in Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Program Understanding and Analysis, 2005.

DependencyViewer – A Tool for Visualizing Package Design Quality Metrics

Michael Wilhelm
Computer Science Department
Saarland University
66041 Saarbrücken, Germany

Stephan Diehl
Computer Science Department
Catholic University Eichstätt
85072 Eichstätt, Germany
diehl@acm.org

Abstract

DependencyViewer *helps to control package dependencies to avoid degeneration of package designs. To this end it computes design quality metrics including stability and abstractness for each Java package and draws the package graph in a way, such that violations of various design principles are immediately visible. In addition* DependencyViewer *provides several features to inspect the packages at various levels of details including at the level of source code.*

1 Introduction

Existing tools that compute package level metrics either do not provide visualizations at all or use standard graph layout for the package graph and simply show the metrics as tables or annotations. In contrast DependencyViewer uses an extended hierarchical layout algorithm that uses the metrics, here abstractness or stability, to determine the graph layers. In the resulting layout backedges are highlighted (colored red[1]) as they indicate violations of design principles.

Packages can be expanded to show more detail on demand, i.e., the classes with their methods and attributes contained in each package. Both expansion of packages as well as the navigation within a package preserves the mental map of the graph, i.e., the overall layout of the graph. Inner edges are connected to ports at the border of the package, when the user selects other classes in a package or browses through the method list, the inner edges are updated while the outer graph remains unchanged.

[1]All screendumps in this paper have been converted into grey scale images. In these images the darker edges are the backedges. Color versions can be found at `http://www.eposoft.org/depview/figures`

2 Design Principles

Robert Martin suggested various principles and metrics [4] for package designs. DependencyViewer provides analyses and visualizations that help in particular to enforce the following of these principles[2]:

Dependency Inversion Principle: *Depend upon abstractions. Do not depend upon concretions.*

Acyclic Dependencies Principle: *The dependencies between packages must not form cycles.*

Stable Dependencies Principle: *Depend in the direction of increasing stability.*

Stable Abstraction Principle: *Stable packages should be abstract packages.*

3 Metrics

The above principles should not only guide the design of a system, but also be enforced during evolution of the system. To this end DependencyViewer computes package design quality metrics from the compiled source code by traversing Java class file directories (or JAR files), including:

Number of Classes and Interfaces The number of concrete C_c and abstract classes A_c (and interfaces) in a package is an indicator of its extensibility.

Afferent Couplings (C_a): The number of other packages that depend on classes within the package is an indicator of the packages responsibility.

Efferent Couplings (C_e): The number of other packages that the classes in the package depend upon is an indicator of the independence of the package.

[2]For a detailed and illustrated discussion of each of these principles see [3]

Abstractness ($A \in [0 \ldots 1]$): The ratio of the number of abstract classes (and interfaces) to the total number of classes. $A = 0$ indicates a completely concrete package and $A = 1$ a completely abstract package.

Instability ($In \in [0 \ldots 1]$): The ratio of efferent coupling to total coupling $In = \frac{C_e}{C_e + C_a}$. $In = 0$ indicates a completely stable package and $In = 1$ a completely instable package.

Distance from the Main Sequence ($Dn \in [0 \ldots 1]$): The perpendicular distance of a package from the idealized line $A + In = 1$ indicates the packages balance between abstractness and stability. Ideal packages are either completely abstract and stable or completely concrete and instable.

In Figure 1 the values of the above metrics are shown as colored columns[3] for the package `eclipse.swt.events`: It has 26 abstract classes or interfaces and 16 concrete classes, so its abstractness is $A = 0.6$. Furthermore it has about as many afferent as efferent couplings, such that its instability is $In = 0.4$. In total the class is quite balanced ($Dn = 0.0$).

Figure 2 summarizes the metrics computed for the packages of SWT. Figure 3 and Figure 4 show the package graph of SWT with respect to abstractness respectively instability. At first glance we see that there are many red backward edges when using abstractness and only a few when using stability to layer the package graph.

4 Implementation

DependencyViewer analyzes byte code stored in class files or JAR archives. Metrics are either computed by the internal analyzer or external ones and stored in the repository. The contents of the repository can be stored as XML reports and even reloaded from these. The package graph is built from the information of the repository and rendered by the client which allows the user to interactively explore the graph.

Our analyzer uses the BCEL API [2] to parse class files and produce an internal object-oriented representation of the symbolic information contained in the class files. Most metrics can be easily computed by traversing this internal representation.

Access to external analysis tools is done through wrapper classes that we call plugins. DependencyViewer has currently plugins for JDepend [1] and Dependency Finder [6].

Many graph drawing algorithms for hierarchical graphs follow the Sugiyama approach [5]. It works in four phases: assigning the nodes to layers, reducing edge crossings, computing absolute coordinates for the nodes and finally edge routing. We adapted in particular the first phase, such that it uses metrics instead of the connectivity of a graph to form layers. As some layers would become very large and to better fit the graph on the screen, we then compute sub-layers for each layer by a depth-first traversal of all nodes in the layer.

Packages and classes are shown as boxes similar to the UML notation for packages and classes. When a package is expanded, one of its classes is shown within the box (and others can be selected via a pull down menu or the scroll bar) and the dependencies of this particular classes are explicitly shown[4]. Thus additional edges have to be drawn. As the expansion does not change the order on the layers, the routing of the existing edges does only slightly change to accommodate the new edges. By using ports at the borders of the expanded packages it is possible to browse through the classes of the package without having to reroute the external edges. Only the inner edges (within the box of a package) which connect the ports to the attributes and methods of a class have to be redrawn.

DependencyViewer is freely available for download at `http://www.eposoft.org/depview`[5].

References

[1] M. Clark. JDepend Homepage. `http://www.clarkware.com/software/JDepend.html`.

[2] M. Dahm. Byte Code Engineering. `http://bcel.sourceforge.net/downloads/paper.pdf`.

[3] R. C. Martin. Design Principles and Design Patterns. `http://www.objectmentor.com/resources/articles/Principles_and_Patterns.PDF`, 2000.

[4] R. C. Martin. The Single Responsibility Principle. In *The Principles, Patterns, and Practices of Agile Software Development*, pages 149–154. Prentice Hall, 2002.

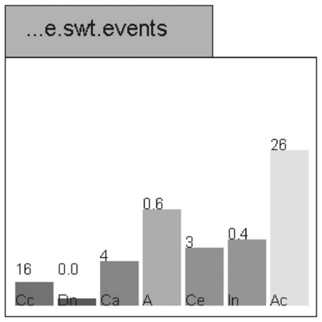
[5] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEE Transactions on Systems, Men, and Cybernetics*, 11(2):109–125, February 1981.

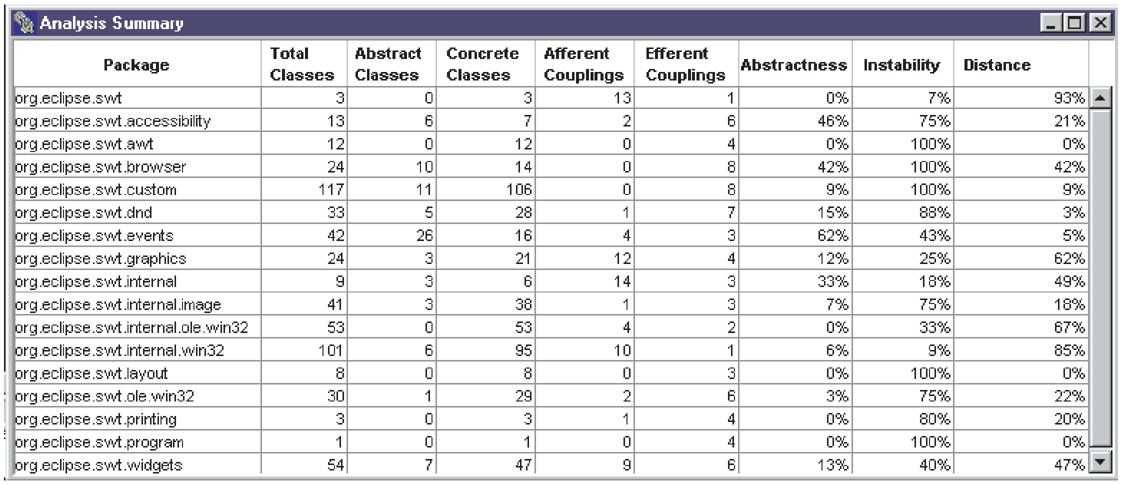
[6] J. Tessier. Dependency Finder Homepage. `http://depfind.sourceforge.net`.

[3]The value of the metrics are visually encoded both by the color (using a linear optimal color scale) as well as the height of the columns.

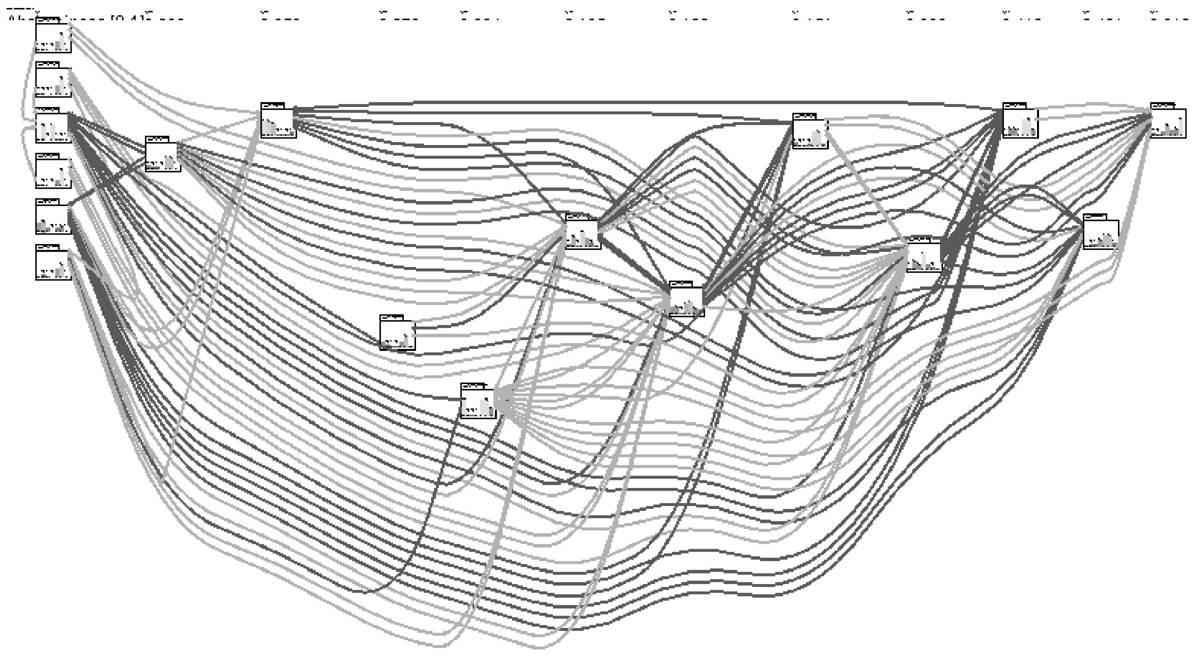
[4]By pressing the SHIFT key and clicking at a method or attribute only the dependencies for that method or attribute are shown.

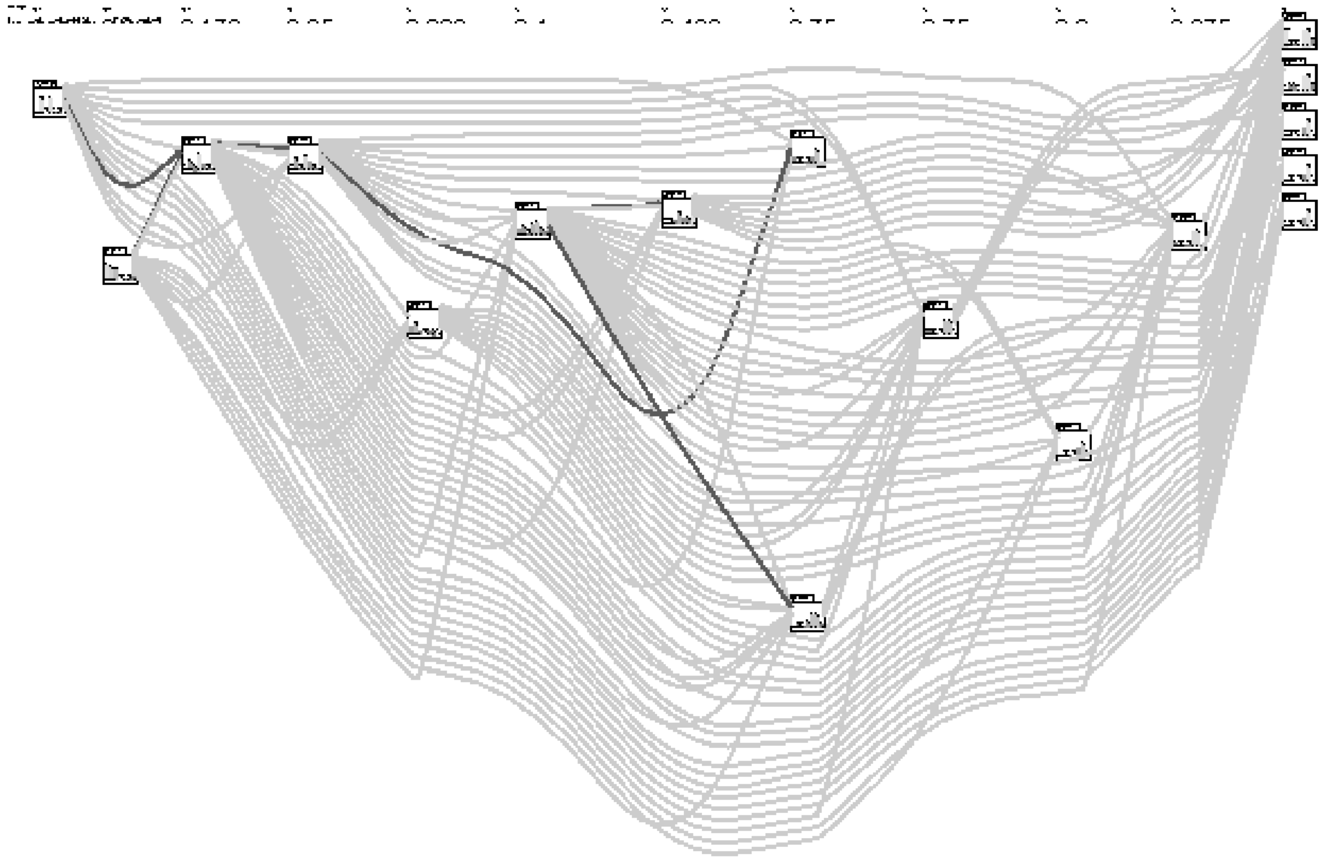
[5]A compressed XML report for SWT can also be downloaded from this site. The XML report allows to inspect the package design of SWT without having to download the Eclipse source code to compute the metrics.

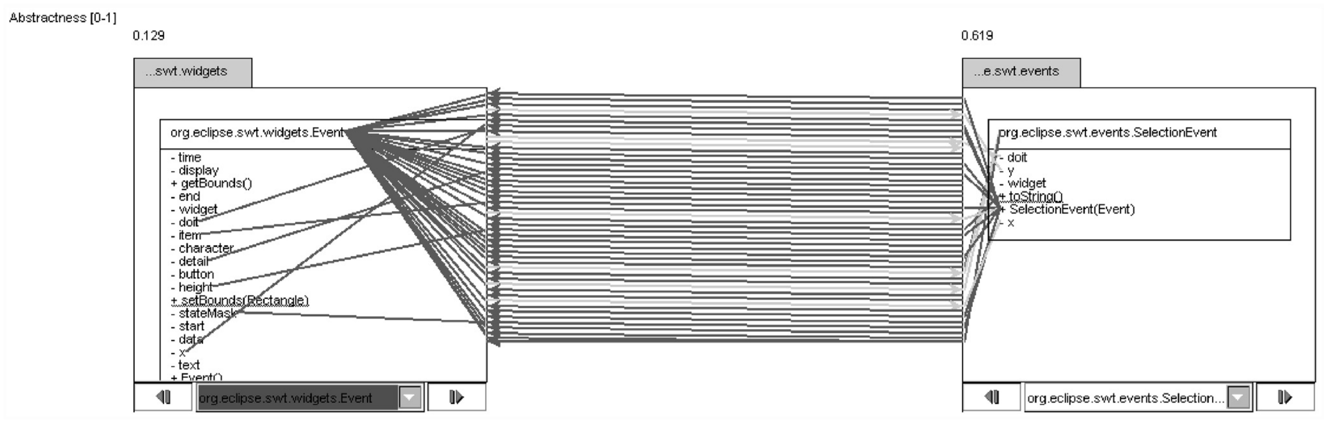




Package	Total Classes	Abstract Classes	Concrete Classes	Afferent Couplings	Efferent Couplings	Abstractness	Instability	Distance
org.eclipse.swt	3	0	3	13	1	0%	7%	93%
org.eclipse.swt.accessibility	13	6	7	2	6	46%	75%	21%
org.eclipse.swt.awt	12	0	12	0	4	0%	100%	0%
org.eclipse.swt.browser	24	10	14	0	8	42%	100%	42%
org.eclipse.swt.custom	117	11	106	0	8	9%	100%	9%
org.eclipse.swt.dnd	33	5	28	1	7	15%	88%	3%
org.eclipse.swt.events	42	26	16	4	3	62%	43%	5%
org.eclipse.swt.graphics	24	3	21	12	4	12%	25%	62%
org.eclipse.swt.internal	9	3	6	14	3	33%	18%	49%
org.eclipse.swt.internal.image	41	3	38	1	3	7%	75%	18%
org.eclipse.swt.internal.ole.win32	53	0	53	4	2	0%	33%	67%
org.eclipse.swt.internal.win32	101	6	95	10	1	6%	9%	85%
org.eclipse.swt.layout	8	0	8	0	3	0%	100%	0%
org.eclipse.swt.ole.win32	30	1	29	2	6	3%	75%	22%
org.eclipse.swt.printing	3	0	3	1	4	0%	80%	20%
org.eclipse.swt.program	1	0	1	0	4	0%	100%	0%
org.eclipse.swt.widgets	54	7	47	9	6	13%	40%	47%







Abstractness [0-1]

0.129

...swt.widgets

org.eclipse.swt.widgets.Event

- time
- display
+ getBounds()
- end
- widget
- doit
- item
- character
- detail
- button
- height
+ setBounds(Rectangle)
- stateMask
- start
- data
- x
- text
+ Event()

org.eclipse.swt.widgets.Event

0.619

...e.swt.events

org.eclipse.swt.events.SelectionEvent

- doit
- y
- widget
+ toString()
+ SelectionEvent(Event)
- x

org.eclipse.swt.events.Selection...