

Why Do We Need Visualization?

Research and practice have shown that the cost of performing maintenance activities highly depends on the underlying design quality of the software systems. In the past, several techniques have been developed for the detection of design problems as a means to support the improvement of design quality in software systems. However, most of these techniques lack the ability to communicate the detected problems to the developers in a comprehensible and effective way. This is one of the reasons justifying the slow and hesitant adoption of preventive maintenance (i.e., maintenance activities aiming to improve future maintainability) as a practice in the software industry.

In this poster, we demonstrate two code smell visualizations for the Feature Envy and God Class design problems. The visualizations have been integrated in JDeodorant Eclipse plug-in, a code smell detection and refactoring tool.

How Does it Work?

After installing JDeodorant from the website (see below), you must select the God Class or Feature Envy option from the "Bad Smells" tab. After selecting which class or package you wish to refactor, click on the "Identify Bad Smells" button. A list of possible refactoring suggestions will be displayed in a table at the bottom of your screen.

The "Code Smell Visualization" Tab opens when you double-click on one of the refactoring suggestions. This tab is easily moveable to anywhere you want in Eclipse. This allows you to view the visualization and the actual code at the same time which makes for a better understanding of what is happening. This is shown in the diagram below:

COMP249_Assignment3 COMP249_Assignment3 COMP249_Assignment4 Jrechart.10.14 Jrechart.10.14 Jrechart.3.3	 	Code Smell Visualization 🛙	Quick Access			
ackage Explorer 😢 " 🗆 COMP249_Assignment3 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 Diffeedrart-1.0.14 Distoraw5.3	⑦ UndoManager 23 ⑦ TextFigure.java ⑦ StandardSto ⑦ FloatingTex ≫ ₇₁ ⊕ /** Collection of undo activities */	Code Smell Visualization 🛙				
COMP249_Assignment3 COMP249_Assignment3_Task1 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment4 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment3_Task1 COMP249_Assignment4 COMP249_Assignment	<pre></pre>					
→ 3 JRE System Library [re7] → JRE System Library [re7] → build → CH → Got → drawings → gravaloc → JavaDrawApplet.html → javaDrawApplet.html → jgpl4cense.html → PertApplet.html → PertApplet.html → README.html → VideoStore	<pre>private Vector undoStack; private int maxStackCapacity; public UndoManager() { this(DEFAUT_BUFFE_SIZE); } public UndoManager(int newUndoStackSize) { maxStackCapacity = newUndoStackSize; undoStack = new Vector(maxStackCapacity); redoStack = new Vector(maxStackCapacity); redoStack = new Vector(maxStackCapacity); redoStack = new Vector(maxStackCapacity); redoStack = new Vector(maxStackCapacity); redoStack.removeElementAt(0);</pre>	C CH.ifa.draw.util.Undo C CH.ifa.draw.util.Undo C CH.ifa.draw.util.Undo C C C C C C C C C C C C C C C C C C	e Class e class			
	<pre>// a not undoable activity clears the tack because // the last activity does not correspond with the</pre>					
	1					
	👔 Problems @ Javadoc 🔞 Declaration 🔗 Search 📮 Console 💥 Feature Envy 💥 God Class 🕱					
	Refactoring Type Source Class/General Concept	Extractable Concept	Entity Placement Rate it			
	CH.ifa.draw.applet.DrawApplet	E.	0.9188596216677388			
	CH.ifa.draw.util.UndoManager	0.9188964776453354				
	CH.ifa.draw.contrib.DragNDropTool	0.9192688194928614				
	E CH.Ifa.draw.figures.TextFigure	0.9192771333057007				
	CH.ifa.draw.util.StandardStorageFormat	0.9192835307098478				
	CH.ifa.draw.util.FloatingTextField	0.919359110228281				
	CH.ifa.draw.util.StorableOutput	0.91942873788128				
	CH.ita.draw.contrib.MDI_DrawApplication	0.9194960202681055				
	CH.ifa.draw.upl.storable.nput CH.ifa.draw.samples.net NodeFigure		0.9195068009967509			
	Construction of the standard Observation (Sec.		0.0105205414040502			

To learn more about Jdeodorant and to see the Code Smell Visualization for yourself, please visit and download Jdeodorant at the following website :

http://www.jdeodorant.com/





www.PosterPresentations.con

Code Smell Visualization Kimberly Dextras-Romagnino, Nikolaos Tsantalis Department of Computer Science and Software Engineering

FEATURE ENVY

Feature Envy occurs when a method references another class through methods and fields more often than it references its own class.

Solution: Move the method to the class that it is most envious of, passing any parameters the new method requires

The Visualization

The Visualization shows the Source Class on the left which originally contains the method to be extracted (shown with a white background). On the right, is the **Target Class** where the method is going to be moved to. In the middle, is the (potentially) Moved Method.

The Moved Method has connections to all the methods and fields it references. The number on the connections indicates the number of times this method or field was accessed. There is a legend to indicate the different types of connections

How does it help?

If the Moved Method has the majority of its connections to the Target Class in comparison to the Source Class, then it is a good refactoring suggestion.

Examples

The following example illustrates an excellent refactoring suggestion: the Moved method makes no reference to anything in the Source Class but calls four different methods from the **Target Class**



The following example is also a good refactoring suggestion: The majority of the calls made by the Moved Method are to methods in the Target Class.



Solution: Extract all the methods and fields which are related to a specific functionality into a separate Class

The Visualization

contained all of the extracted methods and fields. On the right is the Extracted Class which now contains all of the extracted methods and fields. The extracted methods have connections to all the methods and fields they reference. The number on the connections indicates the number of times this method or field was accessed. There is a legend to indicate the different types of connections.

How does it help?

If the extracted methods have the majority of its connections to the Extracted Class in comparison to the Source Class, then it is a good refactoring suggestion.

The following example illustrates an excellent refactoring suggestion: the methods and fields in the Extracted Class are tightly connected and make no reference to any field or method in the Source Class.



The following example is also a good refactoring suggestion: The majority of the calls made by the extracted methods are to methods and fields in the **Extracted Class.**

GOD CLASS

God Class usually violates the single responsibility principle and controls a large number of objects implementing different functionalities.

The Visualization shows the Source Class on the left which originally

Examples





• Sele By using appear an
 G CH get load sho viet
 <
 ⓒ CH ◎ get ◎ load ◎ show ◎ view
র র নার ব্য
• High By movin outgoing
CH.if Create Create Create
A tooltig move th
G CH.
□ fEdit



ect Connections

g the checkbox, you can make the different types of connections nd disappear allowing for a clearer picture

UNIVERSITÉ

Concordia



hlight Connections

ng the mouse pointer over a method or field, you can highlight the connections from that specific entity



oltips

p displaying the full name of the method or field appears when you ne mouse pointer over it

	Extracted Class	
fa.draw.util.Floati	p fContainer	
/idget		
	createOverlay(java.awt.Co	~
	endQuedauQ	er, java.awt.Font)
	createOverlay(Java.awt.Contain	er, java.awt.Fontj