Suraj Eswaran
CS515 Software Maintenance and Evolution

Software Quality & Code Smells

Code Smells are the results of poor coding and design choices that causes issues at the time of last phase of software development. They are considered to be a flag for programmers to show them the code might be functioning inappropriate. Code smells are due to

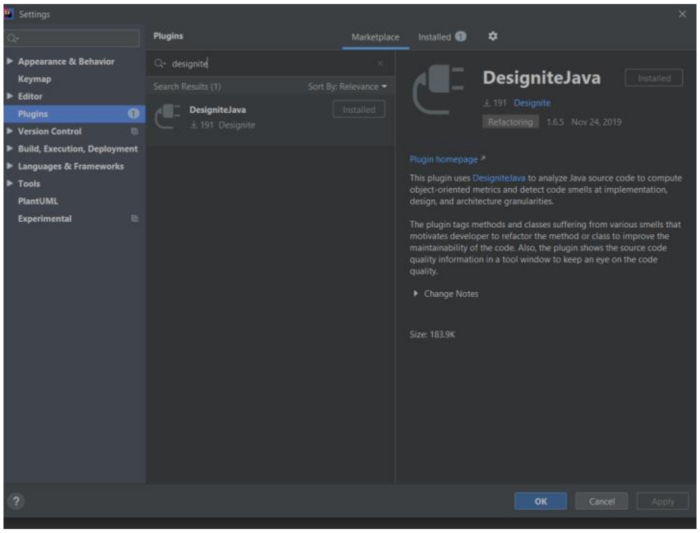
- *Duplicated code: If we see the same code structure in more than one place then it would make the code even more complex to function. Thus, it is necessary to find a way to unite them.*
- *Long method: Longer method would make the code to increase the time complexity, which would be difficult to digest.*
- *Large Class: Larger classes takes on too many responsibilities which would be difficult to focus on the main functionality.*
- *Lazy Class: A lazy class is a class that does not do anything enough to justify its existence.*
- *Lost Intent: If the code does not tend to communicate, thus the steps of the algorithm would blend together which makes the code little sense.*

The main focus of this report is to determine how the changes made in the code affect its quality.

PDFSAM

Here, I have used the tool DesigniteJava , which is free and open-source command-line tool for quality assessment of code written in Java. For now, DesigniteJava is available as a plugin in IntelliJ IDE.

1. DESIGNITEJAVA



This plugin uses DesigniteJava to analyze Java source code to compute object-oriented metrics and detect code smells at implementation, design, and architecture granularities.

The plugin tags methods and classes suffering from various smells that motivates developer to refactor the method or class to improve the maintainability of the code. Also, the plugin shows the source code quality information in a tool window to keep an eye on the code quality.

Figure 1*: Designite Plugin in Intellij*

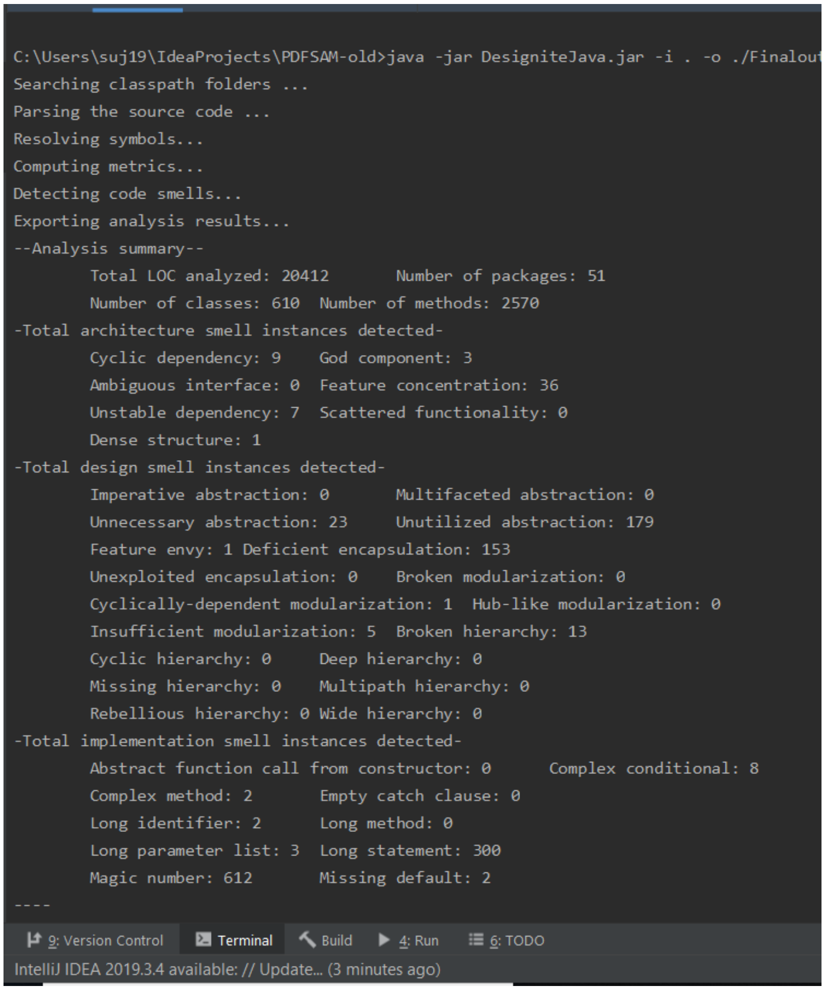
For running DesigniteJava, the following command used:

```
$ java -jar Designite.jar -i <path of the input source folder> -o <path of the output folder>
```

It is necessary to know that the output folder is empty before running the command.

Thus, there are five different kinds of code smells to show their code quality.

1. *Architecture Smells: Based on the architecture of the methods/classes*
2. *Design Smells: Based on the design of the methods/classes*
3. *Implementation Smells: Based on the Implementations done by the classes/methods*
4. *Method Metrics: Based on the metrics found out in methods*
5. *Type Metrics: Based on the metrics found out in classes*



```
C:\Users\suj19\IdeaProjects\PDFSAM-old>java -jar DesigniteJava.jar -i . -o ./Finalou
Searching classpath folders ...
Parsing the source code ...
Resolving symbols...
Computing metrics...
Detecting code smells...
Exporting analysis results...
--Analysis summary--
        Total LOC analyzed: 20412       Number of packages: 51
        Number of classes: 610  Number of methods: 2570
-Total architecture smell instances detected-
        Cyclic dependency: 9    God component: 3
        Ambiguous interface: 0  Feature concentration: 36
        Unstable dependency: 7  Scattered functionality: 0
        Dense structure: 1
-Total design smell instances detected-
        Imperative abstraction: 0       Multifaceted abstraction: 0
        Unnecessary abstraction: 23     Unutilized abstraction: 179
        Feature envy: 1 Deficient encapsulation: 153
        Unexploited encapsulation: 0    Broken modularization: 0
        Cyclically-dependent modularization: 1  Hub-like modularization: 0
        Insufficient modularization: 5  Broken hierarchy: 13
        Cyclic hierarchy: 0     Deep hierarchy: 0
        Missing hierarchy: 0    Multipath hierarchy: 0
        Rebellious hierarchy: 0 Wide hierarchy: 0
-Total implementation smell instances detected-
        Abstract function call from constructor: 0      Complex conditional: 8
        Complex method: 2       Empty catch clause: 0
        Long identifier: 2      Long method: 0
        Long parameter list: 3  Long statement: 300
        Magic number: 612       Missing default: 2
----
```

IntelliJ IDEA 2019.3.4 available: // Update... (3 minutes ago)

***Figure 2**:Output of DesigniteJava*

i. ARCHITECTURE SMELLS

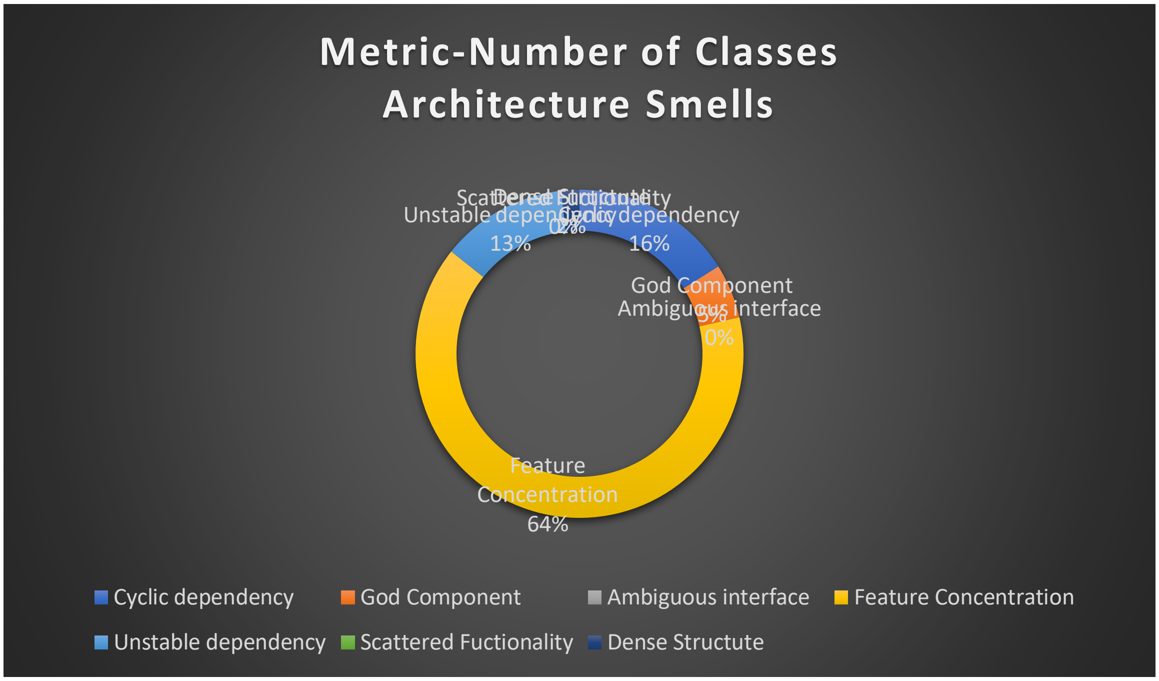


***Figure 3**:Architecture Smells*

CODE STATUS	***PACKAGE NAME***	***TYPE OF ARCHITECTURE SMELL***	***CAUSE OF THE SMELL***
Before Changes	*org.pdfsam.rotate*	*Feature Concentration*	*The tool detected the smell in this component because the component realizes more than one architectural concern/feature. Independent sets of related classes within this component are: [RotateModule]; [ModuleConfig]; [RotateOptionsPane]; [RotateParametersBuilder; RotateOptionsPaneTest; RotateSelectionPaneTest]; [RotateSelectionPane]; [RotateParametersBuilderTest]. LCC (Lack of Component Cohesion) = 0.75*
After Changes		*Feature Concentration*	*The tool detected the smell in this component because the component realizes more than one architectural concern/feature. Independent sets of*

			related classes within this component are: [RotateModule]; [ModuleConfig]; [RotateOptionsPane]; [RotateParametersBuilder; RotateOptionsPaneTest; RotateSelectionPaneTest]; [RotateSelectionPane]; [RotateParametersBuilderTest]. LCC (Lack of Component Cohesion) = 0.75

***Table 1**: Output for Architecture Smells*

ii. Design Smells

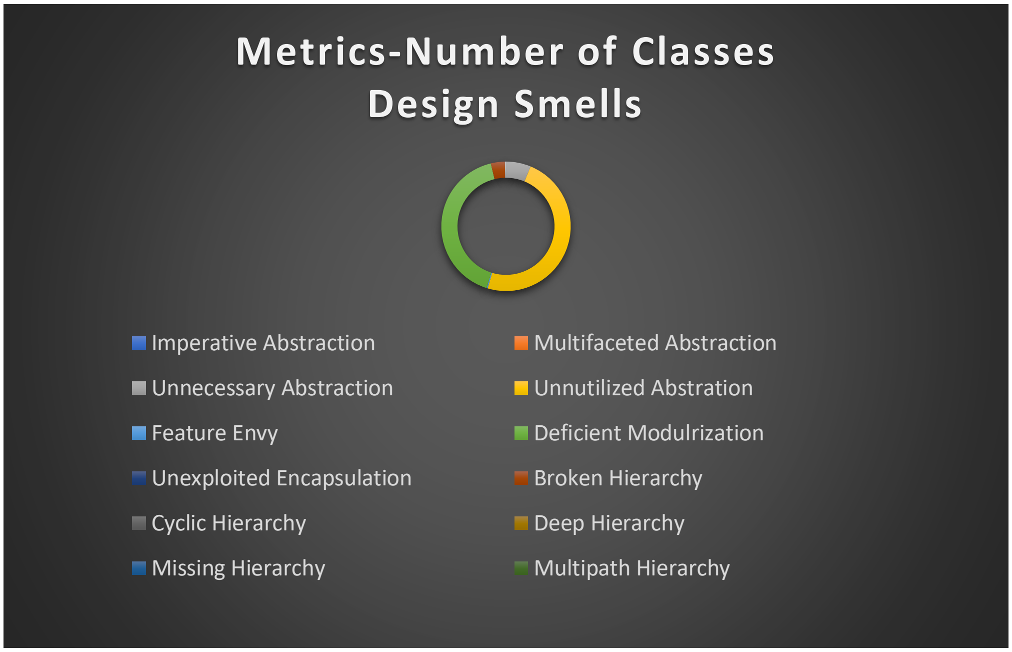


***Figure 4**: Design Smells*

PACKAGE NAME	***TYPE NAME***	***CODE STATUS***	***DESIGN SMELL***	***CAUSE OF THE SMELL***
org.pdfsam.rotate	*ModuleConfig*	*Before Changes*	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated*

				and/or used to serve a specific known purpose.)
		After Changes	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated and/or used to serve a specific known purpose.)*
	RotateOptionsPane	*Before Changes*	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated and/or used to serve a specific known purpose.)*
		After Changes	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated and/or used to serve a specific known purpose.)*
	RotateSelectionPane	*Before Changes*	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated and/or used to*

				serve a specific known purpose.)
		After Changes	*Unutilized Abstraction*	*The tool detected the smell in this class because this class is potentially unused. (Please ignore the smell if the reported class is auto-generated and/or used to serve a specific known purpose.)*
	RotateOptionsPaneTest	*Before Changes*	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public accessiblity: CLEAR_STUDIO; victim*
		After Changes	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public accessiblity: CLEAR_STUDIO; victim*
	RotateParametersBuilderTest	*Before Changes*	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public accessiblity: folder; victim*

		After Changes	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public accessiblity: folder; victim*
	RotateSelectionPaneTest	*Before Changes*	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public accessiblity: MODULE; clear; folder; javaFxThread; builder; onError; victim*
		After Changes	*Deficient Encapsulation*	*The tool detected the smell in this class because the class exposes fields belonging to it with public accessibility. Following fields are declared with public 7ccessibility: MODULE; clear; folder; javaFxThread; builder; onError; victim*

***Table 2**: Output for Design Smells*

iii. IMPLEMENTATION SMELLS

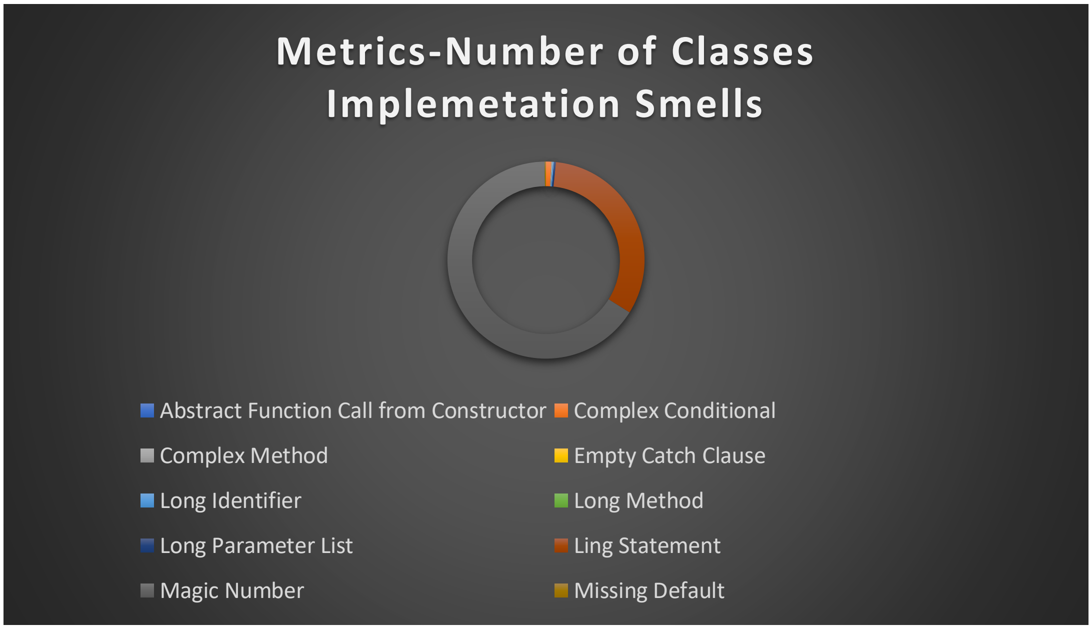


***Figure** 5: Implementation Smells*

PACKAGE NAME	*CODE STATUS*	*TYPE NAME*	*METHOD NAME*	*IMPLEMENTATION SMELL*	*CAUSE OF THE SMELL*
org.pdfsam.rotate	*After Change*	*RotateModule*	*settingPanel*	*Long Statement*	*The length of the statement "pane.getChildren().addAll(selectionPane`options`Views.titledPane(DefaultI18nContext.getInstance().i18n("Output settings")`destinationPane)`prefixTitled);" is 153.*
	Before Change		*settingPanel*	*Long Statement*	*The length of the statement "pane.getChildren().addAll(selectionPane`options`Views.titledPane(DefaultI18nContext.getInstance().i18n("Output settings")`destinationPane)`prefixTitled);" is 153.*
	After Change	*RotateOptionsPane*	*RotateOptionsPane*	*Long Statement*	*The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.ALL_PAGES`DefaultI18nContext.getInstance().i18n("All pages")));" is 126.*
	Before Change			*Long Statement*	*The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.ALL_PAGES`DefaultI18nContext.getInstance().i18n("All pages")));" is 126.*

	After Change	RotateOptionsPane	RotateOptionsPane	Long Statement	The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.EVEN_PAGES`DefaultI18nContext.getInstance().i18n("Even pages")));" is 128.
	Before Change				The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.EVEN_PAGES`DefaultI18nContext.getInstance().i18n("Even pages")));" is 128.
	After Change	RotateOptionsPane	RotateOptionsPane	Long Statement	The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.ODD_PAGES`DefaultI18nContext.getInstance().i18n("Odd pages")));" is 126.
	Before Change				The length of the statement "this.rotationType.getItems().add(keyValue(PredefinedSetOfPages.ODD_PAGES`DefaultI18nContext.getInstance().i18n("Odd pages")));" is 126.
	After Change	RotateOptionsPane	RotateOptionsPane	Long Statement	The length of the statement "this.rotation.getItems().add(keyValue(Rotation.DEGREES_90`DefaultI18nContext.getInstance().i18n("90

					degrees clockwise")));" is 122.
	Before Change				The length of the statement "this.rotation.getItems().add(keyValue(Rotation.DEGREES_90`DefaultI18nContext.getInstance().i18n("90 degrees clockwise")));" is 122.
	After Change	RotateOptionsPane	RotateOptionsPane	Long Statement	The length of the statement "this.rotation.getItems().add(keyValue(Rotation.DEGREES_180`DefaultI18nContext.getInstance().i18n("180 degrees clockwise")));" is 124.
	Before Change				The length of the statement "this.rotation.getItems().add(keyValue(Rotation.DEGREES_180`DefaultI18nContext.getInstance().i18n("180 degrees clockwise")));" is 124.
	After Change	RotateOptionsPane	RotateOptionsPane	Long Statement	The length of the statement "this.rotation.getItems().add(keyValue(Rotation.DEGREES_270`DefaultI18nContext.getInstance().i18n("90 degrees counterclockwise")));" is 130.
	Before Change				The length of the statement "this.rotation.getItems().ad

					d(keyValue(Rotation.DEGREES_270`DefaultI18nContext.getInstance().i18n("90 degrees counterclockwise")));" is 130.
	After Change	RotateOptionsPane	saveStateTo	Long Statement	The length of the statement "data.put("rotation"`Optional.ofNullable(rotation.getSelectionModel().getSelectedItem()).map(i -> i.getKey().toString()).orElse(EMPTY));" is 135.
	Before Change				The length of the statement "data.put("rotation"`Optional.ofNullable(rotation.getSelectionModel().getSelectedItem()).map(i -> i.getKey().toString()).orElse(EMPTY));" is 135.
	After Change	RotateOptionsPane	saveStateTo	Long Statement	The length of the statement "data.put("rotationType"`Optional.ofNullable(rotationType.getSelectionModel().getSelectedItem()).map(i -> i.getKey().toString()).orElse(EMPTY));" is 143.
	Before Change				The length of the statement "data.put("rotationType"`Optional.ofNullable(rotationType.getSelectionModel().getSelectedItem()).map(i -

					> i.getKey().toString()).orElse(EMPTY));" is 143.
	After Change	RotateOptionsPane	restoreStateFrom	Long Statement	The length of the statement "Optional.ofNullable(data.get("rotation")).map(Rotation::valueOf).map(r -> keyEmptyValue(r)).ifPresent(r -> this.rotation.getSelectionModel().select(r));" is 152.
	Before Change				The length of the statement "Optional.ofNullable(data.get("rotation")).map(Rotation::valueOf).map(r -> keyEmptyValue(r)).ifPresent(r -> this.rotation.getSelectionModel().select(r));" is 152.
	After Change	RotateOptionsPane	restoreStateFrom	Long Statement	The length of the statement "Optional.ofNullable(data.get("rotationType")).map(PredefinedSetOfPages::valueOf).map(r -> keyEmptyValue(r)).ifPresent(r -> this.rotationType.getSelectionModel().select(r));" is 172.
	Before Change				The length of the statement "Optional.ofNullable(data.get("rotationType")).map(PredefinedSetOfPages::val

					ueOf).map(r -> keyEmptyValue(r)).ifPresent(r -> this.rotationType.getSelectionModel().select(r));" is 172.
	After Change	RotateParametersBuilder*	addInput	Complex Method	Cyclomatic complexity of the method is 10
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2
	Before Change		-	-	-
	After Change	RotateParametersBuilder*	addInput	Magic Number	The method contains a magic number: 2

	Before Change		-	-	-
	After Change	***RotateParametersBuilder****	***addInput***	***Magic Number***	***The method contains a magic number: 2***
	Before Change		-	-	-
	After Change	***RotateParametersBuilder****	***addInput***	***Magic Number***	***The method contains a magic number: 2***
	Before Change		-	-	-
	After Change	*RotateSelectionPane*	*Apply*	*Long Statement*	*The length of the statement "super(ownerModule`false`false`new SelectionTableColumn<?>[]{new LoadingColumn(ownerModule)`FileColumn.NAME`LongColumn.SIZE`IntColumn.PAGES`LongColumn.LAST_MODIFIED`new PageRangesColumn(DefaultI18nContext.getInstance().i18n("Double click to set pages you want to rotate (ex: 2 or 5-23 or 2`5-7`12-)"))});" is 303.*
	Before Change				*The length of the statement "super(ownerModule`false`false`new SelectionTableColumn<?>[]{new LoadingColumn(ownerModule)`FileColumn.NAME`L*

					ongColumn.SIZE`IntColumn.PAGES`LongColumn.LAST_MODIFIED`new PageRangesColumn(DefaultI18nContext.getInstance().i18n("Double click to set pages you want to rotate (ex: 2 or 5-23 or 2`5-7`12-)"))});" is 303.
	After Change	*RotateSelectionPane*	*RotateSelectionPane*	*Long Statement*	*The length of the statement "table().getItems().stream().filter(s -> !Objects.equals("0"`trim(s.pageSelection.get()))).forEach(i -> builder.addInput(i.descriptor().toPdfFileSource()`i.toPageRangeSet()));" is 174.*
	Before Change				*The length of the statement "table().getItems().stream().filter(s -> !Objects.equals("0"`trim(s.pageSelection.get()))).forEach(i -> builder.addInput(i.descriptor().toPdfFileSource()`i.toPageRangeSet()));" is 174.*
	After Change	*RotateOptionsPaneTest*	*restoreStateFrom*	*Magic Number*	*The method contains a magic number: 2000*
	Before Change				*The method contains a magic number: 2000*
	After Change	*RotateOptionsPaneTest*	*Reset*	*Magic Number*	*The method contains a magic number: 2000*

	Before Change				*The method contains a magic number: 2000*
	After Change	*RotateOptionsPaneTest*	*Reset*	*Magic Number*	*The method contains a magic number: 2000*
	Before Change				*The method contains a magic number: 2000*
	After Change	*RotateParametersBuilderTest*	*buildDefaultSelection*	*Magic Number*	*The method contains a magic number: 3*
	Before Change				*The method contains a magic number: 3*
	After Change	*RotateParametersBuilderTest*	*buildDefaultSelection*	*Magic Number*	*The method contains a magic number: 5*
	Before Change				*The method contains a magic number: 5*
	After Change	*RotateParametersBuilderTest*	*buildDefaultSelection*	*Magic Number*	*The method contains a magic number: 2*
	Before Change				*The method contains a magic number: 2*
	After Change	*RotateParametersBuilderTest*	*buildRanges*	*Magic Number*	*The method contains a magic number: 5*
	Before Change				*The method contains a magic number: 5*
	After Change	*RotateParametersBuilderTest*	*buildRanges*	*Magic Number*	*The method contains a magic number: 4*
	Before Change				*The method contains a magic number: 4*
	After Change	*RotateParametersBuilderTest*	*buildRanges*	*Magic Number*	*The method contains a magic number: 5*
	Before Change				*The method contains a magic number: 5*
	After Change	*RotateParametersBuilderTest*	*buildMultiple*	*Magic Number*	*The method contains a magic number: 2*

	Before Change				*The method contains a magic number: 2*
	After Change	*RotateParametersBuilderTest*	*buildMultiple*	*Magic Number*	*The method contains a magic number: 5*
	Before Change				*The method contains a magic number: 5*
	After Change	*RotateParametersBuilderTest*	*buildMultiple*	*Magic Number*	*The method contains a magic number: 2*
	Before Change				*The method contains a magic number: 2*
	After Change	*RotateSelectionPaneTest*	*notEmptyPageSelection*	*Magic Number*	*The method contains a magic number: 2*
	Before Change				*The method contains a magic number: 2*

Table 3: *Output for Implementation Smells*

*Note: There seems to a change in the code in RotateParametricBuilders before and after , results in the cyclomatic complexity values in it.

iv. Method Metrics

PACKAGE NAME	*TYPE NAME*	*CODE STATUS*	*METHOD METRIC*	*LOC*	*CC*	*PC*
org.pdfsam.rotate	***RotateParametersBuilder***	***Before Changes***	***Addinput***	***8***	***2***	***2***
		After changes		***43***	***10***	***2***
	RotateParametersBuilder	*Before Changes*	*Hasinput*	*3*	*1*	*0*
		After Changes		*3*	*1*	*0*
	RotateParametersBuilder	*Before Changes*	*Output*	*3*	*1*	*1*
		After Changes		*3*	*1*	*1*
	RotateParametersBuilder	*Before Changes*	*Prefix*	*3*	*1*	*1*
		After Changes		*3*	*1*	*1*
	RotateParametersBuilder	*Before Changes*	*getOutput*	*3*	*1*	*0*
		After Changes		*3*	*1*	*0*
	RotateParametersBuilder	*Before Changes*	*getPrefix*	*3*	*1*	*0*
		After Changes		*3*	*1*	*0*
	RotateParametersBuilder	*Before Changes*	*Rotation*	*3*	*1*	*1*
		After Changes		*3*	*1*	*1*
	RotateParametersBuilder	*Before Changes*	*rotationType*	*3*	*1*	*1*
		After Changes		*3*	*1*	*1*
	RotateParametersBuilder	*Before Changes*	*Build*	*10*	*1*	*0*
		After Changes		*10*	*1*	*0*
	RotateOptionsPaneTest	*Before Changes*	*Start*	*6*	*1*	*1*
		After Changes		*6*	*1*	*1*
	RotateOptionsPaneTest	*Before Changes*	*validSteps*	*8*	*1*	*0*
		After Changes		*8*	*1*	*0*
	RotateOptionsPaneTest	*Before Changes*	*onSaveWorkspace*	*6*	*1*	*0*
		After Changes		*6*	*1*	*0*
	RotateOptionsPaneTest	*Before Changes*	*restoreStateFrom*	*10*	*1*	*0*
		After Changes		*10*	*1*	*0*

	RotateOptionsPaneTest	*Before Changes*	*Reset*	*13*	*1*	*0*
		After Changes		*13*	*1*	*0*
	RotateParametersBuilderTest	*Before Changes*	*Setup*	*8*	*1*	*0*
		After Changes		*8*	*1*	*0*
	RotateParametersBuilderTest	*Before Changes*	*buildDefaultSelection*	*20*	*1*	*0*
		After Changes		*20*	*1*	*0*
	RotateParametersBuilderTest	*Before Changes*	*buildRanges*	*14*	*1*	*0*
		After Changes		*14*	*1*	*0*
	RotateParametersBuilderTest	*Before Changes*	*buildMultiple*	*11*	*1*	*0*
		After Changes		*11*	*1*	*0*
	RotateSelectionPaneTest	*Before Changes*	*Setup*	*5*	*1*	*0*
		After Changes		*5*	*1*	*0*
	RotateSelectionPaneTest	*Before Changes*	*Empty*	*5*	*1*	*0*
		After Changes		*5*	*1*	*0*
	RotateSelectionPaneTest	*Before Changes*	*emptyPage*	*9*	*1*	*0*
		After Changes		*9*	*1*	*0*
	RotateSelectionPaneTest	*Before Changes*	*Not EmptyPage*	*10*	*1*	*0*
		After Changes		*10*	*1*	*0*
	RotateSelectionPaneTest*	***Before Changes***	***Conversion***	***7***	***1***	***0***
		After Changes		***8***	***1***	***0***
	RotateSelectionPaneTest	*Before Changes*	*emptyByZero*	*7*	*1*	*0*
		After Changes		*7*	*1*	*0*
	RotateSelectionPaneTest	*Before Changes*	*Populate*	*6*	*1*	*0*
		After Changes		*6*	*1*	*0*

Table 4*: Output for Method Metrics*

*Note: There seems to a change in the code in RotateParametricBuilders results in the cyclomatic complexity values in it.

v. TYPE METRICS

PACKAGE NAME	TYPE NAME	CODE STATUS	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
org.pdfsam.rotate	RotateModule	After Changes	7	0	14	12	87	15	0	0	0.214286	1	2
		Before Changes	7	0	14	12	87	15	0	0	0.214286	1	2
	ModuleConfig	After Changes	0	0	5	5	17	5	0	0	0	0	0
		Before Changes	0	0	5	5	17	5	0	0	0	0	0
	RotateOptionsPane	After Changes	2	0	5	4	39	5	0	0	1	0	1
		Before Changes	2	0	5	4	39	5	0	0	1	0	1
	RotateParametersBuilder	**After Changes**	**5**	**0**	**9**	**5**	**86**	**18**	**0**	**0**	**0.33333**	**2**	**0**
		Before Changes	**5**	**0**	**9**	**5**	**50**	**10**	**0**	**0**	**0.33333**	**2**	**0**
	RotateSelectionPane	After Changes	1	0	2	2	22	3	0	0	1	0	1
		Before Changes	1	0	2	2	22	3	0	0	1	0	1
	RotateOptionsPaneTest	After Changes	2	1	5	5	50	5	0	0	0	0	1
		Before Changes	2	1	5	5	50	5	0	0	0	0	1
	RotateParametersBuilderTest	After Changes	2	1	4	4	60	4	0	0	0	0	0
		Before Changes	2	1	4	4	60	4	0	0	0	0	0

	RotateSelectionPaneTest	*After Changes*	*7*	*3*	*7*	*6*	*61*	*7*	*0*	*0*	*0*	*0*	*2*
		Before Changes	*7*	*3*	*7*	*6*	*61*	*7*	*0*	*0*	*0*	*0*	*2*

***Table 5**: Output for Type Metrics*

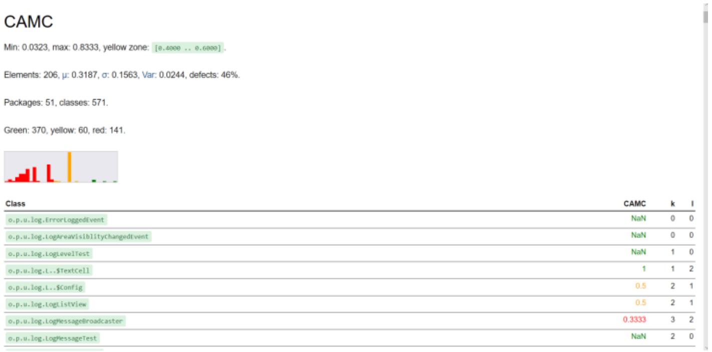
*Note: There seems to a change in the code in RotateParametricBuilders results in the cyclomatic complexity values in it.

2. JPEEK

For running JPeek, the following command must be used:

```
$ java -jar jpeek-0.30.9-jar-with-dependencies.jar --target ./jpeek --sources .
```

JPeek will analyze Java files in the current directory. XML reports will be generated in the./jpeek directory.



CAMC

Min: 0.0323, max: 0.8333, yellow zone: [0.4000 .. 0.6000].

Elements: 206, μ: 0.3187, σ: 0.1563, Var: 0.0244, defects: 46%.

Packages: 51, classes: 571.

Green: 370, yellow: 60, red: 141.

Class	CAMC	k	l
o.p.u.log.ErrorLoggedEvent	NaN	0	0
o.p.u.log.LogAreaVisiblityChangedEvent	NaN	0	0
o.p.u.log.LogLevelTest	NaN	1	0
o.p.u.log.L..$TextCell	1	1	2
o.p.u.log.L..$Config	0.5	2	1
o.p.u.log.LogListView	0.5	2	1
o.p.u.log.LogMessageBroadcaster	0.3333	3	2
o.p.u.log.LogMessageTest	NaN	2	0

***Figure 6**: Resultant CAMC XML file with the help of JPeek*

The top five metrics with the help of DesigniteJava and JPeek tools with highest changes from one version to another:

METRICS	***TYPE HAVING THE HIGHEST CHANGES***	***BEFORE CHANGES***	***AFTER CHANGES***
WMC	*RotateParametersBuilder*	*18*	*10*
NOM	*RotateModule*	*14*	*14*
LOC	*RotateParametersBuilder*	*50*	*86*
LCOM	*RotateParametersBuilder*	*0.3333*	*0.3333*
NOPM	*RotateModule*	*12*	*12*

***Table 6**: Result from the following tools*

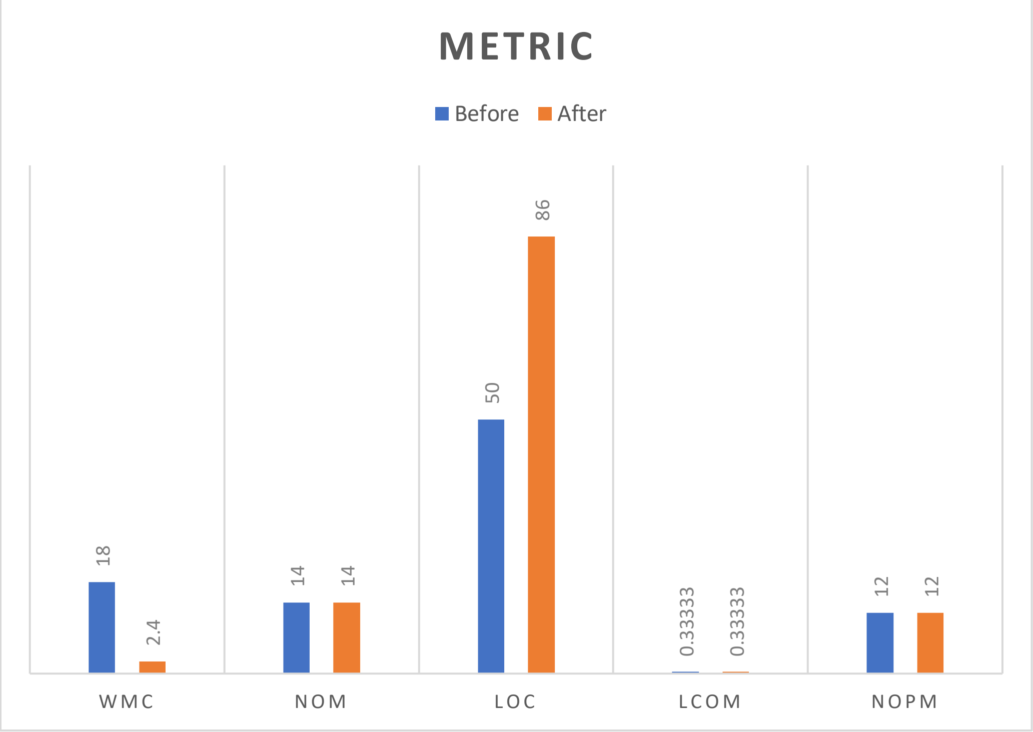


***Figure 6**: Graphical representation of the result*

Coupling can be defined as the measure of independence among modules of a program. Module should have low coupling. Lower the coupling, better the program.

There are different types of coupling:

- *Content Coupling occurs when one of the modules relies on the other module's internal working. It means a change in the second module will lead to the changes in the dependent module.*
- *Common Coupling happens when the same global data are shared by the two modules. In this, the modules will undergo changes if there are changes in the shared resource. It is also known as Global Coupling.*
- *In the case of External Coupling, when an external imposed data format and communication protocol are shared by two modules.*
- *Control Coupling occurs when one module controls the flow of another and passes information from one to another.*
- *Message Coupling can be obtained by the state decentralization, in which the component communication is performed through message passing.*
- *Data Coupling are coupling where modules are connected by the data coupling, if only data can be passed between them.*
- *Stamp Coupling is utilized where the data structure is used to transfer information from on component to another.*

Cohesion can be defined the degree of intra-dependability within elements of a module. Module should have low coupling. Higher the cohesion, better the program.

- *Functional Cohesion is a cohesion where parts of the module are grouped because they all contribute to the module's single well-defined task.*
- *Sequential Cohesion is considered when the parts of modules grouped due to the output from the one part is the input to the other.*
- *In Communication Cohesion, the parts of the module are grouped because they operate on the same data.*
- *For Procedural Cohesion, parts of the module are grouped because a certain sequence of execution is followed by them.*
- *When the module's parts are grouped because they are categorized logically to do the same work, even though they are all have different nature, it is known as Logical Cohesion.*

For Cohesion, the metric LCOM (Lack of Cohesion in Methods-Class) is considered. LCOM is a measure for number of not collected method pairs in a class showing independent parts which is of no cohesion.

Two classes with highest cohesion value are:

- *RotateParametricBuilder*
- *RotateModule*

Two classes with lowest cohesion value are:

- *RotateOptionPanel*
- *RotateSelectionPanel*

Reason for High Cohesion

- *As we know every object is created with a specific pre-defined task and in order to achieve its pre-defined tasks, a collective contribution of all these objects in the program are been considered. While an object is defined , it is necessary to take consideration of high cohesion characteristic, that is they must be closely related to the sub task of all the method. If it is defined closely related methods in a class, it is maintaining a high cohesion. But if we use some original related task with some unrelated methods in a class, it refers to low cohesion.*
- *Because of high cohesion, the reusability of class would be improved ,and also testing and maintenance will be easy whereas low cohesion results in low reusability, thus difficult in testing and maintenance.*
- *As per Table , the value of LCOM for RotateParametricBuilder and RotateModule are low as compared to RotateSelectionModule and RotateOptionPanel.*
  In the case of RotateParamerticBuilder class, it consists of several methods like addInput, hasInput, output, prefix , rotationType and Getprefix. The task of this class is to display rotated Pdf with the necessary options in it. Since they are independently functioning, so it must be having high cohesion.
  LCOM for a class will range between 0 and 1, with 1 being non-cohesive and 0 being totally cohesive. For each field in the class, we must count the number of methods that are referred in it , then divide that by the count of methods times the count of fields, and you subtract the result from one.
  For RotateParametricModule = 1-(7/15)= 0.333 LCOM
  In the case of RotateModule class, it consists of several methods like settingPanel, ModuleConfig, onLoadWorkspace, onSaveWorkspace, RotateModule and graphic. The task of this class is to display rotated Pdf from the following parameters given from the other sub classes. Same as RotateParametricModule, they are independently functioning, so it must be having high cohesion.
  For RotateModule= 1-(27/20)=0.214 LCOM
  Since the value of LCOM is near to zero than one , thus it is considered to have good cohesive. If the value of LCOM is less, then it is meant to be high cohesive. Thus, RotateParametricModule and RotateModule are those classes that has good cohesive.
- *Both RotateParametricModule and RotateModule tend to obtain Functional Cohesion since they all are grouped thus contributing module's single well-defined task.*

Reason for Low Cohesion

- *The class is said to be low cohesion if the functionalities of a module are independent of each other. Low Cohesion leads to less reusability of code, low readability and less testability.*
- *From the Table, RotateOptionPanel and RotateSelectionPanel have high LCOM value as compared to RotateParametricBuilder and RotateModule*
- *As mentioned above, LCOM for a class will range between 0 and 1, with 1 being non-cohesive and 0 being totally cohesive. For each field in the class, we must count the number of methods that are referred in it , then divide that by the count of methods times the count of fields, and you subtract the result from one.*

  In the case of class RotateSelectionPanel, it consists of several methods like apply , constructor RotateSelectionPanel. The task of this class is to select the listed rotated Pdf from the following parameters given from the other sub classes. But they are dependent on each other, so they have high LCOM and low cohesion. If the constructor was not there, then it would have resulted in high cohesion.

  For RotateSelectionPanel= 1-(0/2)=1 LCOM

  In the case of class RotaOptionsPane, it consists of several methods like apply, restView, saveStateTo ,restoreStateFrom constructor RotateOptionPanel. The task of this class is to show the options of rotated Pdf from the following parameters given from the other sub classes. But they are dependent on each other, so they have high LCOM and low cohesion. Same as RotateSelectionPane, if the constructor was not there, then it would have resulted in high cohesion.

  For RotateOptionPanel= 1-(0/2)=1 LCOM

  Since the value of LCOM is equal to one , thus it is considered to have low cohesive. If the value of LCOM is more, then it is meant to be less cohesive. Thus, RotateSelectionPane and RotateOptionPanel are those classes having low cohesion.
- *Since they have low cohesion, RotateSelectionPanel and RotateOptionPanel are considered to have Logical Cohesion since they are categorized logically to do the same work, even though they are all have different nature.*

High Cohesion(for selected classes)	***Low Cohesion(for selected classes)***
The values for LCOM are lesser that 1. Value nearer to zero, more will be the cohesion.	*The values for LCOM are equal to 1. Value nearer to one, less will be the cohesion.*
In the case of RotateParametricBuilder, even though the changes are made in it , the methods in it does not depend within itself.	*In the case of RotateSelectionPanel, the constructor tend to depend on it other method which results in less cohesion.*
Whereas in RotateModule, the sub classes tend to depend , which relatively shows that it would have higher cohesion	*Same as RotateSelectionPanel, RotateOptionPanel too have a constructor which relies on the functionality. Thus, it would result in less cohesion.*

Table 7*: Differences between High Cohesion and Low Cohesion from selected classes*

For Coupling, the metrics LCOM 2-3 (Lack of Cohesion in Method 2-3) and LCOM4 (Lack of Cohesion in Method 4) are considered. LCOM is a measure for number of not collected method pairs in a class showing independent parts which is of no cohesion.

Two classes with highest coupling value are:

- *RotateOptionPanel*
- *RotateSelectionPanel*

Two classes with lowest coupling value are:

- *RotateParametricBuilder*
- *RotateModule*

<u>Reason for High Coupling</u>

- *Coupling shows how closely two modules interact and how independent they are. The degree of coupling between two modules depends on their interface complexity*
- *If the system has low coupling , it is a sign of well-structured computer system and a great design. But if a low coupling with high cohesion is maintained, it supports the mission of high readability and maintainability.*
- *High Coupling happens when system having interconnection in which program unit depends upon each other. Loosely coupling are made up of components which are independent.*
- *As per Table , the value of LCOM for RotateSelectionModule and RotateOptionPanel are high as compared to RotateParametricBuilder and RotateModule.*
  In the case of class RotateSelectionModule, it consists of several methods like apply , constructor RotateSelectionPanel .Since data of one module is passed to another module, which is called data coupling. Hence it has high coupling.

  LCOM for a class will range between 0 and 1, with 1 having high coupling and 0 having low coupling.
  For RotateSelectionPanel= 1 LCOM
  In the case of apply RotateOptionPanel, there are several methods like restView, saveStateTo ,restoreStateFrom constructor RotateOptionPanel. Same as RotateSelectionPanel , they tend to depend on other modules like RotateModule, thus it is considered as high coupling.
  For RotateOptionPanel= 1 LCOM
  If the value of LCOM is high, then it is meant to be high coupling. Thus, RotateSelectionPanel and RotateOptionPanel are those classes that has high coupling.
- *Both RotateSelectionPanel and RotateOptionPanel tend to obtain Control Coupling since it occurs when one module controls the flow of another and passes information from one to another.*

<u>Reason for Low Coupling</u>

- *The class is said to be low cohesion if the functionalities of a module are independent of each other.*
- *From the Table, RotateParametricBuilder and RotateModule have low LCOM value as compared to RotateSelectionPanel and RotateOptionPane.*
- *As mentioned above, LCOM for a class will range between 0 and 1, with 1 being non-cohesive and 0 being totally cohesive. For each field in the class, we must count the number of methods that are referred in it , then divide that by the count of methods times the count of fields, and you subtract the result from one.*
- *In the case of class RotateParametricBuilder, it consists of several methods like addInput, hasInput, output, prefix , rotationType and Getprefix. But they are independent on each other, so they have low LCOM and low coupling.*

  For RotateParameterBuilder= 0.333 LCOM

  In the case of class RotateModule, it consists of several methods like settingPanel, ModuleConfig, onLoadWorkspace, onSaveWorkspace, RotateModule and graphic.

  But they are independent on each other, so they have low LCOM and low coupling.

  For RotateModule= 0.214 LCOM

  Since the value of LCOM is near to zero , thus it is considered to have low coupling. If the value of LCOM is less, then it is meant to be less coupling. Thus, RotateParametersBuilder and RotateModule are those classes having low coupling.
- *Since they have low coupling, RotateModule and RotateParameterBuilders are considered to have Stamp Coupling where the data structure is used to transfer information from on component to another.*

*High Coupling**(for selected classes)***	*Low Coupling**(for selected classes)***
The values for LCOM are equal to 1. Value nearer to one, more will be the coupling.	*The values for LCOM are near to 0. Value nearer to zero, less will be the coupling.*
In the case of RotateSelectionPanel, the constructor tend to depend on it other method which results in more of coupling.	*In the case of RotateParametricBuilder, even though the changes are made in it , the methods in it does not depend within itself.*
Same as RotateSelectionPanel, RotateOptionPanel too have a constructor which relies on the functionality. Thus, it would result in high coupling .	*Whereas in RotateModule, the sub classes tend to depend , which relatively shows that it would have lower coupling.*

***Table 8**: Differences between High Coupling and Low Coupling from selected classes*

JEDIT

For Jedit , I used JPeek and De as a tool for measuring the code metrics, which is a command-line tool for collecting metrics of code written in Java. This tool was built in such a way that it will make it possible to analyze code quality formally.

a. DesigniteJava

For running DesigniteJava, the following command must be used:

```
$ java -jar Designite.jar -i <path of the input source folder> -o <path of the output folder>
```

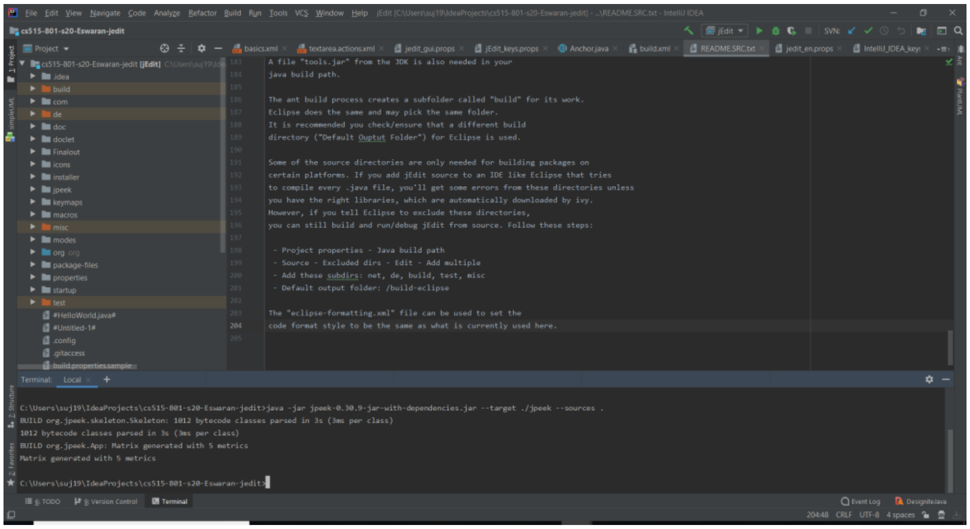
But running DesigniteJava, I found out a issue saying it can cover only up to 50,000 lines of codes , which is more in the case of Jedit project. For running that , it is necessary to buy Designite Java Professional Edition.

b. JPeek

For running JPeek, the following command must be used:

```
$ java -jar jpeek-0.30.9-jar-with-dependencies.jar --target ./jpeek --sources .
```

JPeek will analyze Java files in the current directory. XML reports will be generated in the./jpeek directory.



```
A file "tools.jar" from the JDK is also needed in your
java build path.

The ant build process creates a subfolder called "build" for its work.
Eclipse does the same and may pick the same folder.
It is recommended you check/ensure that a different build
directory ("Default Ouptut Folder") for Eclipse is used.

Some of the source directories are only needed for building packages on
certain platforms. If you add jEdit source to an IDE like Eclipse that tries
to compile every .java file, you'll get some errors from these directories unless
you have the right libraries, which are automatically downloaded by ivy.
However, if you tell Eclipse to exclude these directories,
you can still build and run/debug jEdit from source. Follow these steps:

 - Project properties - Java build path
 - Source - Excluded dirs - Edit - Add multiple
 - Add these subdirs: net, de, build, test, misc
 - Default output folder: /build-eclipse

The "eclipse-formatting.xml" file can be used to set the
code format style to be the same as what is currently used here.
```

```
C:\Users\suj19\IdeaProjects\cs515-801-s20-Eswaran-jedit>java -jar jpeek-0.30.9-jar-with-dependencies.jar --target ./jpeek --sources .
BUILD org.jpeek.skeleton.Skeleton: 1012 bytecode classes parsed in 3s (3ms per class)
1012 bytecode classes parsed in 3s (3ms per class)
BUILD org.jpeek.App: Matrix generated with 5 metrics
Matrix generated with 5 metrics

C:\Users\suj19\IdeaProjects\cs515-801-s20-Eswaran-jedit>
```

***Figure 7**: Output of Jpeek in Intellij*

List of cohesion metrics that were experimented in this tool are:

- *Cohesion Among Method Classes(CAMC)*
- *Lack of Cohesion in Methods(LCOM)*
- *Optimistic Class Cohesion(OCC) and Pessimistic Class Cohesion(PCC)*
- *Method-Method through Attributes Cohesion(MMAC)*
- *Normalized Hamming Distance(NHD)*
- *Lack of Cohesion in Method 2-3(LCOM 2-3)*
- *Class Connection Metric(CCM)*
- *A Sensitive Metric of Class Cohesion(SCOM)*
- *Tight Class Cohesion(TCC)*
- *Transitive Lack of Cohesion in Methods(TLCOM)*
- *Lack of Cohesion in Method 4(LCOM4)*

The top five metrics from Jpeek with highest changes from one version to another:

METRICS	***TYPE HAVING THE HIGHEST CHANGES***	***BEFORE CHANGES***	***AFTER CHANGES***
CAMC	*o.g.s.j.textarea.ScrolLineCount*	*0.875*	*0.89*
NHD	*o.g.s.j.textarea.Selection*	*0.7077*	*0.7077*
MMAC	*o.g.s.j.textarea.BufferHandler*	*0.6786*	*0.6786*
NHD	*o.g.s.j.textare.S..$Range*	*0.7*	*0.7*
LCOM5	*o.g.s.j.options.TextAreaOptionPane*	*0.5475*	*0.5124*

Table 9*: Tabular column of resultant top five metric*

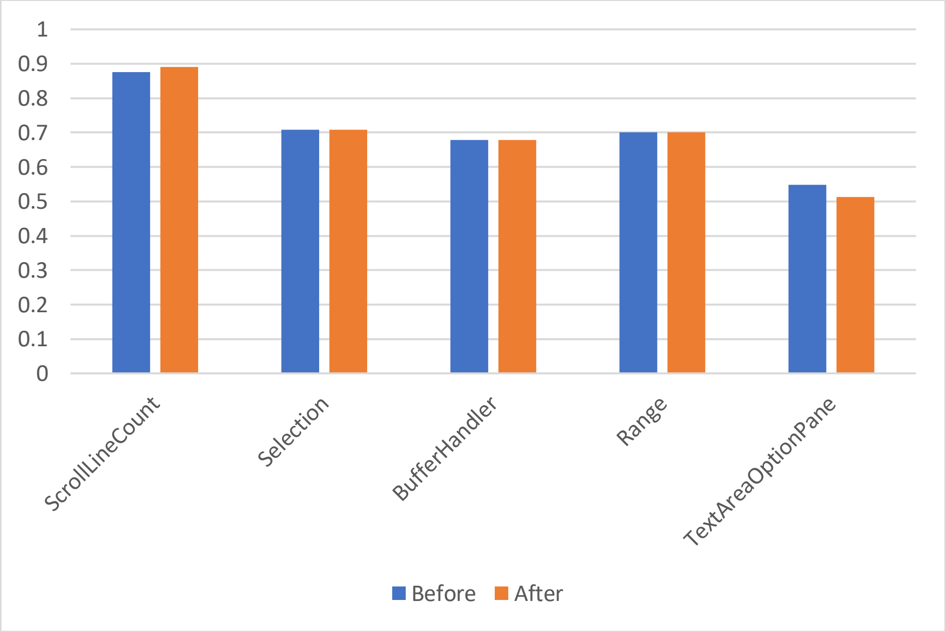


Figure 8*: Graphical Representation of the resultant Metrics*

Coupling can be defined as the measure of independence among modules of a program. Module should have low coupling. Lower the coupling, better the program. There are different types of coupling:

- *Content Coupling occurs when one of the modules relies on the other module's internal working. It means a change in the second module will lead to the changes in the dependent module.*
- *Common Coupling happens when the same global data are shared by the two modules. In this, the modules will undergo changes if there are changes in the shared resource. It is also known as Global Coupling.*
- *In the case of External Coupling, when an external imposed data format and communication protocol are shared by two modules.*
- *Control Coupling occurs when one module controls the flow of another and passes information from one to another.*
- *Message Coupling can be obtained by the state decentralization, in which the component communication is performed through message passing.*
- *Data Coupling are coupling where modules are connected by the data coupling, if only data can be passed between them.*
- *Stamp Coupling is utilized where the data structure is used to transfer information from on component to another.*

Cohesion can be defined the degree of intra-dependability within elements of a module. Module should have low coupling. Higher the cohesion, better the program. There are different types of cohesion:

- *Functional Cohesion is a cohesion where parts of the module are grouped because they all contribute to the module's single well-defined task.*
- *Sequential Cohesion is considered when the parts of modules grouped due to the output from the one part is the input to the other.*
- *In Communication Cohesion, the parts of the module are grouped because they operate on the same data.*
- *For Procedural Cohesion, parts of the module are grouped because a certain sequence of execution is followed by them.*
- *When the module's parts are grouped because they are categorized logically to do the same work, even though they are all have different nature, it is known as Logical Cohesion.*

For Cohesion, the metric LCOM (Lack of Cohesion in Methods-Class) and CACM(Cohesion Among Method Classes) are considered. LCOM is a measure for number of not collected method pairs in a class showing independent parts which is of no cohesion. CACM is a metric that measures the extend of intersection of individual method parameter type lists with the parametric type lists of all methods in the class. It computes the relatedness among methods of a class based upon parameter list of the methods.

Two classes with highest cohesion value based on LCOM and CACM are:

- *ScrollLineCount*
- *TextArea*

Two classes with lowest cohesion value are:

- *StandaloneTextArea*
- *StructureMatcher*

<u>*Reason for High Cohesion*</u>

- *As we know every object is created with a specific pre-defined task and in order to achieve its pre-defined tasks, a collective contribution of all these objects in the program are been considered. While an object is defined , it is necessary to take consideration of high cohesion characteristic, that is they must be closely related to the sub task of all the method. If it is defined closely related methods in a class, it is maintaining a high cohesion. But if we use some original related task with some unrelated methods in a class, it refers to low cohesion.*
- *Because of high cohesion, the reusability of class would be improved ,and also testing and maintenance will be easy whereas low cohesion results in low reusability, thus difficult in testing and maintenance.*
- *As per Table , the value of LCOM for ScrollLineCount and TextArea are low as compared to StandaloneTextArea and StructureMatcher. The value of CACM should be NAN(Null) for a class to be high cohesive ,thus ScrollLineCount and TextArea having good cohesive.*
- *In the case of class ScrollLineCount, it consists of several methods like reset, changed, preContentInserted , contentInserted and contentRemoved. The task of this class is to perform scroll action on jedit. Since they are independently functioning, so it must be having high cohesion.*
  LCOM for a class will range between 0 and 1, with 1 being non-cohesive and 0 being totally cohesive. For each field in the class, we must count the number of methods that are referred in it , then divide that by the count of methods times the count of fields, and you subtract the result from one.
  For ScrollTextArea = 1-(5/15)= 0.7542 LCOM
  In the case of class TextArea, it consists of several methods like JeditActionContext, initInputHandler , setMouseHandler , setTransferHandler, and toString . The task of this class is to display the text area from the following parameters given from the other sub classes. Same as ScrollTextArea , they are independently functioning, so it must be having high cohesion.
  For TextArea = 1-(7/12)= 0.7445 LCOM
  Thus, both the classes are having good cohesive.
- *The value of CACM should be NAN since there would be any intersection of individual method parameter type lists with the parametric type lists of all methods in the class.*

Intersection of parameter type lists would result in dependency; hence they would not be having high cohesion.

- *Both the classes tend to result Functional Cohesion since they all are grouped thus contributing module's single well-defined task.*

<u>*Reason for Low Cohesion*</u>

- *The class is said to be low cohesion if the functionalities of a module are independent of each other. Low Cohesion leads to less reusability of code, low readability and less testability.*
- *From the Table, StandaloneTextArea and StructureMatcher have high LCOM value as compared to ScrollTextArea and TextArea.*
- *As mentioned above, LCOM for a class will range between 0 and 1, with 1 being non-cohesive and 0 being totally cohesive. For each field in the class, we must count the number of methods that are referred in it , then divide that by the count of methods times the count of fields, and you subtract the result from one.*

  In the case of class StandaloneTextArea, it consists of several methods like initTextArea, initGutter , initPainter, getIntegerProperty, loadProperties and, StandaloneTextArea constructor . The task of this class is to select the listed rotated Pdf from the following parameters given from the other sub classes. But they are dependent on each other, so they have high LCOM and low cohesion. If the constructor was not there, then it would have resulted in high cohesion.

  For StandaloneTextArea = 1-(0/2)=1 LCOM

  In the case of class StructureMatcher , it consists of several methods like Highlight, getOffsets, paintHighlight and StructureMatcher constructor. The task of this class is to show the options of rotated Pdf from the following parameters given from the other sub classes. But they are dependent on each other, so they have high LCOM and low cohesion. Same as StandaloneTextArea, if the constructor was not there, then it would have resulted in high cohesion.

  For StructureMatcher = 1-(0/2)=1 LCOM

  Since the value of LCOM is equal to one , thus it is considered to have low cohesive. If the value of LCOM is more, then it is meant to be less cohesive. Thus, and are those classes having low cohesion.
- *The value of CACM should be NAN since there would be any intersection of individual method parameter type lists with the parametric type lists of all methods in the class. Intersection of parameter type lists would result in dependency; hence they would not be having high cohesion.*
- *But they are not NAN instead they have a value of 0.7143 and 0.75 respectively , so they have low cohesion.*
- *Since they have low cohesion, thus they are considered to have Logical Cohesion since they are categorized logically to do the same work, even though they are all have different nature.*

High Cohesion(for selected classes)	***Low Cohesion(for selected classes)***
The values for LCOM are lesser that 1. Value nearer to zero, more will be the cohesion.	*The values for LCOM are equal to 1. Value nearer to one, less will be the cohesion.*
In the case of ScrollTextArea, even though the changes are made in it , the methods in it does not depend within itself.	*In the case of StructuralMatcher, the constructor tend to depend on its other method which results in less cohesion.*
In TextArea, the sub classes tend to depend , which relatively shows that it would have higher cohesion	*But too have a constructor which relies on the functionality. Thus, it would result in less cohesion.*

***Table 10**: Differences between High Coupling and Low Coupling from selected classes*

Two classes with highest coupling value are:

- *StructureMatcher*
- *Selection*

Two classes with lowest coupling value are:

- *TextArea*
- *ScrollLineArea*

Reason for High Coupling

- *Coupling shows how closely two modules interact and how independent they are. The degree of coupling between two modules depends on their interface complexity*
- *If the system has low coupling , it is a sign of well-structured computer system and a great design. But if a low coupling with high cohesion is maintained, it supports the mission of high readability and maintainability.*
- *High Coupling happens when system having interconnection in which program unit depends upon each other. Loosely coupling are made up of components which are independent.*
- *As per Table ,the value of LCOM for StructureMatcher and Selection are high as compared to TextArea and ScrollLineArea.*
  In the case of class StructureMatcher, it consists of several methods like Highlight, getOffsets, paintHighlight and constructor StructureMatcher .Since data of one module is passed to another module, which is called data coupling. Hence it has high coupling.

  LCOM for a class will range between 0 and 1, with 1 having high coupling and 0 having low coupling.
  For StructuralMatcher= 1 LCOM
  Same as StructuralMatcher , they tend to depend on other modules(like TextArea) , thus it is considered as high coupling.
  For Selection= 1 LCOM

If the value of LCOM is high, then it is meant to be high coupling. Thus, StructuralMatcher and Selection are those classes that has high coupling.

- *Both StructuralMatchers and Selection tend to obtain Control Coupling since it occurs when one module controls the flow of another and passes information from one to another.*

Reason for Low Coupling

- *The class is said to be low cohesion if the functionalities of a module are independent of each other.*
- *From the Table, TextArea and ScrollLineCount have low LCOM value as compared to Selection and StructuralMatchers.*

- *In the case of class ScrollLineCount, it consists of several methods like reset, changed, preContentInserted , contentInserted and contentRemoved . But they are independent on each other, so they have low LCOM and low coupling.*
  For ScrollLineCount = 0.333 LCOM
  In the case of class TextArea, it consists of several methods like JeditActionContext, initInputHandler , setMouseHandler , setTransferHandler, and toString.
  But they are independent on each other, so they have low LCOM and low coupling.
  For TextArea= 0.214 LCOM
  Since the value of LCOM is near to zero , thus it is considered to have low coupling. If the value of LCOM is less, then it is meant to be less coupling. Thus, TextArea and ScrollLineCount are those classes having low coupling.
- *Since they have low coupling, TextArea and ScrollLineCount are considered to have Stamp Coupling where the data structure is used to transfer information from on component to another.*

High Coupling(for selected classes)	***Low Coupling(for selected classes)***
The values for LCOM are lesser that 1. Value nearer to zero, more will be the cohesion.	*The values for LCOM are equal to 1. Value nearer to one, less will be the cohesion.*
In Selection, the constructor tend to depend on it other method which results in more of coupling.	*In the case of TextAreal, the constructor tend to depend on its other method which results in less cohesion.*
Same as Selection, StructureMatcher too have a constructor which relies on the functionality. Thus, it would result in high coupling .	*Even ScrollLineCount too have a constructor which relies on the functionality. Thus, it would result in less cohesion.*

***Table 11**: Differences between High Coupling and Low Coupling from selected classes*

REFERENCES

1. Hitz, Martin, and Behzad Montazeri. *Measuring coupling and cohesion in object-oriented systems*. na, 1995.
2. Reijers, Hajo A., and Irene TP Vanderfeesten. "Cohesion and coupling metrics for workflow process design." *International Conference on Business Process management*. Springer, Berlin, Heidelberg, 2004.
3. Candela, Ivan, et al. "Using cohesion and coupling for software remodularization: Is it enough?." *ACM Transactions on Software Engineering and Methodology (TOSEM)* 25.3 (2016): 1-28.