Exploring Evolutionary Coupling in Eclipse

Peter Weißgerber, Leo von Klenze, Michael Burch, Stephan Diehl
Computer Science Department
Catholic University Eichstätt
85072 Eichstätt, Germany
{michael.burch,peter.weissgerber}@ku-eichstaett.de,
research@leo-von-klenze.de,
diehl@acm.org

ABSTRACT

While software archives have been around for a long time, they have been mostly used to store and reconstruct versions of a software system and to coordinate simultaneous changes. Currently many researchers are investigating new ways to exploit the information stored in software archives. In this paper we present an Eclipse plugIn that visualizes evolutionary coupling between files, i.e., how likely it is that two files are changed together. The information is automatically extracted from the software archive and displayed as a pixelmap, where each pixel represents a single coupling. The plugIn nicely integrates with the rest of the Eclipse IDE and thus allows to interactively explore the evolutionary coupling between different files of a project.

Categories and Subject Descriptors

D.2.6 [**Software Engineering**]: Programming Environments; I.3.8 [**Computer Graphics**]: Applications

General Terms

Software archives, evolutionary coupling, co-change, software visualization, software evolution, Eclipse

1. INTRODUCTION

Larger software systems are typically developed using configuration management systems to keep track of the changes to the source code and allow different developers to work on the system at the same time. The resulting software archive consists of the subsequent versions of each file of the system.

Based on the information stored in software archives, we can compute the evolutionary coupling of files, as well as more fine-grained software artifacts like classes or methods. Two software artifacts are *evolutionary coupled*, if they have been changed at the same time. The basic assumption here is that the more frequently software artifacts have been changed together, the stronger they are coupled. This is in contrast to the classical notion of coupling, which is based on one artifact referencing the other.

In previous work we used evolutionary coupling to compare the actual architecture of a software system to its evolutionary one [6] and to recommend program changes to the programmer [7]. We also developed a system called EPOSee that provides various visualization techniques to visually explore evolutionary coupling data [3]. Currently we are integrating some of these visualizations into Eclipse. In the following we describe EPOSpix, our Eclipse plugIn that displays evolutionary coupling data as a pixelmap. In Section 2 the visualization technique itself and the interactive features of the plugIn are explained. Section 3 provides some implementation details and Section 4 discusses related work. Section 5 concludes the paper.

2. VISUALIZATION

After processing the software archive we have for every pair of software artifacts a, b the number of times they have been changed [1] together $S_{a,b}$. We call this number the support count of the coupling. We can now easily compute the conditional probability that a is changed under the condition that b was changed: $C_{a,b} = P(a|b) = \frac{S_{a,b}}{S_{b,b}}$. We call this probability the confidence that a change to b implies a change to a. The matrixes S and C have the software artifacts as their dimensions.

In the example in Figure 1 the artifacts are files. The example shows the numerical values of the support count matrix and in addition the confidence values by color coding. High confidence values are shown as red pixels, low values as blue pixels. White indicates that there is no coupling, i.e., the support count is zero. For example, the file `GUI.java` (respectively `Button.java`) have been changed 4 (respectively 10) times in total (values at the diagonal of the matrix). Three times the two files have been changed together. The confidence that `Button.java` is changed, whenever `GUI.java` is changed is 75%, whereas the inverse implication has only a confidence of 30%. In general, the strongest couplings are those that have both high confidence and high support counts.

[1] We only consider here changes between versions checked into the repository. Developers and projects heavily differ in their policies and frequencies when they check new versions into the repository.

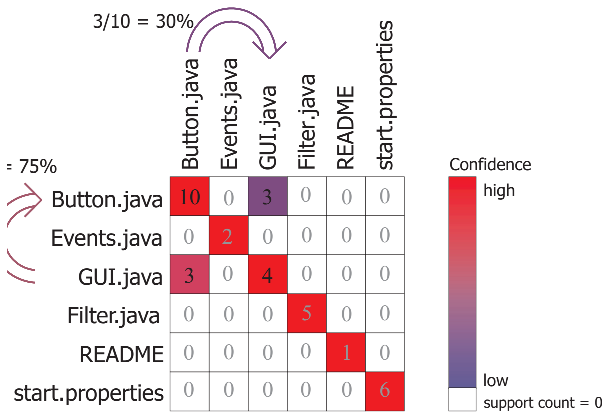


Figure 1: Pixelmap: Support, Confidence and Color Coding

Figure 3 shows a screenshot of our Eclipse plugIn called EPOSpix. It integrates the pixelmap visualization with the Eclipse IDE and thus allows to interactively explore the evolutionary coupling between different files of a project. Interactive features of EPOSpix currently include:

Zoom At the highest resolution each coupling is represented by a single pixel on the screen. The resolution can be reduced, such that squares of pixels represent a single coupling. As pixelmaps and in particular those at low resolution tend to become large, scrolling is supported.

Mouse-Over Hints When the user moves the mouse cursor over the pixelmap the following details about the coupling currently below the arrow head of the cursor are shown in the border at the bottom of the Eclipse IDE window: the full path of the file on the vertical axis (the antecedent of the implication), the full path of the file on the horizontal axes (the consequent of the implication), the support and the confidence of the coupling.

Crosshairs The crosshairs is displayed as a horizontal and and a vertical green line[2]. The coupling at the intersection of both lines is currently selected. Both files related to the coupling are highlighted in the package explorer. If the user presses the right mouse button, a pull-down menu appears which usually contains four file names: the names of antecedent and the consequent of the coupling the mouse pointer currently points at, as well as the names of the antecedent and consequent of the coupling which is currently selected by the crosshairs. If the user selects a file in this menu, the file is opened and shown in the editor. The crosshair can also be set in the inverse way, i.e., the user can select to files in the package-explorer or navigator view and the crosshair is put at the corresponding position in the pixelmap.

Options The color of the pixels can either indicate the support count or the confidence. It is also possible to export the pixelmap as graphics file. Furthermore, the user can filter the couplings to be shown, by setting a minimal support count or confidence value. Thus filtering is typically used to only show strong dependencies.

[2]If the corresponding file does not exist in the latest version of the project, this is indicated by a yellow line instead of a green line.

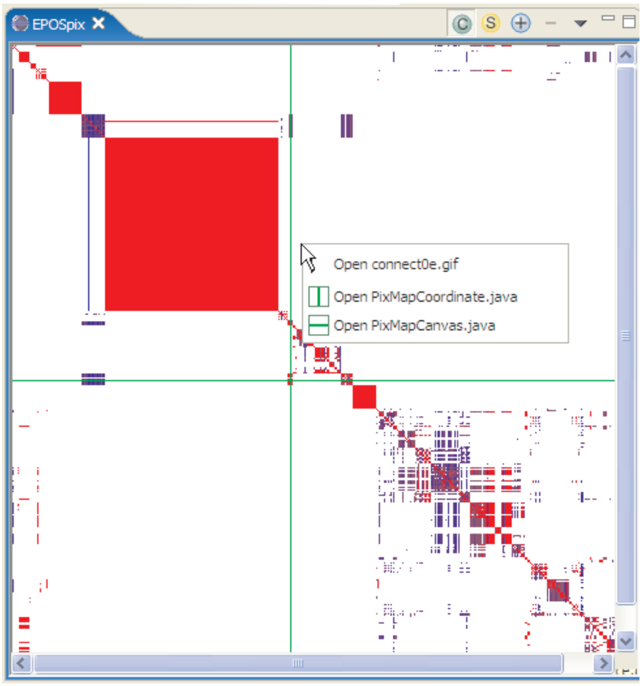


Figure 2: Pixelmap of the authors' software project

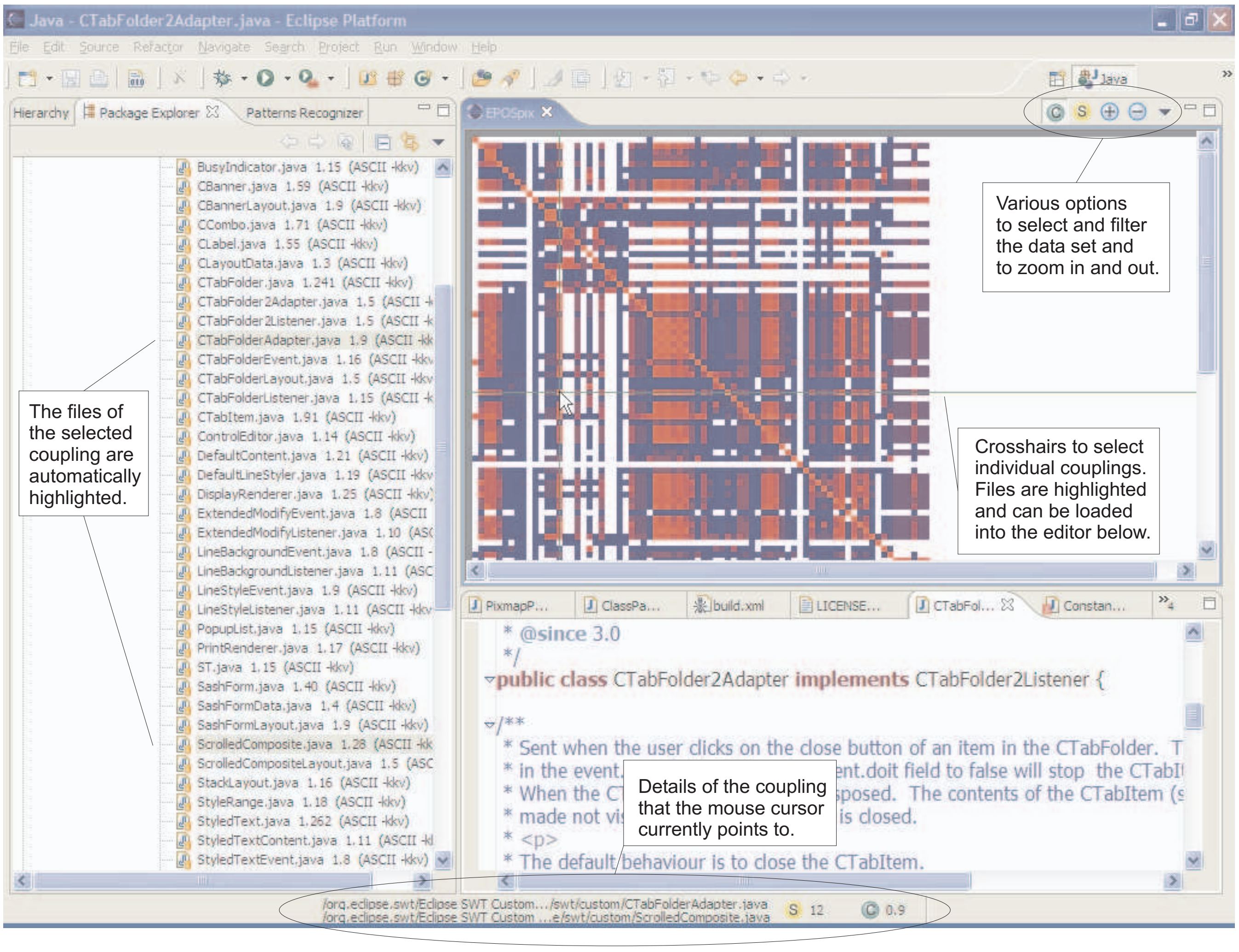
In the pixelmap the files along the axes are sorted lexically including their full path name. As a result files which are at the same hierarchy level in the package hierarchy are close together. As the package hierarchy typically reflects the architecture of the system, we expect that related files are in the same level of the hierarchy. Very closely related files even in the same folder. Furthermore, we expect that files which are related are also more often changed together than those that are unrelated. As we have shown in several case studies [6, 3], the fact that related files are close together in the pixelmap and are typically changed more often together becomes visible in the pixelmap as mostly red squares along its diagonal, as can be seen in Figure 2. In other word, files in the same directory are more frequently changed together than files from two different directories. More interesting are exceptions from this rule, i.e., files which are changed very frequently and do not correspond to the same hierarchy level. We call such exceptions *outliers*. If a software system has lots of outliers the developer should possibly think of restructuring the whole system.

In such a hierarchically sorted pixelmap with a blue-red (cold-hot) color scheme there are two important aspects:

The screenshot in Figure 3 shows the pixelmap of the widgets package of Eclipse SWT. The coupling selected with the crosshair is:

$\texttt{CTabFolderAdapter.java} \Longrightarrow \texttt{ScrolledComposite.java}$

It has a support count of 12 and a confidence of 90%. In


Java - CTabFolder2Adapter.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Java
Hierarchy
Package Explorer
Patterns Recognizer
EPOSpix
Various options to select and filter the data set and to zoom in and out.
The files of the selected coupling are automatically highlighted.
Crosshairs to select individual couplings. Files are highlighted and can be loaded into the editor below.
PixmapP...
ClassPa...
build.xml
LICENSE...
CTabFol...
Constan...
Details of the coupling that the mouse cursor currently points to.

```
BusyIndicator.java 1.15 (ASCII -kkv)
CBanner.java 1.59 (ASCII -kkv)
CBannerLayout.java 1.9 (ASCII -kkv)
CCombo.java 1.71 (ASCII -kkv)
CLabel.java 1.55 (ASCII -kkv)
CLayoutData.java 1.3 (ASCII -kkv)
CTabFolder.java 1.241 (ASCII -kkv)
CTabFolder2Adapter.java 1.5 (ASCII -k
CTabFolder2Listener.java 1.5 (ASCII -k
CTabFolderAdapter.java 1.9 (ASCII -kk
CTabFolderEvent.java 1.16 (ASCII -kkv
CTabFolderLayout.java 1.5 (ASCII -kkv
CTabFolderListener.java 1.15 (ASCII -k
CTabItem.java 1.91 (ASCII -kkv)
ControlEditor.java 1.14 (ASCII -kkv)
DefaultContent.java 1.21 (ASCII -kkv)
DefaultLineStyler.java 1.19 (ASCII -kkv
DisplayRenderer.java 1.25 (ASCII -kkv)
ExtendedModifyEvent.java 1.8 (ASCII
ExtendedModifyListener.java 1.10 (ASC
LineBackgroundEvent.java 1.8 (ASCII -
LineBackgroundListener.java 1.11 (ASC
LineStyleEvent.java 1.9 (ASCII -kkv)
LineStyleListener.java 1.11 (ASCII -kkv
PopupList.java 1.15 (ASCII -kkv)
PrintRenderer.java 1.17 (ASCII -kkv)
ST.java 1.15 (ASCII -kkv)
SashForm.java 1.40 (ASCII -kkv)
SashFormData.java 1.4 (ASCII -kkv)
SashFormLayout.java 1.9 (ASCII -kkv)
ScrolledComposite.java 1.28 (ASCII -kk
ScrolledCompositeLayout.java 1.5 (ASC
StackLayout.java 1.16 (ASCII -kkv)
StyleRange.java 1.18 (ASCII -kkv)
StyledText.java 1.262 (ASCII -kkv)
StyledTextContent.java 1.11 (ASCII -kl
StyledTextEvent.java 1.8 (ASCII -kkv)
```

```
 * @since 3.0
 */
public class CTabFolder2Adapter implements CTabFolder2Listener {

/**
 * Sent when the user clicks on the close button of an item in the CTabFolder.  T
 * in the event.                     ent.doit field to false will stop  the CTabI
 * When the C                        sposed.  The contents of the CTabItem (s
 * made not vi                       is closed.
 * <p>
 * The default behaviour is to close the CTabItem.
```

```
/org.eclipse.swt/Eclipse SWT Custom.../swt/custom/CTabFolderAdapter.java     S 12    C 0.9
/org.eclipse.swt/Eclipse SWT Custom ...e/swt/custom/ScrolledComposite.java
```

Figure 3: Eclipse PlugIn: Pixelmap View

other words, this means that both files have been changed 12 times together and there is a probability of 90 percent that when the file `CTabFolderAdapter.java` is changed also the file `ScrolledComposite.java` has to be changed.

3. IMPLEMENTATION

Integrating the pixelmap visualization into Eclipse offered several advantages, above all the possibility to select files in the visualization and highlight them in the package explorer or load them into the editor.

The plugIn was implemented as a *view*, i.e., a window within a *perspective* in Eclipse. The view always shows the pixelmap of the project currently selected in the package-explorer or navigator view.

3.1 Implemented Interfaces

View inherit from the class `org.eclipse.ui.views.ViewPart` and overwrite methods to interact with the IDE and implement certain interfaces. EPOSpix implements the interface `ISelectionListener` and `IJobChangeListener`. The former listens to changes in the package explorer, the latter to those of the navigator. These listeners allow EPOSpix to put the crosshair at the corresponding position when two files are selected in those views.

When the user presses the right button, a menu pops up and the user can select one of several files (as described above). The file is then opend in in the editor by calling the Eclipse API method `openEditor`.

3.2 Implementation of the Visualization

The visualization itself was originally implemented with AWT. Instead of porting it, we decided to reimplement it in SWT, because we were not quite satisfied with some of the design decisions of our original implementation:

Drawing the Pixelmap The most important task in the implementation is to draw the pixelmap. This is done in the class `PixmapCanvas` which extends the SWT class `Canvas`. `PixmapCanvas` stores all the couplings in the pixelmap, the chosen color scale, the zoom factor, the minimal support cound and confidence, as well as the metric (support or confidence) that should be displayed.

The actual drawing works as follows: First, the complete pixelmap is painted into an Image object which has the same dimensions as the pixelmap, multiplied with square of the zoom factor. This image is initialized with the background color that represents zero in the selected color scale. Then, we iterate over all binary couplings with a support count respectively confidence larger than the minimal support count respectively the minimal confidence. For each such coupling the appropriate pixel (or set of pixels, if the zoom factor is greater than 1) in the image is drawn in the adequate color depending on the selected metric and the selected color scale.

The PaintListener of the class PixmapCanvas just draws the data of this image into the graphics context. The drawback of this technique is that the complete image has to be redrawn if the zoom factor, the selected metric, or the selected color scale changes. The big advantage, however, is that the redrawing process if the window with the pixelmap is moved, raised, or resized, is very fast. We assume that these operations occur much more frequent than zoom factor, metrics, and color scale changes.

Scrolling Pixelmaps can become quite large, often a lot larger as the space that is available on the screen, and zooming even increases this problem. Thus, our implementation must provide facilities to scroll the pixelmap on the screen. For this, a PixmapCanvas object is embedded in an instance of the SWT object ScrolledComposite. This object automatically takes care that its content can be scrolled along both axes, if it is larger than the available space on the screen.

Altogether, we found that it did not take much effort to implement our visualization using SWT and integrate it into the Eclipse IDE.

3.3 Accessing Coupling Data

To compute the evolutionary coupling data we use the eROSE plugIn for Eclipse [4], the Eclipse plugIn version of our recommendation system ROSE [7]. It stores all data (support count, confidence) in a database (which can be an internal as well as an external one) and provides several packages to access this data. In essence, we only need to perform an SQL query to retrieve all necessary data from that database.

4. RELATED WORK

The fact, that software archives contain a lot of so far unused information was already noted by Ball et al. [1].

Several authors use the term co-change relation instead of evolutionary coupling. The paper by Gall et al. is probably the first work which relates this kind of information to the architecture of a system [5].

Instead of using pixelmaps the coupling data can also be visualized using graph drawing techniques [2, 3]. In addition to visualizing binary relations, our tool EPOSee can also visualize N-ary relations and temporal relations between software artifacts, i.e., that some artifacts usually have been changed before others.

5. CONCLUSIONS

EPOSpix is the first of our visualizations that we integrated into Eclipse. We found that this integration allows us to better explore software archives, as we can easily access software archives and once we get the visualization, the files related by a coupling can be inspected using the many features of Eclipse.

In the future we intend to also provide several other visualizations of the EPOSee tool as Eclipse plugIns.

6. REFERENCES

[1] T. Ball, J.-M. Kim, A. A. Porter, and H. P. Siy. If your version control system could talk.... In *ICSE Workshop on Process Modelling and Empirical Studies of Software Engineering*, 1997.

[2] D. Beyer and A. Noack. Clustering software artifacts based on frequent common changes. In *Proceedings of the 13th IEEE International Workshop on Program Comprehension (IWPC 2005)*, pages 259–268, Los Alamitos, CA, 2005. IEEE Computer Society Press.

[3] M. Burch, S. Diehl, and P. Weißgerber. Visual Data Mining In Software Archives. In *Proc. ACM Symposium On Software Visualization (SOFTVIS)*, St. Louis, Missouri, U.S., 2005.

[4] eROSE — Recommender Plugin for Eclipse. http://www.st.cs.uni-sb.de/softevo/erose/, 2005.

[5] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proc. International Conference on Software Maintenance (ICSM '98)*, pages 190–198, Washington D.C., USA, Nov. 1998. IEEE.

[6] T. Zimmermann, S. Diehl, and A. Zeller. How history justifies system architecture (or not). In *Proceedings of International Workshop on Principles of Software Evolution IWPSE'2003, Helsinki, Finland*, pages 73–83. IEEE Computer Society Press, 2003.

[7] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.