**CS 6356: Software Maintenance, Evolution & Re-engineering**

**Assignment 3: Coupling and Cohesion**

**TEAM 13**

TEAM MEMBERS:

1. GAURAV SHARMA (GXS230001)
2. SHALIN KAJI
3. ADITYA KRISHNA

Tools Used for Computing the Metrices

1. Metrics Reloaded for IntelliJ
2. Inspect Code – Intellij
3. CodeMR - Intellij

**Description of the metrics selected: -**

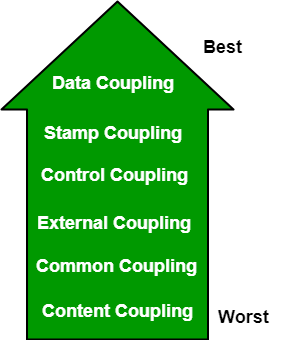
Coupling and Cohesion are two key concepts in software engineering that are used to measure the quality of a software system’s design.

1. **COUPLING**

Coupling refers to the degree of interdependence between software modules.

**High coupling** means that modules are closely connected and changes in one module may affect other modules.

**Low coupling** means that modules are independent and changes in one module have little impact on other modules.



**Types of Coupling:**

**Data Coupling:** The modules are considered data coupled if their dependence is predicated on the fact that they exchange solely data for communication. The components of a data coupling are separate from one another and communicate via data. Tramp data is absent from module communications. Customer billing system, for instance.

**Stamp Coupling** The entire data structure is transmitted from one module to another in a stamp coupling. Thus, tramp data is involved. Efficiency considerations might make it necessary; the intelligent designer, not a slothful coder, made this decision.

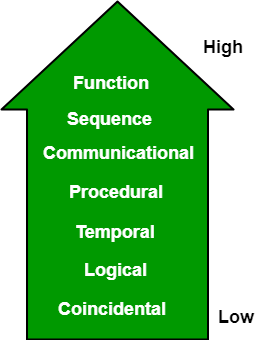
**Control Coupling:** The modules are referred to as control linked if they exchange control information with one another. If arguments allow for factoring and functionality reuse, that can be good; however, it can be negative if the parameters show entirely different behavior. A sort function that accepts the comparison function as an argument is an example.

**External Coupling:** When modules are coupled externally, they rely on one another rather than the software or a certain kind of hardware that is being produced. Device format, external file, protocol, etc.

**Common Coupling:** Global data structures and other shared data are present in the modules. To assess the impact of a modification to global data, all modules that access it must be tracked back to. Thus, it has drawbacks such as limited maintainability, less control over data access, and difficulties reusing modules.

**Content Coupling:** One module can change the data of another module or transfer control flow from one module to the other in a content coupling. The worst kind of coupling is this one, therefore stay away from it.

**Cohesion:** Cohesion is a measure of the degree to which the elements of the module are functionally related. It is the degree to which all elements directed towards performing a single task are contained in the component. Basically, cohesion is the internal glue that keeps the module together. A good software design will have high cohesion.



**Types of Cohesion:**

**Functional Cohesion:** Every essential element for a single computation is contained in the component. A functional cohesion performs the task and functions. It is an ideal situation.

**Sequential Cohesion:** An element outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

**Communicational Cohesion:** Two elements operate on the same input data or contribute towards the same output data. Example- update record in the database and send it to the printer.

**Procedural Cohesion:** Elements of procedural cohesion ensure the order of execution. Actions are still weakly connected and unlikely to be reusable. Ex- calculate student GPA, print student record, calculate cumulative GPA, print cumulative GPA.

**Temporal Cohesion:** The elements are related by their timing involved. A module connected with temporal cohesion all the tasks must be executed in the same time span. This cohesion contains the code for initializing all the parts of the system. Lots of different activities occur, all at unit time.

**Logical Cohesion:** The elements are logically related and not functionally. Ex- A component reads inputs from tape, disk, and network. All the code for these functions is in the same component. Operations are related, but the functions are significantly different.

**Coincidental Cohesion:** The elements are not related(unrelated). The elements have no conceptual relationship other than location in source code. It is accidental and the worst form of cohesion. Ex- print next line and reverse the characters of a string in a single component.

**Advantages of low coupling:**

* Enhanced maintainability: Modifying or replacing individual parts without affecting the system as a whole is made easier by low coupling, which lessens the effect of changes in one module on other modules.
* Enhanced modularity: Code may be produced and tested independently thanks to reduced coupling, which increases code's modularity and reusability.
* Better scalability: The system can be scaled more easily as needed since little coupling makes it easy to add new modules and remove old ones.

**Advantages of high cohesion:**

* Better readability and understandability: Clear, focused modules with a single, well-defined goal are the outcome of high cohesiveness, which facilitates developer comprehension and modification of the code.
* Improved error isolation: Strong cohesiveness lessens the possibility that modifications to one module component will have an impact on other components, which facilitates
* Enhanced dependability: Strong cohesiveness results in modules that operate more consistently and with less error-proneness, enhancing the system's overall dependability.

**Disadvantages of high coupling:**

* Enhanced complexity: A system with a high coupling level has more interdependencies between its parts, which makes it more complex and challenging to comprehend.
* Less flexibility: It is harder to replace or alter specific parts of a high coupling system without also altering the system as a whole.
* Reduced modularity: Code that has a high coupling level is less modular and reusable since it is more difficult to build and test parts separately.

**Disadvantages of low cohesion:**

* Greater code duplication: When components that should work together are divided into different modules, a lack of cohesiveness can result in an increase in code duplication.
* Reduced functionality: Modules with low cohesiveness may have pieces that aren't meant to be together and serve no obvious purpose, which limits their usefulness and makes maintenance more difficult.
* Difficulty comprehending the module: A module with low cohesiveness may be more difficult for developers to comprehend, which can result in mistakes and unclear behavior.

**Measuring and interpreting Cohesion**

Three non-trivial classes with the **highest cohesion**: -

1. For cohesion, the first metric we will measure is **WMC.** One excellent illustration of this is the TextArea.java class, which can be found under org/gjt/sp/jedit/textarea. We had a good amount of time to study this long lesson, which we used for the second homework assignment. However, utilizing Metrics Reloaded, we discovered that this had one of the lesser alternatives for average operation complexity (OCavg) but the second-highest weighted metric complexity (WMC). We used the menu item for complexity metrics to conduct this test.

The WMC and OCavg had precise values of 799 and 2.93 and OCmax is 2 respectively. Since all of the methods and other material in this class are tied to the main function of creating a text area for the user to access JEdit, the class employs good coherence. We believed that this class was an excellent illustration of **communication cohesion** because the various teaching strategies all contributed to the formation of the text area. Upon examining the code, it is evident that the original TextArea constructor is invoking several methods that are specified within the same class. Several scrolling capabilities, including TextAreaPainter(), are developed in this class. In addition, numerous other methods make use of various techniques that are also defined here, serving as a means of internal class communication. The goal of each method is to produce a text area, and they can accomplish this by calling one another.

1. The different widget factory classes were another cohesive group of classes. Other irrelevant classes existed, but we won't investigate them in accordance with the assignment's guidelines. The WMC was 1.00, OCmax was 1 and the OCavg was also 1.00 for the widget factory classes. These were the lowest values that a class could obtain with this tool. We'll be specifically looking at the BufferSetWidgetFactory. This can be found in the directory org/gjt/sp/jedit/gui/statusbar.

This class also makes advantage of **communicational cohesion** since, after the initial getWidget function, bufferSetWidget is created, calling the static class BufferSetWidget, which has all the methods required to create a buffer set widget. We have add and delete notify methods in the constructor class to add to the bus.Additionally, mechanisms for updating/changing properties and mouse listeners are provided.Aside from being communicational, it may also be regarded as functional because its main objective is to set up the buffer set widget. Functional cohesion is based on a program that concentrates on a single action or goal, which this program does do.

1. The AddAbbrevDialog.java class, which is found in org/gjt/sp/jedit/gui, is the final class we will examine for strong cohesiveness. This also yielded really poor results in our metrics test. Both the WMC, OCmax and the OCavg were 2. By examining this class, we can see how each of its methods serves the same purpose—namely, to add an abbreviation to the dialog. The class contains methods specified to add to the ultimate goal and private variables from other classes. Another illustration of cohesive communication is this. The actionPerformed and keyPressed methods are those that are specified in the Action and Key Handler classes. These are rather basic methods that are invoked within the AddAbbrevDialog constructor.

This might be regarded as functional cohesion when considering the other types of cohesiveness. Although the two sub-classes are necessary for the main AddAbbrevDialog, the other classes felt informative. All these classes and methods operate toward the same end, which is to provide an abridged dialog.   
  
Now that we have looked at the examples of high cohesion, we will be looking at **low cohesion** examples. These classes will have high values for intricate operations. The main measure we'll be attempting to use here makes sense because we anticipate that many of the classes will have several actions that can add bloat and decrease usefulness while attempting to achieve a single goal.

1. We will examine the ParseTokenManager.java class first. The directory org/gjt/sp/jedit/bsh contains this. With an average of 14.36, the highest value was found for the WMC, which is 718. At 241, the highest operational complexity was also the most.This class appears to be quite disjointed, as it appears to be a dump for several parsing techniques. While the methods all seem to make sense and serve distinct parsing tasks, this is a nice illustration of logical cohesiveness. Each approach, in contrast to the classes above, addresses a somewhat different problem.The mass code dump also significantly diminishes the readability of this code. Numerous approaches should be simplified as well as streamlined.
2. We will examine the ExtendedGridLayout.java class next. You may find this at org/gjt/sp/jedit/gui. IT's WMC was 136 and its OCavg was 8.00. Its operational complexity (OCmax) peaked at 49. Though not nearly as sophisticated as the ParseTokenManager class, this class also has logical coherence. The techniques covered in this course mostly deal with general layout adjustments. It is not a smart coding practice to just group all of the layout functions into one big class. A great deal of procedures are bloated, and many are empty. RemoveLayoutComponent is an example of an empty method, and getSize, which contains more than 300 lines of code, is an example of a bloated method.

The reasoning behind these approaches is generally sound, but because they are not all aimed at the same thing, they lack functional cohesiveness. In addition to exhibiting logical coherence, it also demonstrates coincidental cohesiveness because some of the approaches are merely unrelated. A few instances include the comparison between the layout methods and the getSize or buildGrid methods. This class might easily be divided into three or even four classes, which would greatly increase the relevance of the content. It makes logical that the layout methods are required in order to work with the getSize or buildGrid methods.

1. CodeWriter.java is the final class we will examine for low cohesiveness. The OCavg score for this class was 7.35, while the WMC score was 228. There was a maximum of 59 operational complexity. In org/gjt/sp/jedit/bsh/org/objectweb/asm is where you may find this class. This class is quite extensive as well, with about 2000 lines of code. Many of the methods in this class make sense, however they don't perform well. Similar to the previous class, this one also demonstrates logical coherence since all of the variables and methods produced are meant to build the code writer. However, the methods may be broken down into more manageable and easily understood classes.All of the first almost 300 lines of code are comment-heavy variable declarations. Not until line 490 is the constructor declared itself.

Declaring the variables in a different class and calling them as needed would be the best approach. Functional cohesion would be the optimum state of affairs. In order to do the task, the CodeWriter class would therefore require the necessary methods. Examining the code more closely reveals that we have logical ways for implementing the interface. The issue is that, once these methods are defined, more "utility methods" are declared, which causes the bloat to increase. Even while combining all of them makes sense logically, it isn't practical.

It is safe to state that the high cohesion classes are objectively simpler to comprehend and deal with now that we have looked at three classes with high cohesion and three classes with low cohesion. It's interesting to compare ParseTokenManager (PTM) with BufferSetWidgetFactory (BSWF) because we have the lowest and maximum complexity metrics, as determined by the Metrics Reloaded tool. The actions and methods in the BSWF class are all related and easily readable. They all combine to achieve the objective. The development of the buffer set widget was the only objective, so while we concluded that there was communicational cohesion, there were indications of functional cohesion as well.

On the other hand, the PTM contained numerous methods with similar names and purposes, but they all had distinct objectives. Token parsing was the core of most of the functions, but it wasn't a good idea to keep them all in one class. It's like creating a class called Course and then adding methods to it for every course offered by the organization. This is another instance when it would be wiser to divide the class into smaller sections that were relevant to a certain subject or even a topic within the subject. With the BSWF, we witnessed exactly precise execution. As previously mentioned, there were numerous classes for the various widgets. It separated each widget into its own class rather to implementing them all in a single class. The cohesiveness was significantly higher with those since it was considerably better coding practice.

COHESION METRICS: -

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| CLASS | OCavg | OCmax | WMC | COUPLING |
| org/gjt/sp/jedit/textarea | 2.93 | 2 | 799 | HIGH COHESION |
| org/gjt/sp/jedit/gui/statusbar | 1.00 | 1.00 | 1.00 |
| org/gjt/sp/jedit/gui | 2.00 | 2.00 | 2.00 |
| org/gjt/sp/jedit/bsh | 14.36 | 241 | 718 | LOW COHESION |
| org/gjt/sp/jedit/gui | 8.00 | 49.00 | 136 |
| org/gjt/sp/jedit/bsh/org/objectweb/asm | 7.35 | 59 | 228 |

**Measuring and interpreting Cohesion**

**OCmax :-**

* OCmax calculates the cohesion based on the most interconnected pair of methods within the module.
* High OCmax indicates high cohesion because it means that there is a strong interdependence between methods, suggesting that they work closely together to achieve a common goal.
* Low OCmax suggests low cohesion as it implies that methods within the module are less interconnected and may have diverse or unrelated responsibilities.

**OCavg :-**

* OCavg calculates the cohesion based on the average interdependence among all pairs of methods within the module.
* High OCavg indicates high cohesion because, on average, the methods in the module are relatively interconnected, contributing to a cohesive unit of functionality.
* Low OCavg suggests low cohesion as it means that, on average, methods within the module have weaker interdependencies, which could be a sign of scattered or unrelated responsibilities.

**JEDIT :-**

Three non-trivial classes with the **highest cohesion using OCmax** : -

1. ParserTokenManager.java (org/gjt/sp/jedit/bsh/ParserTokenManager.java)

For this class **OCmax is 241** which is the highest value in the project.

The ParserTokenManager class is cohesive mostly in its functional sense. When components of a module (class) are connected through the performance of a particular task or function, this is known as functional cohesiveness. In this instance, the class's sole purpose is token parsing, and every one of its variables and methods is directly tied to this ability. The class exhibits functional coherence in the following ways:  
Related Methods: The parsing of tokens is closely related to all of the methods in the class (jjStopStringLiteralDfa\_0, jjStartNfa\_0, jjStopAtPos, jjStartNfaWithStates\_0, jjMoveStringLiteralDfa0\_0, etc.). Token parsing involves various stages and aspects, such as beginning, pausing, and progressing throughout the process, which are all aided by each approach.   
Overall, by concentrating only on token parsing operations and grouping relevant functionality within the class, the ParserTokenManager class demonstrates functional coherence. Cohesion of this kind is advantageous since it increases the class's maintainability, readability, and reusability for token parsing tasks.

1. TextUtilities.java (org/gjt/sp/jedit/TextUtilities.java)

For this class **OCmax is 50** which is the very value in the project

This class demonstrates strong cohesiveness by assembling related techniques that work with comparable data or complete comparable tasks. The degree to which the duties of methods inside a class are focused and closely related is referred to as cohesion. In this instance, the course focuses on similar activities such as text formatting and string manipulation.  
This class demonstrates a **Functional Cohesion**. When methods within a class carry out comparable tasks or cooperate to accomplish a particular functionality or objective, this is known as functional cohesiveness. The following techniques provide this high level of functional cohesion:  
indexIgnoringWhitespace(String str, int index): This function returns the index of characters in a string that are not whitespace up to a specified index. It functions with indexing and character manipulation.   
By combining techniques that cooperate to carry out text formatting, character analysis, and string manipulation, the class exhibits strong Functional Cohesion overall.

1. TextAreaInputHandler.java (org/gjt/sp/jedit/input/TextAreaInputHandler.java)

For this class **OCmax is 20** which is the very value in the project

Both functional coherence and high cohesion are displayed by the TextAreaInputHandler class. The degree of emphasis and relatedness among a class's or module's obligations is referred to as cohesion. When methods within a class are grouped together based on completing a certain task or closely related duties, this is referred to as functional cohesiveness.  
Because all of the properties and methods in the TextAreaInputHandler class are concerned with managing input events for a text area (TextArea), there is a high degree of cohesiveness among them. Let's examine the ways in which this class exhibits functional cohesion:

The TextAreaInputHandler class exhibits functional cohesiveness by combining methods and fields that are directly linked to managing input events for the text area. It has strong cohesion since it concentrates on a single duty or task—managing important events for a text region.

Three non-trivial classes with the **lowest cohesion using OCmax** : -

1. TextAreaDialog (org/gjt/sp/jedit/gui/TextAreaDialog.java)

The OCmax value for this class is 1 which is the lowest value of all.

Because it has several tasks that are not directly related to one another, this TextAreaDialog class lacks cohesiveness. When a class has several unconnected functions or duties, it lacks coherence and the code becomes more difficult to comprehend, maintain, and reuse. In this instance, the class integrates dialog creation, action handling, error management, and UI logic—tasks that, in a perfect world, would be divided into various classes or modules to improve coherence.  
The different forms of cohesiveness this class demonstrated are broken down as follows: **Procedural Cohesion**:  
Methods like init(), ok(), and cancel(), where the stages are set to achieve a certain task (e.g., initializing UI components, closing the dialog on OK or cancel), are prime examples of procedural cohesiveness.  
By separating out certain processes inside their own scope, breaking these methods down into smaller, more focused ways can help with readability and maintainability.

1. Abbrevs.java (org/gjt/sp/jedit/Abbrevs.java)

Due to the fact that this Abbrevs class contains multiple unrelated functionality within it, it lacks cohesiveness. The degree of relationship between a class's data and methods is called cohesion. Low cohesiveness indicates that the class is working on a variety of unrelated, diversified tasks, which makes it difficult to focus and clearly define its design.  
Different types of cohesiveness can be identified in the given class:

**Communicational Cohesion**: To provide error feedback and allow user involvement (AddAbbrevDialog), the class communicates with UI components (javax.swing.UIManager).  
For text expansion and manipulation, it additionally communicates with the buffer and text area components (Buffer, JEditTextArea, and View).  
These communication duties, however, have little bearing on one another or the main objective of abbreviation management.

1. StatusBar.java (org/gjt/sp/jedit/gui/StatusBar.java)

The main reason this StatusBar class lacks cohesiveness is that it contains several unrelated functionalities in one class. When duties and assignments in a class are not tightly related to one another or are not focused, it is referred to as having low cohesiveness. This can cause confusion and make it more difficult to maintain and comprehend the code. The class is exhibiting both temporal and functional cohesiveness in this instance.

When various techniques or components of a class are grouped together because they carry out similar duties, this is known as functional cohesion. Nevertheless, it appears that the propertiesChanged() and addNotify() methods in the provided class handle distinct tasks unrelated to one another or the primary goal of the class.

Three non-trivial classes with the **highest cohesion using OCavg**: -

1. ParserTokenManager.java (org/gjt/sp/jedit/bsh/ParserTokenManager.java)

For this class **OCmax is 241** which is the highest value in the project.

The ParserTokenManager class is cohesive mostly in its functional sense. When components of a module (class) are connected through the performance of a particular task or function, this is known as functional cohesiveness. In this instance, the class's sole purpose is token parsing, and every one of its variables and methods is directly tied to this ability. The class exhibits functional coherence.  
By concentrating only on token parsing operations and grouping relevant functionality within the class, the ParserTokenManager class demonstrates functional coherence. Cohesion of this kind is advantageous since it increases the class's maintainability, readability, and reusability for token parsing tasks.

1. MouseWheelHandler (org/gjt/sp/jedit/textarea/TextArea.java)

The value for Ocavg here is 13 which is very high compared to other classes.

This MouseWheelHandler class merges several unrelated functionality into a single method, mouseWheelMoved, which results in high cohesiveness. A class or approach that attempts to handle too many unrelated tasks is said to have high coherence. The mouseWheelMoved method in this instance manages a variety of mouse wheel events, however it combines disparate behaviors, which leads to a High Cohesion.

The type of cohesion exhibited here is control cohesion or coincidental cohesion. Control cohesion occurs when elements within a module are combined based on the need for common control logic, such as switch statements or if-else conditions. In this case, the method combines different functionalities related to mouse wheel handling, scrolling, caret movement, and text selection, resulting in control-driven cohesion.

1. ActionHandler (org/gjt/sp/jedit/gui/CloseDialog.java)

This class ActionHandler exhibits High cohesion because it contains multiple unrelated functionalities within the actionPerformed method. Cohesion refers to the degree to which the elements of a module (in this case, a class) are related and work together to achieve a single purpose. High cohesion means that the elements of the class are not strongly related or do not work towards a single goal efficiently.

In the provided ActionHandler class, the actionPerformed method handles four distinct actions based on different sources (selectAll, save, discard, and cancel). These actions seem to be related to some user interface components or application functionalities, but they are not directly related to each other in terms of functionality or purpose.

The type of cohesion exhibited in this class is known as "Coincidental Cohesion". Coincidental cohesion occurs when parts of a module are grouped together arbitrarily, often because they happen to be implemented in the same area of code or because they are handled by the same method, even though they are not logically related.

Three non-trivial classes with the **Lowest cohesion using OCavg**: -

1. TextAreaDialog (org/gjt/sp/jedit/gui/TextAreaDialog.java)

The Ocavg value for given class is 1.00 which is the lowest value for Ocavg.

This TextAreaDialog class exhibits low cohesion because it combines multiple unrelated functionalities within a single class. Cohesion refers to the degree to which the elements within a module (class in this case) are related or focused on a single purpose. Low cohesion means that the class is handling multiple responsibilities or concerns that are not strongly related to each other.

1. WarpWidgetFactory.java (org/gjt/sp/jedit/gui/statusbar/WrapWidgetFactory.java)

The Ocavg value for given class is 1.00 which is the lowest value for Ocavg.

This Java class WrapWidgetFactory exhibits low cohesion due to its inclusion of multiple responsibilities within a single class.

Font Metrics and Dimension Calculations:

The WrapWidget class also includes logic for calculating font metrics and dimension adjustments based on the current font. This additional responsibility of managing UI dimensions adds to the class's lack of cohesion.

The type of cohesion exhibited in this class is Coincidental Cohesion. Coincidental cohesion occurs when different responsibilities are grouped together in a class without a clear logical connection. In this case, the class combines unrelated functionalities such as widget creation, event handling, UI updates, property management, and dimension calculations.

1. TaskMonitor.java (org/gjt/sp/jedit/gui/TaskMonitor.java)

The Ocavg value for given class is 1.08 which is the lower compared to other values for Ocavg in the whole project.

The TaskMonitor class exhibits low cohesion because it combines multiple responsibilities that are not directly related to each other. Low cohesion refers to a design issue where a class has too many responsibilities or functionalities that are not closely related. In this case, the class is responsible for managing a GUI component (JPanel), handling tasks (TaskListener methods), and managing the data displayed in a table (TaskTableModel).

the class exhibits low cohesion due to mixing GUI component management, task handling, data management, and event handling in a single class. To improve cohesion, the responsibilities should be divided into separate classes, each handling a specific concern such as GUI management, task management, data management, and event handling.

**Mango**

Three non-trivial classes with the **highest cohesion using OCmax** : -

1. MiscDwr.java (com/serotonin/mango/web/dwr/MiscDwr.java)

OCmax value for this class is 28 which is highest in the non trivial classes.

This Java class MiscDwr exhibits high cohesion by grouping related functionalities together, which is a characteristic of functional cohesion. Let's break down how this class demonstrates functional cohesion and why it doesn't exhibit other types of cohesion:

Functional Cohesion:

* The class primarily deals with various operations related to handling events, user actions, and documentation items. These functionalities are closely related and form a logical unit within the context of the application's functionality.
* Methods like toggleSilence, silenceAll, acknowledgeEvent, acknowledgeAllPendingEvents, toggleUserMuted, and getDocumentationItem all revolve around managing different aspects of event handling, user interactions, and documentation retrieval.
* The class also includes a method jsError that handles JavaScript errors, which is somewhat related to the overall functionality of the class in terms of error handling.

Contrast with Other Types of Cohesion:

* Coincidental Cohesion: This class does not exhibit coincidental cohesion because its methods are not randomly grouped together. They are all related to managing different aspects of events, user actions, and documentation items, which align with the application's purpose.
* Logical Cohesion: While the methods are logically grouped to handle related tasks, the class does more than just processing data in a sequential or logical manner. It encompasses a range of functionalities related to event management, user interactions, and error handling.
* Temporal Cohesion: There is no specific temporal relationship or dependency among the methods in terms of timing or sequence of execution.
* Procedural Cohesion: This class does not exhibit procedural cohesion, as it doesn't represent a single step in a larger process or procedure.
* Communicational Cohesion: Although there is some communication happening within methods (e.g., jsError method logs a warning), the primary focus of the class is not on communication between different modules or components. It's more about performing specific actions based on user input or system events.
* Informational Cohesion: While the getDocumentationItem method deals with retrieving information, the class as a whole is not primarily focused on processing or managing information.

1. **VMStatDataSourceRT.java**

(com/serotonin/mango/rt/dataSource/vmstat/VMStatDataSourceRT.java)

The OCmax value for given class is 24.

This Java class, VMStatDataSourceRT, exhibits high cohesion by encapsulating related functionality within the class itself. Cohesion refers to the degree to which the elements within a module or class belong together. In this case, the class demonstrates functional cohesion, where all methods and attributes are related to the functionality of managing a VMStat data source.

Here's how the class exhibits high functional cohesion:

* Purpose: The class is designed specifically to handle VMStat data sources, including initializing, polling, and handling data.
* Attributes: The class contains attributes such as vo (VMStatDataSourceVO), vmstatProcess, in, attributePositions, terminated, and log, all of which are directly related to managing VMStat data and its processing.
* Methods: Methods like initialize(), terminate(), beginPolling(), run(), readParts(), and readError() are focused on specific tasks related to handling VMStat data sources, such as initializing the data source, terminating it, starting polling, running the data processing loop, parsing data parts, and handling errors.

Now, let's contrast why it is not exhibiting other types of cohesion:

* Coincidental Cohesion: This type of cohesion occurs when elements in a module or class are grouped together arbitrarily, without a clear logical relationship. The VMStatDataSourceRT class does not exhibit coincidental cohesion because all its methods and attributes are directly related to managing VMStat data sources; there are no unrelated elements grouped together arbitrarily.
* Logical Cohesion: Logical cohesion involves grouping elements that are logically related but not functionally related. The class does not exhibit logical cohesion because all its elements are functionally related to handling VMStat data sources.
* Temporal Cohesion: Temporal cohesion occurs when elements are grouped together because they are executed at the same time. While the class has methods like initialize() and run() that are executed at different times, these methods are still functionally related to managing VMStat data sources, not just grouped together based on temporal execution.

1. **DataSourceEditDwr.java**

(com/serotonin/mango/web/dwr/DataSourceEditDwr.java)

The DataSourceEditDwr class exhibits high cohesion, specifically functional cohesion, based on its implementation of related functionality within the same class. Functional cohesion refers to the degree to which elements within a module (in this case, a class) are functionally related and contribute to a single well-defined task or purpose.

Here's how the class demonstrates functional cohesion:

editInit() Method: This method initializes the editing process for a data source and returns an internationalized response object containing data relevant to the editing operation.

tryDataSourceSave() Method: This method attempts to save a data source, validates it, and adds relevant data to the response object if the save operation is successful.

Contrasting with other types of cohesion:

* Coincidental Cohesion: This occurs when elements within a module are grouped together arbitrarily and do not contribute to a single purpose.
* Logical Cohesion: This type of cohesion involves grouping elements based on their logical relatedness, which may not necessarily correspond to a single well-defined task.

Three non-trivial classes with the **Lowest cohesion using OCmax**

1. **PointValueDao.java**  (com/serotonin/mango/db/dao/PointValueDao.java)

The OCmax value for this class is 1 which is the lowest value in the whole project.

The given PointValueDao class exhibits functional cohesion, which is the lowest level of cohesion. Functional cohesion occurs when elements of a module (in this case, methods within the PointValueDao class) are grouped because they all contribute to a single well-defined task or function, which in this case is saving point values.

Here's how the class demonstrates functional cohesion and why it does not exhibit other types of cohesion:

**Functional Cohesion:**

All methods in the class are related to saving point values in different scenarios (synchronously, asynchronously, handling image data, etc.).

The class has methods like savePointValueSync and savePointValueAsync that directly contribute to the task of saving point values.

The savePointValueImpl method is responsible for the actual implementation of saving point values, including handling different data types and sources.

Contrast with Other Cohesion Types:

* Coincidental Cohesion: There are no methods or elements in the class that are coincidentally grouped. Every method serves a clear purpose related to saving point values.
* Logical Cohesion: While there is some logical flow in the class, such as handling retries and concurrency, these are still part of the overall task of saving point values rather than being a separate logical unit.

1. **DataSourceDao.java** (com/serotonin/mango/db/dao/DataSourceDao.java)

The OCmax value for this class is 2.

This Java class, DataSourceDao, exhibits low cohesion, specifically functional cohesion. Functional cohesion is achieved when elements within a module (in this case, the class) perform tasks that are closely related and contribute to a single well-defined purpose, such as managing data sources in this case. It shows the functional cohesion.

Contrast with Other Types of Cohesion:

* Coincidental Cohesion: This class does not exhibit coincidental cohesion, which occurs when elements within a module are grouped together arbitrarily and do not have a clear, meaningful relationship. In this class, all methods and attributes are related to data source management, indicating a purposeful grouping of elements.
* Logical Cohesion: While the class does involve logical operations such as condition checks and data manipulation, these operations are all related to data source management, making it primarily functionally cohesive rather than logically cohesive.
* Temporal Cohesion: Temporal cohesion involves grouping elements based on when they are executed. In this class, methods related to data source operations are grouped together regardless of when they are executed, so it does not exhibit temporal cohesion.

1. **CustomViewPoint.java**

(com/serotonin/mango/view/custom/CustomViewPoint.java)

The provided Java class CustomViewPoint exhibits low cohesion, specifically functional cohesion. Functional cohesion is characterized by grouping related functionalities and operations together within a class. The class demonstrates low functional cohesion.

Contrast with other types of cohesion:

* Logical Cohesion: This class doesn't exhibit logical cohesion, which involves grouping functionalities based on logical relationships that may not be related by purpose. The functionalities in CustomViewPoint are logically related to creating custom views for data points.
* Temporal Cohesion: Temporal cohesion involves grouping functionalities based on the timing of their execution. The CustomViewPoint class does not exhibit temporal cohesion as it focuses on creating custom views regardless of timing or sequence of operations.
* Procedural Cohesion: Procedural cohesion involves grouping functionalities based on procedural flow or sequence. While the CustomViewPoint class follows a procedural flow in its createStateImpl method, its primary focus is on functional aspects related to custom view creation.

Three non-trivial classes with the **highest cohesion using OCavg** : -

1. **ImageCharServlet.java** (com/serotonin/mango/web/servlet/ImageChartServlet.java)

The OCavg value for this class is 7.33 which is a very high value.

This ImageChartServlet class exhibits high functional cohesion. Functional cohesion refers to a situation where elements of a module are related by performing a single function or task. In this class, all the methods and fields are related to the functionality of generating and caching image charts based on incoming requests.

Contrast with Other Types of Cohesion:

* Logical Cohesion: The class does exhibit some logical cohesion as it deals with tasks related to image chart generation and caching. However, it focuses more on functional aspects (generating, caching, and purging) rather than logical relationships between elements.
* Temporal Cohesion: While there is a temporal aspect in the tryCachePurge method (purging cache based on time intervals), it's not the primary focus of the class. The main focus is on image chart generation and caching.
* Procedural Cohesion: The class does not exhibit procedural cohesion as it's not primarily focused on a sequence of steps or procedures. Instead, it focuses on a specific task related to image chart handling.
* Communicational Cohesion: This type of cohesion is about elements that operate on the same input data or share intermediate results. In this class, methods communicate to achieve the overall task of generating and caching image charts, but it's not the primary mode of cohesion.

1. **DataSourceUtils.java**

(com/serotonin/mango/rt/dataSource/DataSourceUtils.java)

The Ocavg value for this class is 7.00

This Java class DataSourceUtils exhibits high cohesion by encapsulating related functionalities within its methods. Cohesion refers to the degree to which elements within a module (such as a class) belong together and work towards a common purpose. Let's analyze how this class demonstrates high cohesion and what type of cohesion it represents:

Functional Cohesion:

The class focuses on providing utility methods related to data source handling (getValue, getValueTime).

Each method performs a specific, well-defined function related to data parsing and value retrieval.

The methods work together to achieve the common goal of processing data and returning appropriate MangoValue objects.

This class is not exhibiting other types of cohesion such as:

* Coincidental Cohesion: There are no unrelated functionalities grouped together just by coincidence. All methods are directly related to data source handling and value extraction.
* Logical Cohesion: While the class does follow logical patterns and flows for data processing, its main focus is on functional and informational cohesion rather than organizing code based solely on logical operations.
* Temporal Cohesion: There is no specific time-related grouping or operations that would classify this class as exhibiting temporal cohesion. The methods operate independently of time-related considerations.

1. EventManager.java (com/serotonin/mango/rt/EventManager.java)

This EventManager class exhibits high cohesion primarily through functional cohesion. Let's break down the aspects of cohesion observed and why it contrasts with other types of cohesion:

Functional Cohesion (High Cohesion):

The class focuses on a single well-defined task: event management, including raising events, returning events to normal, canceling events, and handling event lifecycle.

Methods such as raiseEvent, returnToNormal, cancelEventsForDataPoint, cancelEventsForDataSource, and cancelEventsForPublisher are all related to event management and manipulation.

Contrast with Other Cohesion Types:

* Coincidental Cohesion: There are no randomly grouped methods or attributes that lack a clear relationship. Everything revolves around event management.
* Logical Cohesion: While there is logical flow within methods, the class's primary cohesion is functional, centered on event management rather than logical operations.
* Temporal Cohesion: There are no methods grouped solely by the timing of their execution. All methods are related to event handling throughout the application's lifecycle.
* Procedural Cohesion: Although methods follow a sequence of steps, they are tightly related to event handling rather than general procedural tasks.
* Communicational Cohesion: While there is communication with DAOs and user-related operations, these are part of event management and don't form a separate cohesive group.

Three non-trivial classes with the **Lowest cohesion using OCavg** : -

1. ReportDao.java (com/serotonin/mango/db/dao/ReportDao.java)

The provided class ReportDao exhibits low cohesion because it combines several unrelated responsibilities and functionalities within a single class. Cohesion refers to how closely related and focused the responsibilities of a class are. Low cohesion occurs when a class has multiple responsibilities that are not closely related to each other.

Here are the observations on the cohesion type and why it is not another type out of the listed types:

Low Cohesion Type: Coincidental Cohesion

Coincidental cohesion occurs when different parts of a class are grouped together without any meaningful relationship.

Contrast with Other Cohesion Types:

* Logical Cohesion: Logical cohesion involves grouping methods that logically belong together. The PointInfo class does not exhibit logical cohesion because its methods do not logically relate to each other.
* Temporal Cohesion: Temporal cohesion involves grouping methods that are executed at the same time. The methods in PointInfo are not necessarily executed at the same time, so it does not exhibit temporal cohesion.
* Procedural Cohesion: Procedural cohesion involves grouping methods that are related to a specific procedure or process. The methods in PointInfo do not form a cohesive procedure, so it does not exhibit procedural cohesion.
* Communicational Cohesion: Communicational cohesion involves grouping methods that operate on the same data. While PointInfo deals with data related to a point, it also includes unrelated information like color and chart consolidation, making it not purely communicational.

1. UserDao.java (com/serotonin/mango/db/dao/UserDao.java)

The OCavg value for the given class is 1.19 which is very low.

The provided Java class UserDao exhibits low cohesion due to the presence of multiple responsibilities and functionalities within the same class. Cohesion refers to how closely related and focused the responsibilities of a class are. Low cohesion occurs when a class handles unrelated tasks or performs multiple functionalities that are not closely related. Let's analyze the cohesion type observed in this class and contrast it with other cohesion types to understand why it is not exhibiting them.

It exhibits functional cohesion.

Contrast with Other Cohesion Types:

* Coincidental Cohesion: This class does not exhibit coincidental cohesion because the methods and functionalities present are not randomly or coincidentally grouped. They are related to user management, albeit in a loosely coupled manner.
* Logical Cohesion: The class does not demonstrate logical cohesion as it does not solely focus on a specific logical aspect or operation. Instead, it covers a range of functionalities related to user management.
* Temporal Cohesion: Temporal cohesion is not observed because the methods in the class are not necessarily executed in a specific sequence or temporal order.

1. ShareUser.java (com/serotonin/mango/view/ShareUser.java)

The OCavg value for given class is 1.67 which is very less compared to the other classes.

This Java class ShareUser exhibits low cohesion because it combines several unrelated functionalities within the same class. Cohesion refers to the degree to which elements of a module (such as a class) are related and work together towards a common purpose. Low cohesion means that the class contains disparate or unrelated functionalities.

To contrast, let's analyze why this class is not exhibiting other types of cohesion:

* Coincidental Cohesion: Coincidental cohesion occurs when elements of a module are grouped together arbitrarily without any meaningful relationship. In this class, although there are disparate functionalities, they are related to the overall purpose of managing shared user access, making it more than coincidental cohesion.
* Logical Cohesion: Logical cohesion involves grouping elements based on their logical relationship and functionality. While the class does have some logical grouping (e.g., user-related fields and access control constants), it still combines unrelated functionalities, indicating low cohesion rather than logical cohesion.
* Temporal Cohesion: Temporal cohesion involves grouping elements that are used at the same time. This class does not exhibit temporal cohesion as the elements are not necessarily used together at the same time; for instance, database access and JSON serialization may not occur simultaneously.
* Procedural Cohesion: Procedural cohesion occurs when elements are grouped based on the order in which they are executed. The class does not exhibit procedural cohesion as it combines functionalities that may not follow a strict procedural order.