ASSIGNMENT 3: SOFTWARE QUALITY & CODE SMELLS

Alalyani,Nada, Sanket Mehrotra

EVOLUTIONARY RAMS CS515

Contents

JEdit 2
1. The metrics analyzed 2
2 Analyzing code metrics 4
2.1 Metrics for the original project modules. 4
2.2 Metrics for the modified project modules. 10
2.3 Affected Metrics 15
3 Cohesion and Coupling 17
3.1 Cohesion 17
3.2 Coupling 19
PDFsam 21
1. The metrics analyzed 21
2. The projects analyzed 22
3. Analyzing code metrics 23
3.1. Metrics for the original project modules. 23
3.2. Metrics for the modified project modules. 27
3.3. Affected Metrics 31
4. Cohesion and Coupling 34
4.1. Cohesion 34
4.2. Coupling 36

JEdit

TOOL: DESIGNITEJAVA

1. The metrics analyzed

1. **Method level metrics**
    a. **Lines of Code (LOC)**: Computes the lines of **non-comment, non-blank** code in the function.
    b. **Cyclomatic complexity (CC)**: At the method level, computes the number of possible paths to measure the complexity of the function.
    c. **Parameter Count (PC)**: The number of arguments given to a function.
2. **Class level metrics**
    a. **Number of fields (NOF)**: calculates the number of data members of a class.
    b. **Number of public fields (NOPF)):** calculates the number of public data-members in a class.
    c. **Number of methods (NOM):** calculates the total number of methods in a class.
    d. **Number of public methods (NOPM)**: calculates the number of public methods in a class.
    e. **Lines of code (LOC)**: Computes the number of lines of code in line with the conditions highlighted above (1.a) but for the whole class.
    f. **Weighted methods per class (WMC)**: The Cyclomatic Complexity values of each method in the class are summed to find the complexity of a class, this is the WMC. It can be used to determine whether a class needs to be restructured into smaller classes. WMC should be low, since it will keep classes restricted to a single responsibility and prevent a greater potential impact on derived classes.
    g. **Number of children (NOC)**: NOC equals the number of immediate child classes derived from a base class.
    h. **Depth of inheritance tree (DIT)**: Finds the position of the class in the Inheritance tree. This is used to measure the complexity of the class structure
    i. **Lack of cohesion in methods (LCOM)**: According to Chidamber & Kemerer, to calculate LCOM, two methods of a class are checked; if they accessed different variables, P is increased by one. However, if they access the same variable, Q is increased by one. The lower LCOM, the more cohesive class. It can also be used to determine whether a class needs to be restructured into smaller classes, like WMC.

LCOM1 = P - Q , if P > Q

LCOM1 = 0 otherwise

LCOM1 = 0 indicates a cohesive class.

LCOM1 > 0 indicates that the class needs or can be split into two or more classes, since its variables belong in disjoint sets

---

j. **Fan-in(FANIN)**: For a given class X, the fan-in structural metric can be used to calculate how much code is being reused. It can be defined as the number of classes that depend on class X. Where 'dependence' could be using X's methods, its variables, constants or use its sub-classes or enums.
k. **Fan-out(FANOUT)**: Fan-out denotes how many classes a given class Y depends on. Here, Y's dependence includes using other classes' methods, variables, enums or constants. Fan out can be used to calculate the coupling of classes; high FO denotes strongly coupled code. The class Y depends on other code and is probably more complex to execute and test.

2 Analyzing code metrics

2.1 Metrics for the original project modules.

Since the project has thousands of classes within a large number of packages, we selected only some classes from some of the packages that involve the modifications to be shown in the report after analyzing the whole project. To see the whole results, you can see the attached CSV files.

- **Class level metrics**
  - **This package is org.gjt.sp.jedit.search**

Table 1: Class level metrics calculated by DesigniteJava for the JEdit project

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FAN-OUT
SearchMatcher	6	3	7	4	83	13	2	0	0.2857	7	2
Match	3	3	1	1	8	1	0	0	0	7	0
SearchAndReplace	12	0	44	33	805	151	0	0	0.0454	14	26
HyperSearchResults	26	6	60	37	637	132	0	1	0.2166	5	28
ActionHandler	0	0	1	1	27	8	0	0	-1	1	5
HighlightingTree	2	0	2	1	54	9	0	0	0	0	4
KeyHandler	0	0	2	2	26	5	0	0	-1	0	0
MouseHandler	1	0	2	1	44	10	0	0	0	0	3
RemoveTreeNodeAction	0	0	2	1	8	2	0	0	-1	0	1
RemoveAllTreeNodesAction	0	0	2	1	8	2	0	0	-1	0	1
NewSearchAction	0	0	2	1	15	3	0	0	-1	0	5
ExpandChildTreeNodesAction	0	0	2	1	10	2	0	0	-1	0	1
CopyToClipboardAction	0	0	2	1	14	2	0	0	-1	0	3
ToStringNodes	1	0	1	1	14	3	0	1	0	3	1
CollapseChildTreeNodesAction	0	0	2	1	14	3	0	0	-1	0	1
RedoSearchAction	1	0	2	1	16	2	0	0	0	0	3
TreeDisplayAction	0	0	1	1	30	4	0	0	-1	0	1
GoToNodeAction	1	0	2	1	10	2	0	0	0	0	1
ResultCellRenderer	4	0	4	1	45	9	0	1	0.5	0	4
CountNodes	2	0	1	1	12	2	0	1	0	2	0
ResultTreeTransferHandler	0	0	1	1	13	2	0	0	-1	0	1
ResultVisitor	0	0	1	0	3	1	2	0	-1	1	2
BufferLoadedVisitor	0	0	1	1	5	1	0	1	-1	0	2
BufferClosedVisitor	0	0	1	1	5	1	0	1	-1	0	2
TreeNodeCallbackAdapter	0	0	1	1	5	1	0	1	-1	0	0
BoyerMooreSearchMatcher	7	0	8	5	150	31	0	1	0	1	3
HyperSearchFileNode	4	2	7	7	51	10	0	1	0.2857	5	6
PatternSearchMatcher	3	0	7	6	170	36	0	1	0	2	3
SearchFileSet	0	0	7	0	43	7	2	0	-1	6	1
HyperSearchRequest	7	0	8	4	172	27	0	2	0	0	16
SearchBar	16	0	16	12	262	51	0	0	0.125	1	12
ActionHandler	0	0	1	1	23	7	0	0	-1	0	2
DocumentHandler	0	0	3	3	32	9	0	0	-1	0	2
KeyHandler	0	0	1	1	19	5	0	0	-1	0	0
FocusHandler	0	0	1	1	5	1	0	0	-1	0	0

– **Method Level**
  - **HyperSearchResults.java**

Table 2: Method level metrics calculated by DesigniteJava for HyperSearchResults

Class Name	Method Name	LOC	CC	PC
HyperSearchResults	removeSelectedNode	23	5	0
	removeAllNodes	6	1	0
	hideDockable	3	1	0
	trimSearchString	8	2	0
	parseHighlightStyle	4	1	1
	actionPerformed	25	8	1
	HighlightingTree	8	2	1
	convertValueToText	42	7	6
	keyPressed	24	4	1
	run	3	1	0
	mousePressed	10	4	1
	showPopupMenu	31	6	1
	RemoveTreeNodeAction	3	1	0
	actionPerformed	3	1	1
	RemoveAllTreeNodesAction	3	1	0
	actionPerformed	3	1	1
	NewSearchAction	3	1	0

- **Class level metrics**
  - **This package is org.gjt.sp.jedit.gui**

Table 3: Class level metrics calculated by DesigniteJava for the Jedit project

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
FocusHandler	0	0	2	2	8	2	0	0	-1	0	0
EscapeHandler	0	0	1	1	6	1	0	0	-1	0	0
TabHandler	0	0	1	1	5	1	0	0	-1	0	0
InsertHandler	0	0	1	1	5	1	0	0	-1	0	0
ClearHandler	0	0	1	1	5	1	0	0	-1	0	0
Entry	3	0	4	4	22	4	0	0	0	0	0
DummyRenderer	0	0	1	1	5	1	0	0	-1	0	0
LabelRenderer	2	0	2	1	14	3	0	0	0	0	1
CheckBoxListModel	1	0	10	7	62	19	0	0	0.4	1	3
AbstractContextOptionPane	12	0	14	4	210	30	2	2	0	3	9
MenuItemCompare	0	0	1	1	5	1	0	0	-1	0	2
MenuItem	2	0	2	1	11	2	0	0	0	4	1
ActionHandler	0	0	1	1	61	11	0	0	-1	0	6
ListHandler	0	0	1	1	5	1	0	0	-1	0	0
MutableListModel	0	0	2	2	7	2	2	0	-1	1	0
DefaultInputHandler	0	0	8	8	139	28	0	2	-1	1	5
MouseHandler	1	0	1	1	10	2	0	0	0	0	1
ContainerHandler	0	0	4	2	26	8	0	0	-1	0	0
KeyHandler	0	0	2	1	49	18	0	0	-1	3	2
WindowHandler	0	0	1	1	5	1	0	0	-1	0	0
StatusBar	25	0	22	20	306	59	0	0	0.272	4	14
MouseHandler	0	0	1	1	8	2	0	0	-1	0	0
AbbrevEditor	3	0	8	8	146	24	0	0	0	2	1
RolloverButton	2	0	9	9	70	11	6	0	0.444	8	0
MouseOverHandler	0	0	2	2	13	2	0	0	-1	0	0
InputHandler	1	0	14	9	264	68	1	1	0.357	3	18
AnimatedIcon	6	0	10	10	68	12	0	0	0	0	0
Animator	0	0	1	1	7	1	0	0	-1	0	0
FontSelectorDialog	12	0	14	8	209	30	0	1	0	1	4
ActionHandler	0	0	1	1	6	3	0	0	-1	0	0
ListHandler	0	0	1	1	18	7	0	0	-1	0	0
BeanShellErrorDialog	0	0	2	2	13	2	0	2	-1	0	1
DockableWindowFactory	22	0	36	24	301	60	0	0	0.138	9	11

- **Method Level**
  - **Statusbar.java**

Table 4: Method level metrics calculated by DesigniteJava for Jedit Project

Class Name	Method Name	LOC	CC	PC
StatusBar	StatusBar	28	1	1
	propertiesChanged	54	6	0
	addNotify	4	1	0
	removeNotify	4	1	0
	run	17	4	0
	waiting	3	1	1
	running	2	1	1
	done	3	1	1
	statusUpdated	2	1	1
	maximumUpdated	2	1	1
	valueUpdated	2	1	1
	getMessage	3	1	0
	setMessageAndClear	12	2	1
	actionPerformed	3	2	1
	setMessage	12	4	1
	setMessageComponent	8	2	1
	updateCaretStatus	54	12	0
	updateBufferStatus	9	1	0
	updateMiscStatus	5	1	0
	getWidget	13	11	1
	_getWidget	7	2	1

- **Class Level**
  - **This package is org.gjt.sp.jedit.options**

Table 5: Class level metrics calculated by DesigniteJava for the Jedit project

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
MouseHandler	0	0	1	1	11	3	0	0	-1	0	1
BrowserColorsModel	3	0	16	10	115	27	0	0	0.4375	1	3
Entry	2	0	1	0	8	1	0	0	0	3	0
ColorRenderer	0	0	2	1	19	4	0	0	-1	0	1
ModeSettingsPane	40	0	12	2	287	24	0	2	0.25	1	6
ActionHandler	0	0	1	1	13	3	0	0	-1	0	0
ModeProperties	19	0	4	0	115	11	0	0	0.5	1	2
ShortcutsOptionPane	17	1	42	28	450	95	0	2	0.21428571	0	14
HeaderMouseHandler	0	0	1	1	17	4	0	0	-1	0	1
TableMouseHandler	0	0	1	1	10	3	0	0	-1	0	2
ActionHandler	0	0	1	1	63	14	0	0	-1	0	4
ShortcutsModel	4	1	14	12	100	30	0	0	0.142857143	3	4
KeyCompare	1	0	2	1	26	7	0	0	0	0	1
KeymapsModel	2	0	7	4	33	11	0	0	0	2	2
KeymapCellRenderer	0	0	1	1	8	1	0	0	-1	0	1
GlobalOptions	0	0	8	5	44	10	0	2	-1	0	5
PrintOptionPane	10	0	3	1	62	3	0	2	0.666666667	0	2
StatusBarOptionPane	28	0	14	8	359	41	0	2	0.21428571	0	9
ActionHandler	0	0	1	1	58	12	0	0	-1	0	1
ListHandler	0	0	1	1	5	1	0	0	-1	0	0
WidgetListCellRenderer	0	0	1	1	10	2	0	0	-1	0	1
WidgetSelectionDialog	9	0	6	4	129	16	0	1	0.33333333a	2	4
ActionHandler	0	0	1	1	21	5	0	0	-1	0	0
EditModesPane	9	0	15	10	247	42	0	2	0.2	1	9
ActionHandler	0	0	1	1	21	4	0	0	-1	0	4
MyCellRenderer	0	0	1	1	10	1	0	0	-1	0	1
MyListSelectionListener	0	0	1	1	13	3	0	0	-1	0	0
EditingOptionPane	3	0	3	1	32	3	0	2	0.66666666	0	3
EncodingsOptionPane	6	0	3	1	85	13	0	2	0.66666666	0	5
SyntaxHiliteOptionPane	7	1	18	10	146	30	0	2	0.33333333	5	11
MouseHandler	0	0	1	1	16	4	0	0	-1	0	4
StyleTableModel	4	0	12	8	89	21	0	0	0.41666666	1	8
StyleChoice	3	0	2	1	13	2	0	0	0	2	1

– **Method level**
  - StatusBarOptionPane.java

Table 6: Method level metrics calculated by DesigniteJava for Jedit Project

Class Name	Method Name	LOC	CC	PC
StatusBarOptionPane	StatusBarOptionPane	3	1	0
	_init	89	2	0
	_save	20	3	0
	updateButtons	7	1	0
	updatePreview	10	3	0
	actionPerformed	56	12	1
	valueChanged	3	1	1
	getListCellRendererComponent	8	2	5
	WidgetSelectionDialog	78	6	2
	WidgetSelectionDialog	3	1	1
	ok	9	2	0
	cancel	4	1	0
	getValue	3	1	0
	actionPerformed	19	5	1

2.2 Metrics for the modified project modules.

1. **Class level metrics**
    - **This package is org.gjt.sp.jedit.search**

Table 7: Class level metrics calculated by DesigniteJava for modified pdfsam-merge

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FAN-OUT
SearchMatcher	6	3	7	4	83	13	2	0	0.2857	7	2
Match	3	3	1	1	8	1	0	0	0	7	0
SearchAndReplace	12	0	44	33	805	151	0	0	0.0454	14	26
HyperSearchResults	26	6	60	37	634	132	0	1	0.2166	5	28
ActionHandler	0	0	1	1	27	8	0	0	-1	1	5
HighlightingTree	2	0	2	1	51	9	0	0	0	0	4
KeyHandler	0	0	2	2	26	5	0	0	-1	0	0
MouseHandler	1	0	2	1	44	10	0	0	0	0	3
RemoveTreeNodeAction	0	0	2	1	8	2	0	0	-1	0	1
RemoveAllTreeNodesAction	0	0	2	1	8	2	0	0	-1	0	1
NewSearchAction	0	0	2	1	15	3	0	0	-1	0	5
ExpandChildTreeNodesAction	0	0	2	1	10	2	0	0	-1	0	1
CopyToClipboardAction	0	0	2	1	14	2	0	0	-1	0	3
ToStringNodes	1	0	1	1	14	3	0	1	0	3	1
CollapseChildTreeNodesAction	0	0	2	1	14	3	0	0	-1	0	1
RedoSearchAction	1	0	2	1	16	2	0	0	0	0	3
TreeDisplayAction	0	0	1	1	30	4	0	0	-1	0	1
GoToNodeAction	1	0	2	1	10	2	0	0	0	0	1
ResultCellRenderer	4	0	4	1	45	9	0	1	0.5	0	4
CountNodes	2	0	1	1	12	2	0	1	0	2	0
ResultTreeTransferHandler	0	0	1	1	13	2	0	0	-1	0	1
ResultVisitor	0	0	1	0	3	1	2	0	-1	1	2
BufferLoadedVisitor	0	0	1	1	5	1	0	1	-1	0	2
BufferClosedVisitor	0	0	1	1	5	1	0	1	-1	0	2
TreeNodeCallbackAdapter	0	0	1	1	5	1	0	1	-1	0	0
BoyerMooreSearchMatcher	7	0	8	5	150	31	0	1	0	1	3
HyperSearchFileNode	4	2	7	7	51	10	0	1	0.2857	5	6
PatternSearchMatcher	3	0	7	6	170	36	0	1	0	2	3
SearchFileSet	0	0	7	0	43	7	2	0	-1	6	1
HyperSearchRequest	7	0	8	4	172	27	0	2	0	0	16
SearchBar	16	0	16	12	262	51	0	0	0.125	1	12
ActionHandler	0	0	1	1	23	7	0	0	-1	0	2
DocumentHandler	0	0	3	3	32	9	0	0	-1	0	2
KeyHandler	0	0	1	1	19	5	0	0	-1	0	0

FocusHandler	0	0	1	1	5	1	0	0	-1	0	0
HyperSearchFolderNode	3	0	3	3	24	4	0	0	0	3	0
DirectoryListSet	3	0	9	8	68	9	0	2	0	2	4
CurrentBufferSet	0	0	7	7	30	9	0	1	-1	0	1
HyperSearchResult	13	11	17	7	139	31	0	1	0	7	12
Occur	5	5	3	0	19	4	0	0	0	2	1
GotoDelayed	2	0	4	2	41	10	0	0	0	0	4
HyperSearchNode	0	0	2	0	8	2	2	0	-1	1	2
HyperSearchOperationNode	5	0	10	10	128	30	0	0	0.3	5	4
SearchDialog	36	3	33	20	721	94	0	1	0.0606	6	17
ReplaceActionHandler	0	0	1	1	6	1	0	0	-1	0	1
SettingsActionHandler	0	0	1	1	9	3	0	0	-1	0	0
MultiFileActionHandler	0	0	2	1	24	7	0	0	-1	0	4
ButtonActionHandler	0	0	1	1	39	12	0	0	-1	0	2
FocusOrder	1	0	7	7	46	11	0	0	0	0	0
BufferListSet	1	0	10	8	85	30	2	1	0.2	0	4
HyperSearchTreeNodeCallback	0	0	1	1	6	1	3	0	-1	1	0
AllBufferSet	2	0	5	4	55	7	0	2	0	0	3

- **Method Level**
  - **HyperSearchResults.Java**

Table 8: Method level metrics calculated by DesigniteJava for the modified Jedit project

Class Name	Method Name	LOC	CC	PC
HyperSearchResults	removeSelectedNode	23	5	0
	removeAllNodes	6	1	0
	hideDockable	3	1	0
	trimSearchString	8	2	0
	parseHighlightStyle	4	1	1
	actionPerformed	25	8	1
	HighlightingTree	8	2	1
	convertValueToText	39	7	6
	keyPressed	24	4	1
	run	3	1	0
	mousePressed	10	4	1
	showPopupMenu	31	6	1
	RemoveTreeNodeAction	3	1	0
	actionPerformed	3	1	1
	RemoveAllTreeNodesAction	3	1	0
	actionPerformed	3	1	1
	NewSearchAction	3	1	0

- **Class Level**
  - **This package is org.gjt.sp.jedit.gui**

Table 9: Class level metrics calculated by DesigniteJava for the modified Jedit project

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
BufferSwitcherTransferable	3	0	4	4	19	5	0	0	0.5	0	2
BufferTransferableData	2	0	3	3	14	3	0	0	0	2	1
BufferSwitcherTransferHandler	0	0	5	5	103	15	0	0	-1	0	9
ErrorListDialog	4	0	9	9	128	19	0	1	0.333	3	5
ErrorEntry	2	0	4	4	43	9	0	0	0	3	3
JTextPaneSized	0	0	1	1	11	1	0	0	-1	0	0
ActionHandler	0	0	1	1	8	3	0	0	-1	1	1
DockingLayoutManager	10	0	20	9	166	52	0	1	0.1	3	14
LoadPerspectiveAction	1	0	2	2	10	2	0	2	1	0	3
TipOfTheDay	5	0	5	4	84	10	0	1	0.6	0	4
ActionHandler	0	0	1	1	10	4	0	0	-1	0	1
AddModeDialog	8	0	9	9	116	16	0	1	0	0	5
ActionHandler	0	0	1	1	24	5	0	0	-1	0	4
PingPongList	16	1	47	42	301	70	0	0	0.148	2	5
MyListModel	1	0	8	7	35	10	0	0	0	2	0
MyTransferHandler	2	0	4	3	58	10	0	0	0.75	1	3
MyTransferable	2	1	5	3	19	5	0	0	0.4	2	0
ActionHandler	0	0	1	1	15	3	0	0	-1	0	0
MyListDataListener	0	0	4	3	15	4	0	0	-1	0	0
CloseDialog	9	0	8	7	150	23	0	1	0	0	8
ActionHandler	0	0	1	1	51	11	0	0	-1	0	4
ListHandler	0	0	1	1	18	4	0	0	-1	1	3
FilePropertiesDialog	9	0	11	10	195	29	0	1	0	0	12
ButtonActionHandler	0	0	1	1	11	3	0	0	-1	1	0
AboutPanel	25	0	9	6	152	24	0	0	0	2	5
HistoryModel	8	0	20	19	158	31	0	1	0	11	2
Renderer	0	0	1	1	27	8	0	0	-1	0	3
ListHandler	0	0	1	1	26	6	0	0	-1	0	2
MouseHandler	0	0	1	1	18	5	0	0	-1	1	3
DocumentHandler	0	0	5	3	26	6	0	0	-1	0	0
FocusHandler	0	0	2	2	8	2	0	0	-1	0	0
EscapeHandler	0	0	1	1	6	1	0	0	-1	0	0
TabHandler	0	0	1	1	5	1	0	0	-1	0	0
InsertHandler	0	0	1	1	5	1	0	0	-1	0	0
ClearHandler	0	0	1	1	5	1	0	0	-1	0	0
Entry	3	0	4	4	22	4	0	0	0	0	0
DummyRenderer	0	0	1	1	5	1	0	0	-1	0	0
LabelRenderer	2	0	2	1	14	3	0	0	0	0	1
CheckBoxListModel	1	0	10	7	62	19	0	0	0.4	1	3
AbstractContextOptionPane	12	0	14	4	210	30	2	2	0	3	9
MenuItemCompare	0	0	1	1	5	1	0	0	-1	0	2
MenuItem	2	0	2	1	11	2	0	0	0	4	1

ActionHandler	0	0	1	1	61	11	0	0	-1	0	6
ListHandler	0	0	1	1	5	1	0	0	-1	0	0
MutableListModel	0	0	2	2	7	2	2	0	-1	1	0
DefaultInputHandler	0	0	8	8	139	28	0	2	-1	1	5
MouseHandler	1	0	1	1	10	2	0	0	0	0	1
ContainerHandler	0	0	4	2	26	8	0	0	-1	0	0
KeyHandler	0	0	2	1	49	18	0	0	-1	3	2
WindowHandler	0	0	1	1	5	1	0	0	-1	0	0
StatusBar	25	0	22	20	329	62	0	0	0.272	4	14
MouseHandler	0	0	1	1	8	2	0	0	-1	0	0
AbbrevEditor	3	0	8	8	146	24	0	0	0	2	1
RolloverButton	2	0	9	9	70	11	6	0	0.444	8	0
MouseOverHandler	0	0	2	2	13	2	0	0	-1	0	0
InputHandler	1	0	14	9	264	68	1	1	0.357	3	18
AnimatedIcon	6	0	10	10	68	12	0	0	0	0	0
Animator	0	0	1	1	7	1	0	0	-1	0	0
FontSelectorDialog	12	0	14	8	209	30	0	1	0	1	4
ActionHandler	0	0	1	1	6	3	0	0	-1	0	0
ListHandler	0	0	1	1	18	7	0	0	-1	0	0
BeanShellErrorDialog	0	0	2	2	13	2	0	2	-1	0	1
DockableWindowFactory	22	0	36	24	301	60	0	0	0.138	9	11

- **Method Level**
  - Statusbar.java

Table 10: Method level metrics calculated by DesigniteJava for modified Jedit project

Class Name	Method Name	LOC	CC	PC
StatusBar	StatusBar	28	1	1
	propertiesChanged	54	6	0
	addNotify	4	1	0
	removeNotify	4	1	0
	run	17	4	0
	waiting	3	1	1
	running	2	1	1
	done	3	1	1
	statusUpdated	2	1	1
	maximumUpdated	2	1	1
	valueUpdated	2	1	1
	getMessage	3	1	0
	setMessageAndClear	12	2	1
	actionPerformed	3	2	1
	setMessage	12	4	1
	setMessageComponent	8	2	1
	updateCaretStatus	77	15	0
	updateBufferStatus	9	1	0
	updateMiscStatus	5	1	0
	getWidget	13	11	1
	_getWidget	7	2	1

- **Class level**

- **Org.gjt.sp.jedit.options**

Table 11: Class level metrics calculated by DesigniteJava for the modified jedit project

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
MouseHandler	0	0	1	1	11	3	0	0	-1	0	1
BrowserColorsModel	3	0	16	10	115	27	0	0	0.4375	1	3
Entry	2	0	1	0	8	1	0	0	0	3	0
ColorRenderer	0	0	2	1	19	4	0	0	-1	0	1
ModeSettingsPane	40	0	12	2	287	24	0	2	0.25	1	6
ActionHandler	0	0	1	1	13	3	0	0	-1	0	0
ModeProperties	19	0	4	0	115	11	0	0	0.5	1	2
ShortcutsOptionPane	17	1	42	28	450	95	0	2	0.214285714	0	14
HeaderMouseHandler	0	0	1	1	17	4	0	0	-1	0	1
TableMouseHandler	0	0	1	1	10	3	0	0	-1	0	2
ActionHandler	0	0	1	1	63	14	0	0	-1	0	4
ShortcutsModel	4	1	14	12	100	30	0	0	0.142857143	3	4
KeyCompare	1	0	2	1	26	7	0	0	0	0	1
KeymapsModel	2	0	7	4	33	11	0	0	0	2	2
KeymapCellRenderer	0	0	1	1	8	1	0	0	-1	0	1
GlobalOptions	0	0	8	5	44	10	0	2	-1	0	5
PrintOptionPane	10	0	3	1	62	3	0	2	0.6666	0	2
StatusBarOptionPane	30	0	14	8	369	41	0	2	0.2142	0	9
ActionHandler	0	0	1	1	58	12	0	0	-1	0	1
ListHandler	0	0	1	1	5	1	0	0	-1	0	0
WidgetListCellRenderer	0	0	1	1	10	2	0	0	-1	0	1
WidgetSelectionDialog	9	0	6	4	129	16	0	1	0.333333333	2	4
ActionHandler	0	0	1	1	21	5	0	0	-1	0	0
EditModesPane	9	0	15	10	247	42	0	2	0.2	1	9
ActionHandler	0	0	1	1	21	4	0	0	-1	0	4
MyCellRenderer	0	0	1	1	10	1	0	0	-1	0	1
MyListSelectionListener	0	0	1	1	13	3	0	0	-1	0	0
EditingOptionPane	3	0	3	1	32	3	0	2	0.6666	0	3
EncodingsOptionPane	6	0	3	1	85	13	0	2	0.6666	0	5
SyntaxHiliteOptionPane	7	1	18	10	146	30	0	2	0.3333	5	11
MouseHandler	0	0	1	1	16	4	0	0	-1	0	4
StyleTableModel	4	0	12	8	89	21	0	0	0.4166	1	8
StyleChoice	3	0	2	1	13	2	0	0	0	2	1

– **Method level**
  - StatusBarOptionPane.java

Table 12: Method level metrics calculated by DesigniteJava for modified Jedit project

Class Name	Method Name	LOC	CC	PC
StatusBarOptionPane	StatusBarOptionPane	3	1	0
	_init	95	2	0
	_save	22	3	0
	updateButtons	7	1	0
	updatePreview	10	3	0
	actionPerformed	56	12	1
	valueChanged	3	1	1
	getListCellRendererComponent	8	2	5
	WidgetSelectionDialog	78	6	2
	WidgetSelectionDialog	3	1	1
	ok	9	2	0
	cancel	4	1	0
	getValue	3	1	0
	actionPerformed	19	5	1

2.3 Affected Metrics

- **Class level Metrics**

Table 13: Affected Metrics in the Class Level for Jedit project

Version	Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FAN-OUT
Original	HyperSearchResults	26	6	60	37	637	132	0	1	0.216	5	28
Modified	HyperSearchResults	26	6	60	37	634	132	0	1	0.216	5	28
Original	HighlightingTree	2	0	2	1	54	9	0	0	0	0	4
Modified	HighlightingTree	2	0	2	1	51	9	0	0	0	0	4
Original	StatusBar	25	0	22	20	306	59	0	0	0.272	4	14
Modified	StatusBar	25	0	22	20	329	62	0	0	0.272	4	14
Original	StatusBarOptionPane	28	0	14	8	359	41	0	2	0.214	0	9
Modified	StatusBarOptionPane	30	0	14	8	369	41	0	2	0.214	0	9

- **Method Level Metrics**

Table 14: Affected Metrics in the method Level for Jedit project

version	Class Name	Method Name	LOC	CC	PC
Original	HyperSearchResults	convertValueToText	42	7	6
Modified	HyperSearchResults	convertValueToText	39	7	6
Original	StatusBar	updateCaretStatus	54	12	0
Modified	StatusBar	updateCaretStatus	77	15	0
Original	StatusBarOptionPane	_init	89	2	0
Modified	StatusBarOptionPane	_init	95	2	0
Original	StatusBarOptionPane	_save	20	3	0
Modified	StatusBarOptionPane	_save	22	3	0

As we noticed, the most metrics that have changed at the method-level are Lines of code and Cyclomatic complexity. At the class level however, there are other measures which are Weighted methods per class, number of fields and LOC of classes that have been affected.

The lines of code that we had added when making these changes are the direct cause of the increase of the LOC metric. But more interestingly is the unexpected increase in cyclomatic complexity.

After looking into our code changes, we see that we have added certain four statements that create independent paths that will increase the number of paths in the updateCaretStatus() function which leads to the increase of **Cyclomatic Complexity.**

The increase of Cyclomatic Complexity reflects in increasing the values of **Weighted methods per class**. As we mentioned above, Weighted Methods per Class metric is the sum of the cyclomatic complexity of the functions in the statusbar class. Therefore, an increase in the WMC can be attributed to the increase in the Cyclomatic complexity of each method.

The final metric is the N**umber of fields** in the statusbarOptionPane class. There are two additional checkboxes that are defined to implement the configurability of the change in the Options -> StatusBar menu; that is why this metric is affected.

3 Cohesion and Coupling

3.1 Cohesion

3.1.1 Types of Cohesion measured

LCOM (Lack of Cohesion in Methods – Class level metric): The metric that is shown by the tool to measure the cohesion is LCOM. This metric measures the **functional cohesion** since it focuses on how class elements (methods and attributes) are related to each other. As it is mentioned in the descriptions above, LCOM analyzes the relation between attributes and methods.

Table 15: High/Low Cohesion Classes for Jedit project

3.1.2 Classes and their Cohesion values

Table 15: High/Low Cohesion Classes for Jedit project

Classes	Level of cohesion (High /Low)	LCOM
PrintRangeType.java Org/gjt/sp/jedit/print/	Low	1
BSHFormalParameter org.gjt.sp.jedit.bsh	Low	1
BSHEnhancedForStatement.jav org.gjt.sp.jedit.bsh	Low	1
BSHWhileStatement.java Org.gjt.sp.jedit.bsh	Low	1
ColumnBlockLine Org/gjt/sp/jedit/textarea	high	0
BufferOptions.java Org.gjt.sp.jedit.gui	high	0
Debug.java Org/gjt/sp/jedit/	high	0
RectSelectWidgetFactory.java Org/gjt/sp/jedit/gui/statusbar/	High	0

In terms of low cohesion classes, according to the data in the cohesion table, LCOM value for **PrintRangeType**, **BSHFormalParameter**, **BSHEnhancedForStatement**, and **BSHWhileStatement** is 1 which is the highest LCOM values among all classes' cohesion values in Jedit project. The **BSHWhileStatement** and **BSHEnhancedForStatement are subclasses** of **SimpleNode**.java and **ParserConstants** interface. In the case of an inheritance relationship, LCOM considers attributes that are present in the parent class but only accessed by its subclasses (**BSHWhileStatement**). There are 131 attributes in the parent class; only 3 attributes are accessed in the child class. Similarly, the **BSHFormalParameter** which is derived from the SimpleNode parent class, does not access attributes of the base class and the methods of this class do not intersect in using instance variables of the same child class. The **PrintRangeType** class which is a child class of **IntegerSyntax** has the same problem as the previous classes that leads to low cohesion of the class.

However, in terms of high cohesion classes, all the next four classes have a high value of LCOM, which is 0. This is the lowest LCOM value for non-trivial classes. After looking at these classes (**ColumnBlockLine**,

BufferOptions and RectSelectWidgetFactory), we found that the methods of these classes intersect in accessing the instance variables of the class and these methods lead to one objective of the class. In the case of the **Debug** class, there are no attributes in the class, and it has only one method. Therefore, LCOM equals to 0.

3.1.3 The difference between the higher and lower cohesion classes

After we checked the classes, we found that high cohesion classes have one functionality that is intended to be implemented, and if there are attributes in the classes, they are mostly accessed. However, in the low cohesion classes, most of the methods in the child classes do not access the instance variables in the base class which leads to a considerable number of components to be occurred in the class. This might lead to misunderstanding of code, which causes the complexity of code maintenance.

3.2 Coupling

3.2.1 Types of Coupling measured

FANIN and FANOUT (Class level metric): Fan-in and Fan-out values measure the relationships between classes and between procedures. In the project modules, the Fan-in for a class can be understood as the number of classes in the same module that refers to the class. The Fan-out can conversely be defined as the number of classes that a given class depends on.
As mentioned earlier in section 1, this class 'dependency' may be a procedure call in another class, reading/writing of variables, accessing constants, or using sub-classes, interfaces, or enum declarations.
Since any form of inter-class reference is considered under this umbrella term, we feel that It would measure **Data, Stamp, Control and Content coupling** very well.

1. Content Coupling: Any direct reference between class members (attributes or methods) will be counted as part of the fan-in metric for the referenced class and the fan-out metric for the referring class.
2. Control Coupling: This kind of coupling involves the passing of a control switch between modules. This passing in most cases would involve a method call. Which again would be counted in the fan-in and fan-out metrics as explained above.
3. Stamp Coupling: Similarly, any controlled, intentional passing of data between classes as takes place in stamp coupling would also require a function-call of sorts. These would count as dependencies between methods and be included in the Fan-in/fan-out metric calculation.
4. Data Coupling: A better form of stamp coupling, there is no doubt that data coupling will be counted through these metrics.

3.2.2 *Classes with high/low coupling (and why)*

Table 16: High/Low Coupling Classes for Jedit project

Classes	Level of coupling (High /Low)	FANIN
Jedit.java Org/gjt/sp/jedit/	high	331
Log.java Org/gjt/sp/unit/	high	191
view.java Org/gjt/sp/jedit/	high	147
GUIUtilities.java Org/gjt/sp/jedit/help/	high	128
HyperSearchRequest.java Org/gjt/sp/jedit/search	Low	0
TabbedOptionDialog.java Org/gjt/sp/jedit/	Low	0
JEditMode.java Org/gjt/sp/jedit/	Low	0
WhitespaceRule.java Org/gjt/sp/jedit/	Low	0

As it is shown by the DesigniteJava tool, there are 2 metrics that measure the coupling between classes. We chose the FANIN metric to measure the coupling in Jedit while we chose FANOUT to measure the coupling of pdfsam projects.

The first four classes have high coupling based on FANIN results. The first class I (**jedit)** has the highest FANIN value in the project since it is referenced in 331 classes. Moving to **log** which has the second-highest value for FANIN metric (191), this refers to the number of classes that depend on this **log** class. The next two classes that have the high coupling are **view** and **GUIUtilities** classes, which means also that they are used by a large number of classes 147 and 128, respectively, as it is computed by FANIN metric. The second four classes have the lowest coupling values based on the results of the FANIN metric. This metric returns 0 for these classes; **HyperSearchRequest, TabbedOptionDialog**, **JEditMode**, and **WhitespaceRule.** That means that these four classes are not referenced in other classes.

3.2.3 *The difference between the higher and lower coupling classes*

In the case of high coupling classes, the modification of code needs effort and time because of the dependency between the classes. As we notice from this table, there are large numbers of classes that depend on other classes. If there is a change that is applied to a class that is referenced by others, the change of these classes must be considered. Therefore, the high dependency leads to less ability of maintenance; however, the low dependency facilitates the change of code by maintainers.

PDFsam

TOOL: DESIGNITEJAVA

1. The metrics analyzed

2. Method level metrics
    a. **Lines of Code (LOC)**: Computes the lines of **non-comment, non-blank** code in the function.
    b. **Cyclomatic complexity (CC)**: At the method level, computes the number of possible paths to measure the complexity of the function.
    c. **Parameter Count (PC)**: The number of arguments given to a function.
3. Class level metrics
    a. **Number of fields (NOF)**: calculates the number of data members of a class.
    b. **Number of public fields (NOPF)):** calculates the number of public data-members in a class.
    c. **Number of methods (NOM):** calculates the total number of methods in a class.
    d. **Number of public methods (NOPM)**: calculates the number of public methods in a class.
    e. **Lines of code (LOC)**: Computes the number of lines of code in line with the conditions highlighted above (1.a) but for the whole class.
    f. **Weighted methods per class (WMC)**: The Cyclomatic Complexity values of each method in the class are summed to find the complexity of a class, this is the WMC. It can be used to determine whether a class needs to be restructured into smaller classes. WMC should be low, since it will keep classes restricted to a single responsibility and prevent a greater potential impact on derived classes.
    g. **Number of children (NOC)**: NOC equals the number of immediate child classes derived from a base class.
    h. **Depth of inheritance tree (DIT)**: Finds the position of the class in the Inheritance tree. This is used to measure the complexity of the class structure
    i. **Lack of cohesion in methods (LCOM)**: According to Chidamber & Kemerer, to calculate LCOM, two methods of a class are checked; if they accessed different variables, P is increased by one. However, if they access the same variable, Q is increased by one. The lower LCOM, the more cohesive class. It can also be used to determine whether a class needs to be restructured into smaller classes, like WMC.

LCOM1 = P - Q , if P > Q

LCOM1 = 0 otherwise

LCOM1 = 0 indicates a cohesive class.

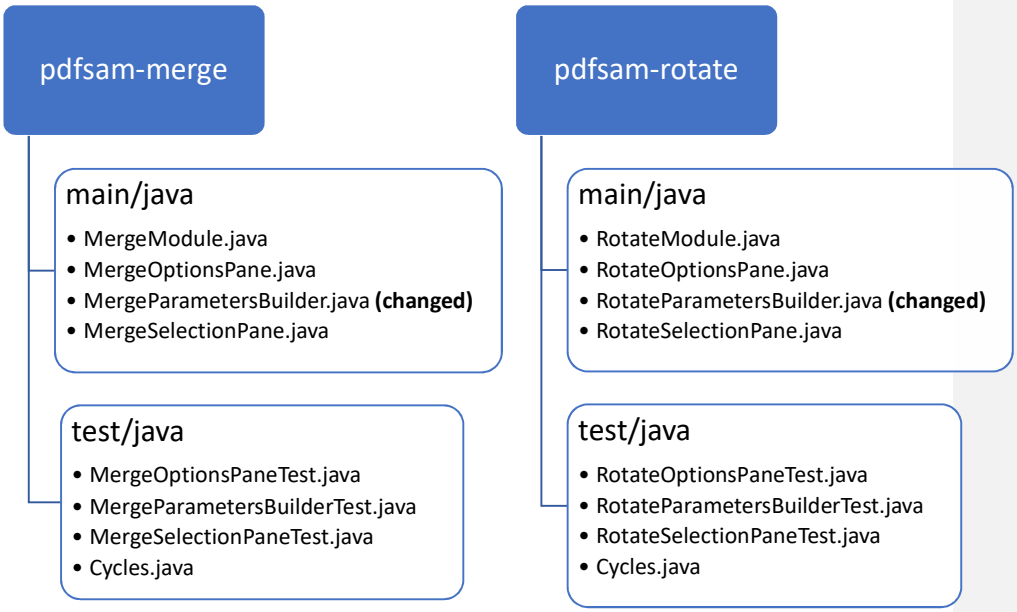
LCOM1 > 0 indicates that the class needs or can be split into two or more classes, since its variables belong in disjoint sets.

---

j. **Fan-in(FANIN)**: For a given class X, the fan-in structural metric can be used to calculate how much code is being reused. It can be defined as the number of classes that depend on class X. Where 'dependence' could be using X's methods, its variables, constants or use its sub-classes or enums.

k. **Fan-out(FANOUT)**: Fan-out denotes how many classes a given class Y depends on. Here, Y's dependence includes using other classes' methods, variables, enums or constants. Fan out can be used to calculate the coupling of classes, high FO denotes strongly coupled code. The class Y depends on other code and is probably more complex to execute and test.

2. The projects analyzed



3. Analyzing code metrics

3.1. Metrics for the original project modules.

A limitation we faced in this project (illustrated in Section2) was the fact that each of the modified projects, pdfsam-merge and pdfsam-rotate had a total of only 8 class files each. Of these 8 classes, 4 were used in the functionality of the application, 3 were test classes and one class (Cycles.java) was an empty class. These projects are designed in a highly independent manner (one stand-alone module each for the merge and rotate functionalities). This makes them highly cohesive internally and with low coupling to the rest of the code.

- **pdfsam-merge**

Table 17: Class level metrics calculated by DesigniteJava for pdfsam-merge

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
MergeModule	6	0	13	11	74	14	0	0	0.23077	0	3
ModuleConfig	0	0	4	4	14	4	0	0	-1	0	0
MergeOptionsPane	6	0	5	4	91	5	0	0	0	2	1
MergeParametersBuilder	8	0	10	2	56	10	0	0	0	6	0
MergeSelectionPane	1	0	2	2	22	3	0	0	1	2	1
MergeOptionsPaneTest	4	1	6	6	92	6	0	0	0	0	2
MergeParametersBuilderTest	1	1	1	1	36	1	0	0	0	0	1
MergeSelectionPaneTest	7	3	7	6	61	7	0	0	0	0	2
Cycles	0	0	0	0	5	0	0	0	-1	0	0

Table 18: Method level metrics calculated by DesigniteJava for pdfsam-merge

Class Name	Method Name	LOC	CC	PC
MergeModule	MergeModule	6	1	3
	descriptor	3	1	0
	onSaveWorkspace	6	1	1
	onLoadWorkspace	6	1	1
	getBuilder	8	1	1
	settingPanel	7	1	0
	id	3	1	0
	onClearModule	6	2	1
	graphic	3	1	0
	destinationFileField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
ModuleConfig	destinationFileField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0

MergeOptionsPane	MergeOptionsPane	47	1	0
	resetView	8	1	0
	apply	8	1	2
	saveStateTo	8	1	1
	restoreStateFrom	8	1	1
MergeParametersBuilder	addInput	3	1	1
	hasInput	3	1	0
	outlinePolicy	3	1	1
MergeParametersBuilder	blankPageIfOdd	3	1	1
	footer	3	1	1
	normalize	3	1	1
	acroFormsPolicy	3	1	1
	tocPolicy	3	1	1
	output	3	1	1
	build	15	1	0
MergeSelectionPane	MergeSelectionPane	3	1	1
	apply	12	2	2
MergeOptionsPaneTest	setUp	4	1	0
	start	6	1	1
	validSteps	11	1	0
	onSaveWorkspace	11	1	0
	restoreStateFrom	22	1	0
	reset	29	1	0
MergeParametersBuilderTest	build	30	1	0
MergeSelectionPaneTest	setUp	5	1	0
	empty	5	1	0
	emptyByZeroPagesSelected	7	1	0
	emptyPageSelection	9	1	0
	notEmptyPageSelection	10	1	0
	converstionException	7	1	0
	populate	6	1	0

- **pdfsam-rotate**

Table 19: Class level metrics calculated by DesigniteJava for pdfsam-rotate

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
RotateModule	7	0	14	12	87	15	0	0	0.21429	0	3
ModuleConfig	0	0	5	5	17	5	0	0	-1	0	0
RotateOptionsPane	2	0	5	4	39	5	0	0	0	2	1
RotateParametersBuilder	5	0	9	5	50	10	0	0	0	6	0
RotateSelectionPane	1	0	2	2	22	3	0	0	1	2	1
RotateOptionsPaneTest	2	1	5	5	50	5	0	0	0	0	2
RotateParametersBuilderTest	2	1	4	4	60	4	0	0	0	0	1
RotateSelectionPaneTest	7	3	7	6	61	7	0	0	0	0	2
Cycles	0	0	0	0	5	0	0	0	-1	0	0

Table 20: Method level metrics calculated by DesigniteJava for pdfsam-rotate

Class Name	Method Name	LOC	CC	PC
RotateModule	RotateModule	7	1	4
	descriptor	3	1	0
	getBuilder	9	1	1
	onSaveWorkspace	7	1	1
	onLoadWorkspace	7	1	1
	settingPanel	11	1	0
	id	3	1	0
	onClearModule	7	2	1
	graphic	3	1	0
	destinationDirectoryField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
	prefixPane	3	1	1
ModuleConfig	destinationDirectoryField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
	prefixPane	3	1	1
RotateOptionsPane	RotateOptionsPane	15	1	0
	resetView	4	1	0
	apply	4	1	2
	saveStateTo	4	1	1
	restoreStateFrom	4	1	1
RotateParametersBuilder	addInput	8	2	2
	hasInput	3	1	0
	output	3	1	1
	prefix	3	1	1

	getOutput	3	1	0
RotateParametersBuilder	getPrefix	3	1	0
	rotation	3	1	1
	rotationType	3	1	1
	build	10	1	0
RotateSelectionPane	RotateSelectionPane	3	1	1
	apply	12	2	2
RotateOptionsPaneTest	start	6	1	1
	validSteps	8	1	0
	onSaveWorkspace	6	1	0
	restoreStateFrom	10	1	0
	reset	13	1	0
RotateParametersBuilderTest	setUp	8	1	0
	buildDefaultSelection	20	1	0
	buildRanges	14	1	0
	buildMultiple	11	1	0
RotateSelectionPaneTest	setUp	5	1	0
	empty	5	1	0
	emptyPageSelection	9	1	0
	notEmptyPageSelection	10	1	0
	converstionException	7	1	0
	emptyByZeroPagesSelected	7	1	0
	populate	6	1	0

3.2. Metrics for the modified project modules.

Our changes to the code to implement the change requests were minimal, with minor adjustments to how the input was handled. We did not expect many changes in the code quality metrics of these modules. The Eclipse tool o3smeasures that we used did not show us any change to any of the 26 metrics calculated by it. Some of the modified metrics shown by DesigniteJava were expected, but a few surprised us.

- **pdfsam-merge**

Table 21: Class level metrics calculated by DesigniteJava for modified pdfsam-merge

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
MergeModule	6	0	13	11	74	14	0	0	0.23077	0	3
ModuleConfig	0	0	4	4	14	4	0	0	-1	0	0
MergeOptionsPane	6	0	5	4	91	5	0	0	0	2	1
MergeParametersBuilder	8	0	10	2	62	11	0	0	0	6	0
MergeSelectionPane	1	0	2	2	22	3	0	0	1	2	1
MergeOptionsPaneTest	4	1	6	6	92	6	0	0	0	0	2
MergeParametersBuilderTest	1	1	1	1	36	1	0	0	0	0	1
MergeSelectionPaneTest	7	3	7	6	61	7	0	0	0	0	2
Cycles	0	0	0	0	5	0	0	0	-1	0	0

Table 22: Method level metrics calculated by DesigniteJava for modified pdfsam-merge

Class Name	Method Name	LOC	CC	PC
MergeModule	MergeModule	6	1	3
	descriptor	3	1	0
	onSaveWorkspace	6	1	1
	onLoadWorkspace	6	1	1
	getBuilder	8	1	1
	settingPanel	7	1	0
	id	3	1	0
	onClearModule	6	2	1
	graphic	3	1	0
	destinationFileField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
ModuleConfig	destinationFileField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
MergeOptionsPane	MergeOptionsPane	47	1	0
	resetView	8	1	0
	apply	8	1	2

	saveStateTo	8	1	1
	restoreStateFrom	8	1	1
MergeParametersBuilder	addInput	9	2	1
	hasInput	3	1	0
	outlinePolicy	3	1	1
	blankPageIfOdd	3	1	1
MergeParametersBuilder	footer	3	1	1
	normalize	3	1	1
	acroFormsPolicy	3	1	1
	tocPolicy	3	1	1
	output	3	1	1
	build	15	1	0
MergeSelectionPane	MergeSelectionPane	3	1	1
	apply	12	2	2
MergeOptionsPaneTest	setUp	4	1	0
MergeOptionsPaneTest	start	6	1	1
	validSteps	11	1	0
	onSaveWorkspace	11	1	0
	restoreStateFrom	22	1	0
	reset	29	1	0
MergeParametersBuilderTest	build	30	1	0
MergeSelectionPaneTest	setUp	5	1	0
	empty	5	1	0
	emptyByZeroPagesSelected	7	1	0
	emptyPageSelection	9	1	0
	notEmptyPageSelection	10	1	0
	converstionException	7	1	0
	populate	6	1	0

- **pdfsam-rotate**

Table 23: Class level metrics calculated by DesigniteJava for modified pdfsam-rotate

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
RotateModule	7	0	14	12	87	15	0	0	0.214286	0	3
ModuleConfig	0	0	5	5	17	5	0	0	-1	0	0
RotateOptionsPane	2	0	5	4	39	5	0	0	0	2	1
RotateParametersBuilder	5	0	11	7	79	23	0	0	0.272727	6	0
RotateSelectionPane	1	0	2	2	22	3	0	0	1	2	1
RotateOptionsPaneTest	2	1	5	5	50	5	0	0	0	0	2
RotateParametersBuilderTest	2	1	4	4	60	4	0	0	0	0	1
RotateSelectionPaneTest	7	3	7	6	61	7	0	0	0	0	2
Cycles	0	0	0	0	5	0	0	0	-1	0	0

Table 24: Method level metrics calculated by DesigniteJava for modified pdfsam-rotate

Class Name	Method Name	LOC	CC	PC
RotateModule	RotateModule	7	1	4
	descriptor	3	1	0
	getBuilder	9	1	1
	onSaveWorkspace	7	1	1
	onLoadWorkspace	7	1	1
	settingPanel	11	1	0
	id	3	1	0
	onClearModule	7	2	1
	graphic	3	1	0
	destinationDirectoryField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
	prefixPane	3	1	1
ModuleConfig	destinationDirectoryField	3	1	0
	destinationPane	3	1	2
	footer	3	1	2
	openButton	3	1	0
	prefixPane	3	1	1
RotateOptionsPane	RotateOptionsPane	15	1	0
	resetView	4	1	0
	apply	4	1	2
	saveStateTo	4	1	1
	restoreStateFrom	4	1	1
RotateParametersBuilder	addInput	37	9	2
	accept	7	3	1

	accept	7	3	1
RotateParametersBuilder	hasInput	3	1	0
	output	3	1	1
	prefix	3	1	1
	getOutput	3	1	0
	getPrefix	3	1	0
	rotation	3	1	1
	rotationType	3	1	1
	build	10	1	0
RotateSelectionPane	RotateSelectionPane	3	1	1
	apply	12	2	2
RotateOptionsPaneTest	start	6	1	1
	validSteps	8	1	0
	onSaveWorkspace	6	1	0
	restoreStateFrom	10	1	0
	reset	13	1	0
RotateParametersBuilderTest	setUp	8	1	0
	buildDefaultSelection	20	1	0
	buildRanges	14	1	0
	buildMultiple	11	1	0
RotateSelectionPaneTest	setUp	5	1	0
	empty	5	1	0
	emptyPageSelection	9	1	0
	notEmptyPageSelection	10	1	0
	converstionException	7	1	0
	emptyByZeroPagesSelected	7	1	0
	populate	6	1	0

3.3. Affected Metrics

To summarize the changes that we have seen reflected in the metrics calculated by DesigniteJava, we present the tables below, showing relevant classes and the changes in the values of their measures.

- **pdfsam-merge**

Table 25: Contrast of class-level metrics before and after the change request implementation

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
Before Changes											
MergeParametersBuilder	8	0	10	2	56	10	0	0	0	6	0
After Changes											
MergeParametersBuilder	8	0	10	2	62	11	0	0	0	6	0

Table 26: Contrast of method-level metrics before and after the change request implementation

Class Name	Method Name	LOC	CC	PC		Mod. LOC	Mod. CC	Mod. PC
MergeParametersBuilder	addInput	3	1	1		9	2	1
	hasInput	3	1	0		3	1	0
	outlinePolicy	3	1	1		3	1	1
	blankPageIfOdd	3	1	1		3	1	1
	footer	3	1	1		3	1	1
	normalize	3	1	1		3	1	1
	acroFormsPolicy	3	1	1		3	1	1
	tocPolicy	3	1	1		3	1	1
	output	3	1	1		3	1	1
	build	15	1	0		15	1	0

- **pdfsam-rotate**

Table 27: Contrast of class-level metrics before and after the change request implementation

Class Name	NOF	NOPF	NOM	NOPM	LOC	WMC	NC	DIT	LCOM	FANIN	FANOUT
Before Changes											
RotateParametersBuilder	5	0	9	5	50	10	0	0	0	6	0
After Changes											
RotateParametersBuilder	5	0	11	7	79	23	0	0	0.272727	6	0

Table 28: Contrast of class-level metrics before and after the change request implementation

Class Name	Method Name	LOC	CC	PC		Mod. LOC	Mod. CC	Mod. PC
RotateParametersBuilder	addInput	8	2	2		37	9	2
	accept					7	3	1
	accept					7	3	1
	hasInput	3	1	0		3	1	0
	output	3	1	1		3	1	1
	prefix	3	1	1		3	1	1
	getOutput	3	1	0		3	1	0
	getPrefix	3	1	0		3	1	0
	rotation	3	1	1		3	1	1
	rotationType	3	1	1		3	1	1
	build	10	1	0		10	1	0

Notably, the most metrics that have changed at the method-level are **Lines of code** and **Cyclomatic complexity**. At the class level however, there are other measures such as **Weighted methods per class**, **number of methods, number of weighted methods**, and **Lack of cohesion in methods** that have been affected.

The lines of code that we had added when making these changes are the direct cause of the increase of the LOC metric. But more interestingly is the unexpected increase in **cyclomatic complexity**.

O3smeasures, the eclipse plugin tool that we had used before had not shown us these changes. Now that we look into our code changes, we see that we **have added if-else conditions** that will definitely **increase the number of paths** in the addInput() function (i.e. the Cyclomatic Complexity). We would have expected it to increase by the sum of the cyclomatic complexity of the two new functions in the Rotate Parameters Builder class, but it increased by 7, not 6 (3+3). This made us investigate the change itself, and after counting **the number of if-else statements**, we realized that there were 7.

A common change in both modified modules is the **Weighted methods per class** metric. This metric initially had us a little puzzled. We initially came across a definition describing WMC as a modification of the number of methods. Only after understanding how WCM was reflected in our changed modules, we could see that it was also a **sum of the cyclic complexity (**try it!**)**. This was a surprising connection that we were not expecting. Thus, an increase in the WMC can be attributed to the increase in Cyclomatic complexity explained above.

The **Lack of cohesion between modules** value changed for the pdfsam-rotate module after our modifications. This means that our changes, apart from increasing the number of paths in a class and the WMC, also reduced the overall cohesion of the class. Now going on the metric definition of LCOM provided in section 1, we see that its value is affected by whether methods of a class operate on the same variables. We suspect our introduction of a new local variable **'tempPageArray1'** to hold intermediate odd/even page ranges and **two** new lambda methods to operate exclusively on this variable are probably responsible for this change.

An interesting unexpected outcome is the addition of two new methods in the Rotate Parameters Builder class. This is because of our use of two anonymous functions implementing the **Consumer** interface. These two functions were part of **forEach** loops used to iterate over collections of Page Ranges.

A direct consequence of these forEach implementations is the increase in the **Number of methods** and **Number of public methods** metrics as shown in the table above. We expected these anonymous functions to be interpreted like lambda functions and not affect the metrics. However, what we learnt after examining the output of DesigniteJava is that even these anonymous Consumer functions are included in the calculation of these metrics.

Designitejava also measures **code smells**, and It has indicated that our changes have introduced a **magic number code smell** into the rotate-module code. This probably refers to the calculation deciding whether a page range is even or odd in the pdfsam-rotate project, in which we perform a modulo operation(mod2) on the given input page numbers. It also classifies the function that we had changed with a **complex class code smell**, indicating that our changes have transformed an earlier simple class to something convoluted and labyrinthine.

4. Cohesion and Coupling

4.1. Cohesion

4.1.1. Types of Cohesion measured

LCOM (Lack of Cohesion in Methods – Class level metric): The metric that is shown by the tool to measure the cohesion is LCOM. This metric measures the **functional cohesion** since it focuses on how class elements (methods and attributes) are related to each other. As it is mentioned in the descriptions above, LCOM analyzes the relation between attributes and methods.

4.1.2. Classes and their Cohesion values

- **pdfsam-merge**

Table 29: LCOM values and the level of cohesion for classes in pdfsam-merge

Classes	Level of cohesion (High /Low)	LCOM
ModuleConfig	NA	-1
Cycles	NA	-1
MergeOptionsPane	High	0
MergeParametersBuilder	High	0
MergeOptionsPaneTest	High	0
MergeParametersBuilderTest	High	0
MergeSelectionPaneTest	High	0
MergeModule	Low	0.230769
MergeSelectionPane	Low	1

- **pdfsam-rotate**

Table 30: LCOM values and the level of cohesion for classes in pdfsam-rotate

Classes	Level of cohesion (High /Low)	LCOM
ModuleConfig	NA	-1
Cycles	NA	-1
RotateOptionsPane	High	0
RotateOptionsPaneTest	High	0
RotateParametersBuilderTest	High	0
RotateSelectionPaneTest	High	0
RotateModule	Low	0.214286
RotateParametersBuilder	Low	0.272727
RotateSelectionPane	Low	1

A high LCOM value (closer to 1) indicates a conflict in the functionality written in a class. This metric can be used to identify classes that are attempting to achieve multiple

objectives. Such classes could be more error-prone and more difficult to test and could be split into two or more classes that are more well defined in their behavior[1].

These two projects(merge and rotate) are developed as **independent** modules of the main application, and show many good code practices, such **single-responsibility classes** (RotateOptionsPane, RotateParametersBuilderTest). Therefore, it is not surprising to see the values of LCOM staying low, around zero or in most cases, 0.

As discussed earlier in section 3.3, the modification of the RotateParametersBuilder class **increased** its LCOM value from 0 to **0.272727**.

Other patterns visible are the fact that the RotateModule class and the MergeModule class both have values that are > 0, this is possibly because they differ from other intra-project classes in that they contain an **extra component sub-class**(ModuleConfig). This sub-class is **not visibly called** anywhere within the class. This **seemingly unrelated class** being put inside the MergeModule/RotateModule classes may lead to the **increase of LCOM** compared to the other classes in the projects.

It was mentioned in a post by the developer of DesigniteJava[2] that one of the main challenges faced by modern LCOM calculation tools was the **dependence on method-attribute access**. It stated that the calculation of LCOM relied on **common attribute accesses** from methods as a basis to decide whether two methods are cohesive and that in the absence of such accesses, as seen in Cycles.java or the ModuleConfig sub-class, the tool may produce either a 0 or a non-deterministic LCOM value. The calculation of LCOM here for these respective classes is indeed a garbage-type value of -1.

4.1.3. The difference between the higher and lower cohesion classes

Above we discuss the cohesion values for different classes and possible reasons why they have these values. If asked specifically about the difference between the higher and lower cohesion value classes, we would point out the **presence of extra sub-classes** in these higher LCOM classes, which makes them more complex and thus more likely to have a higher LCOM. As mentioned earlier, the classes with LCOM = 0 such as the MergeOptionPane class are **single-purposely designed** to be components in other classes(MergeModule.java) and others like MergeOptionPaneTest class **are tests targeted to only one class**. These test classes are also ~~pretty~~ **independent of other components** apart from the ones they are testing.

Finally, the highest classes with LCOMs of 1(MergeSelectionPane and RotateSelectionPane), though another type of component, have only two methods, which do not share any parameters. These disjoint methods lead to a higher calculated LCOM, making the calculating tool believe that they are unrelated methods because they do not operate on the same attributes.

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: Bold

Formatted: Font: Bold

[1] https://www.aivosto.com/project/help/pm-oo-cohesion.html
[2] http://www.tusharma.in/technical/revisiting-lcom/

4.2. Coupling

4.2.1. Types of Coupling measured

FANIN and FANOUT (Class level metric): Fan-in and Fan-out values measure the relationships between classes and between procedures. In the pdfsam modules, the Fan-in for a class can be understood as the number of classes in the same module that refers to the class. The Fan-out can conversely be defined as the number of classes that a given class depends on.

As mentioned earlier in section 1, this class 'dependency' may be a procedure call in another class, reading/writing of variables, accessing constants, or using sub-classes, interfaces, or enum declarations.

Since any form of inter-class reference is considered under this umbrella term, we feel that It would measure **Data, Stamp, Control and Content coupling** very well.

1. Content Coupling: Any direct reference between class members (attributes or methods) will be counted as part of the fan-in metric for the referenced class and the fan-out metric for the referring class.
2. Control Coupling: This kind of coupling involves the passing of a control switch between modules. This passing in most cases would involve a method call. Which would be counted in the fan-in and fan-out metrics as explained above.
3. Stamp Coupling: Similarly, any controlled, intentional passing of data between classes as takes place in stamp coupling would also require a function-call or object declaration of sorts. These would count as dependencies between methods and be included in the Fan-in/fan-out metric calculation.
4. Data Coupling: A better form of stamp coupling in which only the relevant data is shared between classes, there is no doubt that this type of coupling will be counted through these metrics.

4.2.2. Classes with high/low coupling(and why)

- **pdfsam-merge**

Table 31: Fan-out values and the level of coupling for classes in pdfsam-merge

Type Name	Level of coupling (High /Low)	FANOUT
ModuleConfig	Low	0
MergeParametersBuilder	Low	0
Cycles	Low	0
MergeOptionsPane	Low	1
MergeSelectionPane	Low~~High~~	1
MergeParametersBuilderTest	Low~~High~~	1
MergeOptionsPaneTest	High	2
MergeSelectionPaneTest	High	2
MergeModule	High	3

- **pdfsam-rotate**

Table 32: Fan-out values and the level of coupling for classes in pdfsam-rotate

Type Name	Level of coupling (High /Low)	FANOUT
ModuleConfig	Low	0
RotateParametersBuilder	Low	0
Cycles	Low	0
RotateOptionsPane	Low ~~Low~~	1
RotateSelectionPane	Low~~High~~	1
RotateParametersBuilderTest	Low~~High~~	1
RotateOptionsPaneTest	High	2
RotateSelectionPaneTest	High	2
RotateModule	High	3

This tool shows different fan-out values compared to the value shown by the o3smeasures, which showed a consistent fan-out of 1 for all classes. A high Fan-in indicates a heavily used class, while a low Fan-in is the opposite. A high Fan-out means the class depends on many others. A class with Fan-out=0 is independent of others. While a high Fan-in can be an indication of good reuse of code, relatively large values within a project can represent a lot of cross-class coupling. A high Fan-out denotes strongly coupled code. The code depends on other code and is probably more complex to execute and test. A low or zero fan-out means independent, self-sufficient code. This kind of code is easier to reuse in another project or for another purpose.

In the tables above, we can see that the Fan-out for all the classes in the document is within the range 0-3. From our discussion above we know that a class with Fan-out=0 is independent of others. Cycles is an tempty class that has no members of its own and only extends the TestCycles class. This would mean it is dependent on at least one class: the TestCycles class. This dependency should have shown in its value of Fan-out but oddly, it is not reflected. As mentioned earlier, the o3smeasures tool showed a fan-out of 1, which makes more sense than the 0 mentioned by DesigniteJava. We suspect that is because we ran the tool only on the pdfsam-rotate project, and the TestCycles class is in a different module, pdfsam-core.

Other zero-valued fan-out classes are RotateParametersBuilder and ModuleConfig. Of these modules, RotateParametersBuilder extends a class, implements an interface and imports several packages as dependencies, yet its fan-out value is 0. Initially this was puzzling to understand and difficult to interpret, but looking at the case of Cycles.java discussed above, we can apply the same cause to this class as well. On the other hand, ModuleConfig is a simple sub-class within the RotateModule class and can be presumed to be an independent stand-alone class.

Among the classes with higher fan-out values, In classes like RotateModule instantiate the other classes (RotateParametersBuilder, RotateOptionsPane, RotateSelectionPane) and orchestrate them to perform the desired functions. Therefore, we can see that it depends

on three other classes in the same module/project to run and thus validate its Fanits Fan-out measure of 3.
Both RotateOptionsPaneTest, RotateSelectionPaneTest have a fan out value of 2 despite being test classes designed to test only one class. Looking into the classes, we can see they instantiate both their respective classes to be tested(RotateOptionsPane and RotateSelectionPane), and also one object of RotateParametersBuilder. This agrees with their fan-out metric value.
Similar analysis can be applied to the pdfsam-merge module, seeing how its structure very closely mirrors the structure of the rotate module.

4.2.3.The difference between the higher and lower coupling classes

To conclude the above discussion about the classes in the rotate and merge modules and their respective coupling values, we believe that this metric as calculated by DesigniteJava is a valuable instrument in discerning highly coupled classes within a given project/module.

To point out differences between highly coupled and less-coupled classes here would involve looking at the designated utility of the class under observation and noting carefully how it interacts with other classes(importing packages, instantiating other classes). The classes which are designed to be components would have less fan-out and less coupling with other code. However classes such as RotateModule or MergeModule, which bring these components together will have a higher fan-out and consequently, higher coupling.