sp.util.StandardUtilities:** CBO Score - 1

1. **org.gjt.sp.jedit.jEdit**: (CBO score 93)
   - This is the main class of the JEdit text editor.
   - It is responsible for setting up (invoking the methods) the GUI along with other connections, plugins, buffers, macros, startup scripts, language, and other properties.
   - For this setup, the main class depends heavily on other classes.
   - This class also has its methods for getting jedit properties, setting up the buffer and its properties, setting up views, and other methods.
   - These methods are utilized by other classes whenever required.
   - Since CBO doesn't care about the direction of dependency, the CBO score is the highest for this class.
   - All the methods of this class are independent and perform their respective functions when they are invoked by other classes and data is passed onto them.
   - This seems to resonate with **Data coupling** over all the other discussed types.
   - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
   - Since this class does not use global elements, this is **not Common coupling.**
   - There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**
   - This may/may not be Stamp coupling as all the method calls are passing data to the invoked methods. While the data passed is exact or more than needed is complex to determine, this **can pass as Stamp coupling.**

2. **org.gjt.sp.jedit.browser.VFSBrowser**: (CBO score 49)
   - This is the main class of VFS browser used in JEdit.
   - This is used as a dockable and is embedded in the VFSFileChoserDialog.
   - The VFSBrowser is responsible for file browser dialogs and windows, view buffers, and handling other directory features like editing, renaming, deleting as well as managing the clipboard.
   - These are some of the core functionalities of JEdit software and this class is used by several other classes to handle respective functionality.
   - Hence, this class has high coupling.
   - Each method in this class is independent and performs its specific tasks when called by other classes and given input from other classes.
   - This seems to be **Data coupling** as well.

- Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
- Since this class does not use global elements, this is **not Common coupling.**
- There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**
- This may/may not be Stamp coupling as all the method calls are passing data to the invoked methods. While the data passed is exact or more than needed is complex to determine, this **can pass as Stamp coupling.**

3. **org.gjt.sp.jedit.gui.ExtendedGridLayout: (**CBO score 1)
    - This is just a layout manager.
    - The class is responsible for placing components in a rectangular grid with variable cell sizes.
    - It supports colspans and rowspans.
    - Since this is responsible for just setting up the layout, it has a very low CBO score and low coupling.
    - All the methods of this class require the passing of inputs (data) for setting up and updating/ modifying the layout.
    - This makes the type of coupling **Data coupling.**
    - This **can also be Stamp coupling** as the classes invoking this class may/ may not pass more than the required data.
    - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
    - Since this class does not use global elements, this is **not Common coupling.**
    - There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**

4. **org.gjt.sp.util.StandardUtilities: (**CBO Score 1)
    - This class contains some basic standard utilities, several of which depend on JDK only.
    - Most of the utilities are helper functions.
    - Hence, this has very low coupling.
    - All these helper functions are passed with some data which is then processed and the transformed values are returned to the invoking class.
    - This makes this coupling too as **Data coupling.**
    - Again, this may/ may not be **Stamp coupling** as it depends on the invoking classes what data are they passing to this class, and its methods.
    - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
    - Since this class does not use global elements, this is **not Common coupling.**
    - There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**

- **Response for class (RFC) -**
  - ➢ **Tightly coupled (highest coupling):**
    - **org.gjt.sp.jedit.jEdit:** RFC score - 2113
    - **org.gjt.sp.jedit.search.SearchAndReplace:** RFC score - 1566
  - ➢ **Loosely coupled (lowest coupling):**
    - **org.gjt.sp.jedit.print.PrintPreviewModel:** RFC score - 29
    - **org.gjt.sp.jedit.bsh.Types:** RFC score - 45

1. **org.gjt.sp.jedit.jEdit**: (RFC score 2113)
   - As reported in the CBO section, this class has very high coupling along with a high CBO and RFC score.
   - SInce RFC is the set of methods that can potentially be executed in response to a message received by an object of that class and considering the high number of methods in the jEdit class and the high number of method calls, the RFC score of this class is the highest resulting in high coupling.
   - This is again, Data coupling, and not any other type of coupling, as discussed above.
2. **org.gjt.sp.jedit.search.SearchAndReplace: (**RFC score 1566)
   - This class is used for search within jEdit buffers.
   - Since this class is being used for a regular expression and literal search within the entire buffer, and has several methods that can be executed in response to a class, this is a high coupling class with a high RFC score.
   - The functions are passed with some data which is then processed(search, replace, etc) and the transformed values are returned to the invoking class.
   - This makes this coupling too as **Data coupling.**
   - Again, this may/ may not be **Stamp coupling** as it depends on the invoking classes what data are they passing to this class, and its methods.
   - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
   - Since this class does not use global elements, this is **not Common coupling.**
   - There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**

3. **org.gjt.sp.jedit.print.PrintPreviewModel: (**RFC score 29)
   - This class is responsible for the print view pane.
   - The methods of this class are just for setting and getting the print preview display
   - Hence, this is a very low coupling class with a very low RFC score.
   - The methods are invoked by other classes to get and set the preview. These are called by passing some data to them. Hence, this too is data coupling.
   - Again, this may/ may not be **Stamp coupling** as it depends on the invoking classes what data are they passing to this class, and its methods.
   - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**

- Since this class does not use global elements, this is **not Common coupling.**
- There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**

4. **org.gjt.sp.jedit.bsh.Types: (**RFC score 45)
    - This class has static routines that support type comparison and conversion in BeanShell.
    - The methods are pretty basic and hence have a very low coupling and RFC score.
    - Since the methods in this class are just for type comparison and conversion, they are invoked by passing some data and returning some data post-processing.
    - Hence, this too is **Data Coupling.**
    - This may/ may not be **Stamp coupling** as it depends on the invoking classes what data are they passing to this class, and its methods.
    - Since this module does not directly reference the contents of other modules, this is **not Content coupling.**
    - Since this class does not use global elements, this is **not Common coupling.**
    - There is no passing of control from one module to another and all the modules are invoked just by passing relevant data to them, this is **not Control coupling.**

Comparisons:

1. **Classes with highest coupling vs classes with lowest coupling:**
    - The classes with the highest coupling viz. jEdit, VFSBrowser, and SearchAndReplace have some of the most called methods and classes.
    - These classes are responsible for major functionalities of the jEdit program.
    - The jEdit class is responsible for the entire program, the VFSBrowser is responsible for setting up views and the Search and Replace class is responsible for functions related to searching and renaming.
    - On the other hand, the classes with the lowest coupling perform some extra helper functions and do not play a major role in this software. Hence they have low coupling.
    - Most of these low coupling classes are standard utilities, features that could be implemented in place instead of separate classes.(Although this would have resulted in code redundancy and ultimately in a code smell).

2. **Among the classes with the highest and lowest coupling according to different metrics:**
    - The jEdit class has the highest RFC and CBO scores. This means the class has one of the highest couplings.
    - The VFSBrowser has the second-highest CBO score but the third-highest RFC score while the Search and Replace has the second-highest RFC score and is amongst the top CBO scorers.

- This is due to the fact that SearchAndReplace has a higher total number of methods that can potentially be executed in response to a message received by an object of a class while VFSBrowser has a higher number of classes calling/called by this class.
- This is similar in the case of ExtendedGridLayout, StandardUtilities vs PrintPrefviewModel, and Types classes.
- ExtendedGridLayout, StandardUtilities has the lowest number of classes calling/called by this class while PrintPrefviewModel and Types classes have the lowest total number of methods that can be executed in response to calls.

Overall, the project has followed a pretty good approach in its design and implementation. The coupling and cohesion that we observed are all good/favorable types.

Tasks performed by each member:

We analyzed all the provided tools together and decided to use CodeMR for its simplicity and easy-to-read GUI.
We then used this tool together to measure coupling and cohesion for the entire project and analyzed the classes that had the highest and lowest scores amongst the metrics that we chose. We did a lot of rough work for the analysis and used the same to complete this report.
Post analysis, we divided the task of reporting our findings.
Deep - Wrote the report for Cohesion.
Aditya - Completed the report for Coupling.

Contribution information:
Aditya Khandare - 50%
Deep Gosalia - 50%

Link to folder in which analysis is saved:
https://github.com/adiK97/JeditTeam05/tree/main/Assignment%203%20-%20Coupling%20and%20Cohesion