in Proceedings of ACM Symposium on Software Visualization SOFTVIS'05, St. Louis, May 2005.

Visual Data Mining in Software Archives

Michael Burch
michael.burch@ku-eichstaett.de

Stephan Diehl
diehl@acm.org

Peter Weißgerber
peter.weissgerber@ku-eichstaett.de

Catholic University Eichstätt-Ingolstadt
Ostenstr. 14, 85072 Eichstätt,Germany

Abstract

Software archives contain historical information about the development process of a software system. Using data mining techniques rules can be extracted from these archives. In this paper we discuss how standard visualization techniques can be applied to interactively explore these rules. To this end we extended the standard visualization techniques for association rules and sequence rules to also show the hierarchical order of items. Clusters and outliers in the resulting visualizations provide interesting insights into the relation between the temporal development of a system and its static structure. As an example we look at the large software archive of the MOZILLA *open source project. Finally we discuss what kind of regularities and anomalies we found and how these can then be leveraged to support software engineers.*

1. Introduction

During the life time of a software system many versions will be produced. Analyzing the source code of these versions, as well as documentation and other meta-information can reveal regularities and anomalies in the development process of the system at hand [7].

Some very general rules have been suggested based on some case studies by Lehman, basically stating that the size, functionality, and complexity of a software system increase over time, while its growth rate and quality decrease [10]. For many of the rules published in software engineering literature [6] the relevance with respect to other projects is unclear: either the rules are too general, or results of the case studies cannot be transferred, because the constraints of the case studies are not well documented. To remedy this situation the development of tools for validating rules based on the history of ones own project or even discovering new project-specific rules is an active area of research. In this paper we use visual data mining to this end.

Industrial, as well as open source projects keep track of versions and changes using configuration management systems [4] like RCS and CVS. Other tools keep track of additional information, e.g. bug databases or electronic mail. The information stored by a configuration management system and related tools is called a software archive. The software archive provides the history of a software system.

In previous work we have used data mining to extract association rules from such archives to characterize the development process [17] or to support programmers [18]. We also introduced the term *evolutionary coupling*, which is based on the simultaneous changes of files rather than on one referencing the other.

In this paper we discuss the visualization techniques that we implemented to analyze both association as well as sequence rules. Association rules suggest further changes that should be performed for a set of given changes. Sequence rules additionally indicate the order of these changes. By means of examples we show the kinds of insights that can be gained about the evolution of a software system by visualizing these rules.

The remainder of this paper is organized as follows: Section 2 describes the features of our visualization tool EPOSee that integrates various visualization techniques. In Section 3 we introduce hierarchical items. Sections 4, 5 and 6 describe visualization techniques for different kinds of rules with hierarchical items. A case study is presented in Section 7. Finally, we discuss related work in Section 8 and Section 9 concludes this paper.

2. An Integrated Visualization Tool

To interactively explore the mining rules extracted from software archives we developed EPOSee [1] (see Figure 3) which provides the following visualizations:

- Visualization of Binary Association Rules
  - Pixelmap (overview, context)
  - Support Graph (force-directed, polar layout)

[1] Evolution Patterns of Software

  - 3D Bar Chart (of selected association rules, focus)
  - Rule Detail Window (of selected rule, focus)
- Visualization of n-ary Association Rules
  - Association Rule Matrix (overview, context)
  - Bar Charts (for support and confidence)
  - Rule Detail Window (of selected rule, focus)
  - Item Legend Window
- Visualization of Sequence Rules
  - Parallel Coordinates View (overview, context)
  - Decision Tree (overview, context)
  - 3D Branch View (of selected sequence rules, focus)
  - Rule Detail Window (of selected rule, focus)
- Histogram (distribution of confidence and support)

Typically the visual data mining process works as follows: first an overview of the rule set is provided, here interesting visual patterns like clusters can be detected. Next the user can inspect the rules of a visual pattern by selecting the rules involved and viewing them in a zoomed display (for example in the 3D view). Additionally, filters allow her to constrain the set of rules that are shown at all. Last but not least, she can select single rules which are shown in detail then.

This conforms to Ben Sheiderman's visualization mantra: "Overview first, zoom and filter, then details on demand" [12].

In the following we will explain how EPOSee integrates the different views for binary association rules, n-ary association rules and sequence rules and how the visualizations interact with each other.

Figure 3(a) shows our tool in the binary association rule mode. There are three windows (which can be resized and moved according to the user's need) that show the pixelmap, the support graph and the 3D bar chart view for a set of rules. At the bottom of the window the selected rule is shown in textual detail. When a user selects an interesting part of the pixelmap, this part is zoomed in the 3D bar chart view. Next, the user can select a single rule from the bar chart view which will be shown in detail. An advantage of EPOSee over single applications for each view is that the different views are internally connected, e.g. if a rule is selected in the 3D bar chart view, the corresponding items are emphasized in the support graph.

When displaying n-ary association rules EPOSee opens a window containing the rule matrix and a second window that shows the list of items involved. The user can select a single rule in the matrix which will be shown in the detail view at the bottom of the application window. The n-ary association rule mode of our application is shown in Figure 1. The histogram in the bottom right corner shows the distribution of support respectively confidence over the rules and can be displayed for all kinds of rules.

If the user opens a file containing sequence rules EPOSee enters the sequence rule mode which is illustrated in Figure 2. In this mode EPOSee displays windows that show the parallel coordinates view, the decision tree, as well as the 3D Branch view. The parallel coordinates view and the decision tree give the user an overview of the set of rules. Additionally, the selected rule is zoomed in the 3D branch view and shown in detail in an additional window at the bottom of the application.

Independent of the kind of rules, EPOSee allows to filter the set of rules according to their support and confidence, and to search for keywords (see Figure 3(c)). Moreover, various schemes for color-coding can be used as shown in Figure 3(b). Thus EPOSee supports visual data mining on data mining results, i.e. large sets of association and sequence rules. In the sequel we discuss each of the different kinds of rules and their visualizations in more detail.

3. Hierarchical Items

The items in the rules extracted from software archives are software artifacts like files, classes, methods or functions. Instead of starting from a random order when visualizing these items in a rule, we will use a total order derived from a hierarchy stemming from the application domain, e.g. methods are contained in classes, classes are contained in files, files are contained in directories, and directories are contained in other directories.

4. Visualizing Binary Association Rules

To detect relations between items we first look at how often two items have been changed together — that is, how often have they been checked into the software archive at the same time, i.e. in the same transaction. The *support* $\mathsf{supp}(S)$ of a set of items S is the number of transactions containing the set S.

The support $\mathsf{supp}(\{i,j\})$ of two items i and j indicates how much evidence is there for their dependency. In particular, $\mathsf{supp}(\{i\})$ is the total number of times item i was changed.

Next we compute the strength of the dependencies, i.e. the number of changes of a pair of items relative to the number of changes of a single item. As a result we get the *confidence*:

$$\mathsf{conf}(i \Rightarrow j) = \frac{\mathsf{supp}(\{i,j\})}{\mathsf{supp}(\{i\})}$$

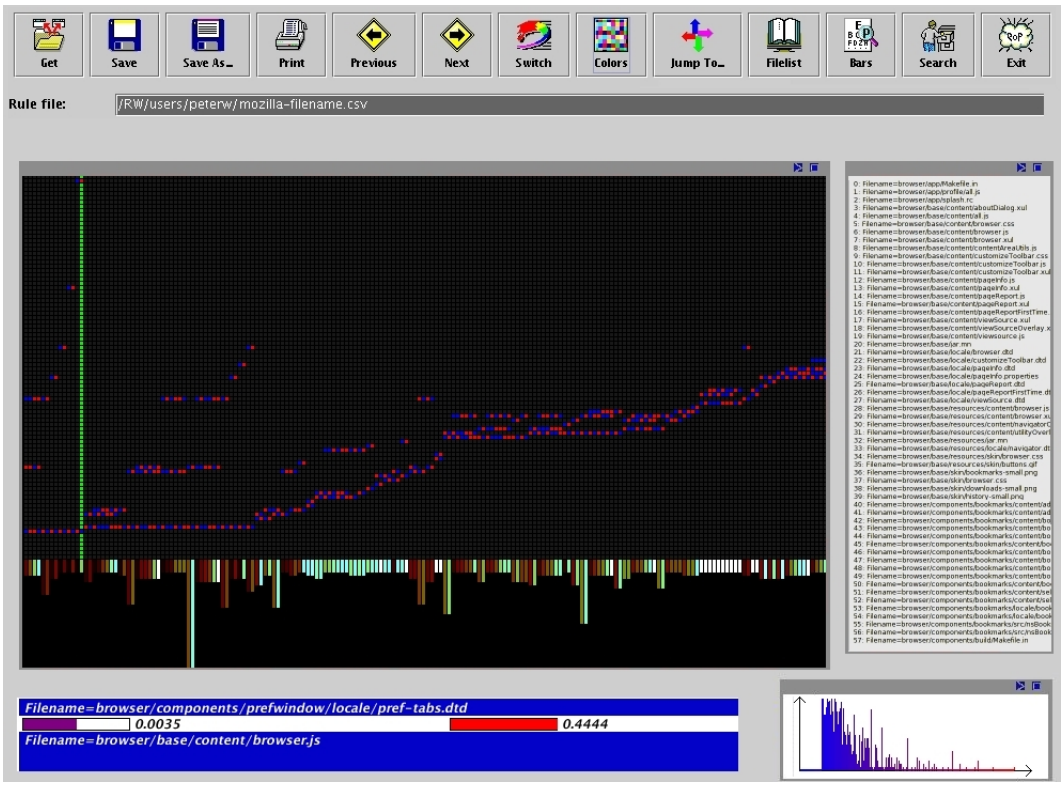


Figure 1. Association Rules (n-ary)

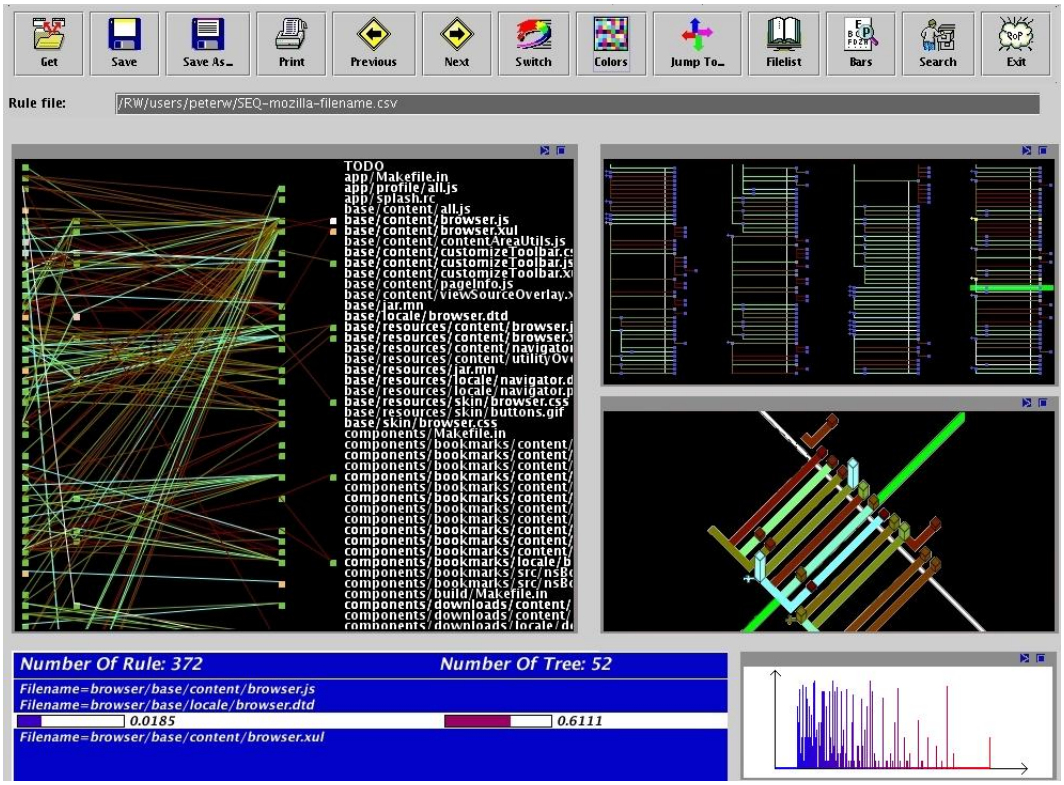
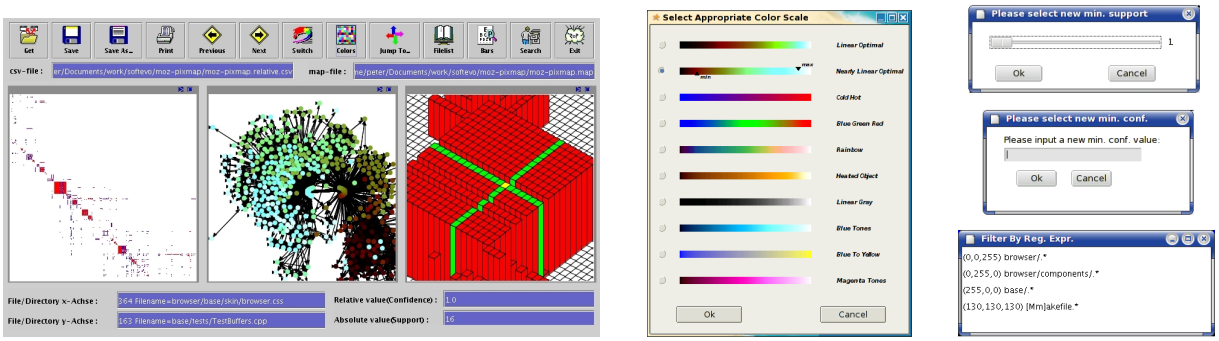


Figure 2. Sequence Rules



(a) Binary Association Rules (b) Color Scale Selection (c) Filter Dialogs

Figure 3. Interactive tool to visually explore different kinds of rules.

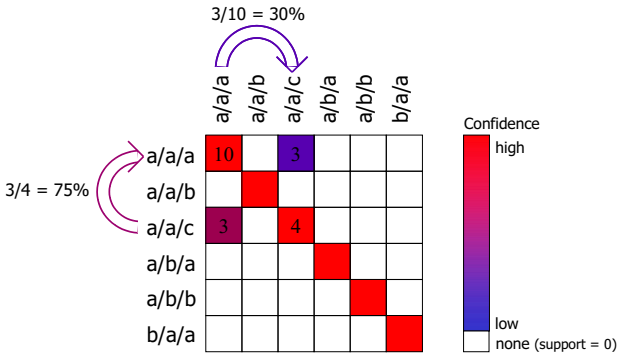


Figure 4. Pixelmap with sorted items and color-coded confidence (numbers indicate support)

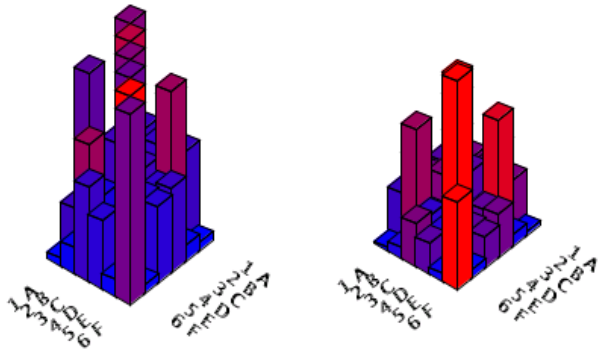


Figure 5. Dependency strength between items. *Left bar chart:* **Greater height indicates higher confidence, red color indicates high support, blue color indicates low support.** *Right bar chart:* **Height indicates support, color indicates confidence.**

The example in Figure 4 indicates that a change to `a/a/c`[2] implies a change to `a/a/a` with a probability of 75%, whereas the inverse implication has only a probability of 30%.

As we sort the items hierarchically, both sets are often the same and correspond to some level in the hierarchy, e.g. a directory.

The strongest dependencies are those that have both high confidence and high support. To find these dependencies, we use 3D bar charts to emphasize strong dependencies, see Figure 5. Here color is used to encode one value, while the third dimension (height) is used to encode a second value.

In addition to the pixelmap and the bar chart, we can gain insight into the dependencies between the items by drawing the support graph, see middle of Figure 3(a). In this graph related items are connected by edges and sets of items with many interrelations form clusters. Using color to indicate neighborhood with respect to the hierarchical sorting, we can also see outliers, i.e. nodes with different colors in clusters of nodes with mostly the same color.

[2]The notation `x/y/z` means that item z is located in the hierarchy y which is a sub-hierarchy of x.

5. Visualizing *n*-ary Association Rules

Binary association rules describe correlations between exactly two items. *N*-ary association rules improve this by taking more items into account: The rule $A \Rightarrow B$ where A and B are disjoint sets containing at least one item each implies that in cases when all items in A have been altered, all items in B have been changed, too. We call A the antecedent and B the consequent of the rule.

To assess the evidence and strength of an association, we use again the two measures *support* and *confidence*. Here, the support of an n-ary association rule $A \Rightarrow B$ is the support of the set $A \cup B$. The confidence of such an association rule

is the probability that all items in B are changed when all items in A are changed and therefore defined as

$$\mathsf{conf}(A \Rightarrow B) = \frac{\mathsf{supp}(A \cup B)}{\mathsf{supp}(A)}$$

In contrast to binary association rules, this general kind of association rules cannot be visualized in a reasonable way using the pixelmap technique. We would have to annotate sets instead of items on the axes but there is no total order on these sets that allows us to sort them hierarchically: For example, it is unclear if the set $\{a/ab, z/ab\}$ has to be arranged next to a/a or z/a.

Instead, we use another visualization technique. The association rule matrix [16] is a two-dimensional view which arranges the items in their hierarchical order on the y-axis and the rules arbitrarily ordered on the x-axis, see Figure 9.

There are three kinds of pixels in this view:

Red pixels If an item is contained in the antecedent of the rule, the association rule matrix has a red pixel at the corresponding position. Multiple adjacent red pixels in the same column appear if the antecedent contains multiple items which belong to the same hierarchical level, e.g. the same directory.

Blue pixels Analogous, blue pixels indicate that the corresponding item is contained in the consequent of the corresponding rule. Therefore, adjacent blue pixels in a column mean that the rule suggests to change multiple items of the same hierarchical level.

White pixels White pixels indicate that an item is not affected by a rule.

Furthermore, below each association rule there is a bar that indicates the support of the rule by the length and the confidence by the color. A textual representation of the currently selected rule is shown at the bottom line of this view.

6. Visualizing Sequence Rules

The kind of association rules that we discussed above provides insight into what items are related because of the fact that they have been changed at the same time. In addition, we would like to know in what temporal order changes typically occur. For example, there can be a rule claiming "if method `print()` and method `show()` of file `Account.java` have been changed in a row, then later the documentation `GUI.tex` has been changed, too".

To this end we compute and visualize sequence rules. Both the antecedent and the consequent of a sequence rule are sequences of items. This gives the antecedent and consequent a time component. For example the sequence rule $a_1 \rightarrow a_2 \rightarrow a_3 \Rightarrow b_1 \rightarrow b_2$ means that if a_1 is changed before or at the same time as a_2 and a_2 before or at the same time as a_3, then it is likely that some time later b_1 and simultaneously or later b_2 will be changed.

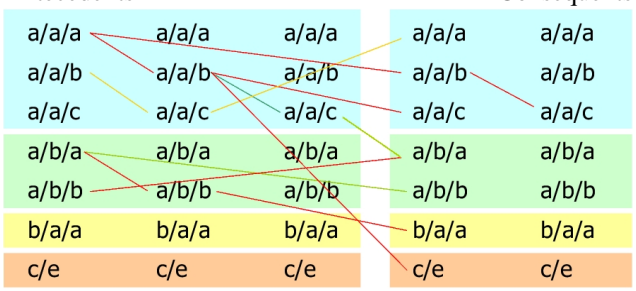


Figure 6. Parallel Coordinates View with sorted items

Similar to association rules, we characterize the evidence and strength of a sequence rule using the two measures support and confidence. A sequence p is a subsequence of another sequence q, if one can derive p from q by deleting elements from q. Several subsequent transactions into the software archive can be combined into transaction sequences. The support $\mathsf{supp}(s)$ of a sequence s is the number of transaction sequences it is a subsequence of. The confidence of a sequence rule $s_1 \Rightarrow s_2$ is then defined as

$$\mathsf{conf}(s_1 \Rightarrow s_2) = \frac{\mathsf{supp}(s_1 \cup s_2)}{\mathsf{supp}(s_1)}$$

Figure 6 shows a parallel coordinates view [8] of sequence rules. In this view every sequence rule is displayed by connecting the node in the n-th column representing the n-th item in the sequence with the node in the $n+1$-th column representing the $n+1$-th item. The items are sorted hierarchically and we can see clusters, i.e. all rules only contain items of the same directory. There is only one exception, the rule `a/a/a` $\rightarrow$ `a/a/b` $\Rightarrow$ `c/e` indicates that when `a` and `b` have been changed in directory `a/a` then also file `e` in directory `c` has to be changed. We will discuss this case in more detail later in Section 7 when we look at a real software archive.

In contrast to the parallel coordinates view in which one edge can belong to multiple rules, the decision tree visualization (see upper right corner of Figure 2) allows to have a closer look at the *single* rules. Due to the color coding it is easily possible to find strong rules. Furthermore, one can see the structure of the rules, e.g. the length of the antecedents and consequents of the rule set, or the number of consequents for one given antecedent.

7. Case Study: MOZILLA

MOZILLA is the code name of an Internet client that

was originally developed by Netscape Communications Corporation and has later turned into an open source project [13]. Its archive contains more than 77,000 files; consequently, we obtain a pixelmap with about 77,000 × 77,000 binary association rules. Counting functions rather than files would result in even larger maps.

For the following case study we used data mining to extract rules from the MOZILLA CVS archive and applied the different visualization techniques described above. In particular we looked for outliers. As we did not have prior knowledge of the internals of the MOZILLA project, we had to look into the files involved as well as the MOZILLA documentation to be able to explain these outliers.

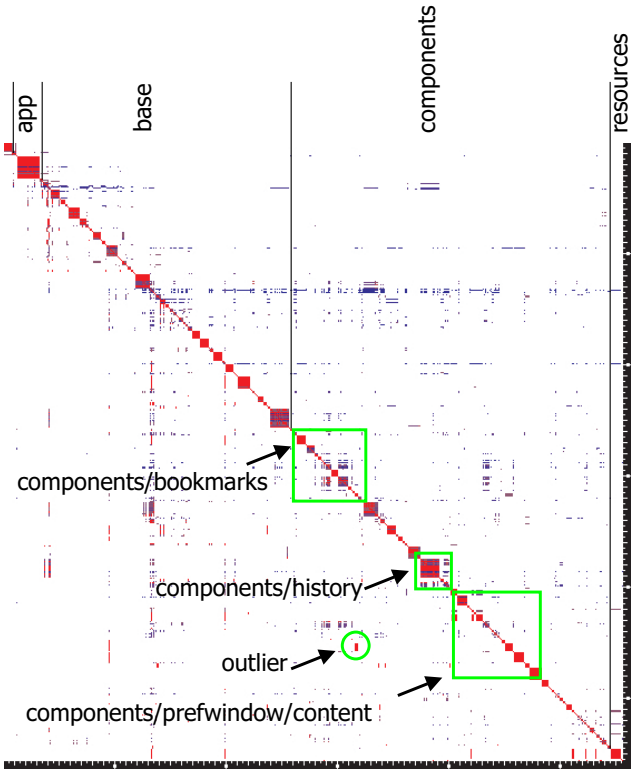


Figure 7. Pixelmap of the confidence matrix of MOZILLA

The pixelmap in Figure 7 shows the associations of the files in the `/browser` subdirectory of the CVS software archive of the MOZILLA project, which contains the web browser called FIREFOX. As the files are ordered hierarchically one can see that files which are next to each other, i.e. those that are in the same part of the hierarchy, are stronger related than others. Thus clusters typically extend along the diagonal of the pixelmap. These clusters very much correspond to the hierarchical structure of the system.

In other words, if in the pixelmap we do not find rectangular areas nicely aligned along the diagonal, then it might be a good idea to restructure the system, so that later on related changes are restricted to a certain subdirectory.

Software developers are mainly interested in outliers. These are those pixels representing couplings between files in different directories. Outliers can be a sign of aspects orthogonal to the system hierarchy, but also a sign of a bad system architecture.

One outlier is highlighted by a circle in the pixelmap in Figure 7. A closer inspection reveals that in directory `/components` the files

```
bookmarks/resources/locale/pref-bookmarks.dtd
bookmarks/skin/Bookmarks-toolbar.png
bookmarks/skin/bookmarksManager.css
```

are strongly related to the files

```
prefwindow/content/pref-popups.xul
prefwindow/content/pref-privacy.js
prefwindow/content/pref-privacy.xul
prefwindow/content/pref-proxies.js
prefwindow/content/pref-proxies.xul
prefwindow/content/pref-proxy-manual.xul
prefwindow/content/pref-scripts.js
```

The graphical user interface of this browser implementation is based on so-called XUL files. These are XML files which describe the elements of GUI and link them to actions which are JavaScript functions. These functions can then access all kinds of system functionality, e.g. COM objects, C++ code, etc. Furthermore, the appearance of the GUI elements can be customized using CSS files (Cascading Style Sheets) which for example contain references to the images to be used to display certain GUI elements. In the above example of an outlier, our first hypothesis was that the XUL files in `prefwindows/content` would directly or indirectly reference the DTD in `bookmarks/resources/locale/pref-bookmarks.dtd`. In this case the relation would have been easy to detect with a textual analysis of the files. But a closer look revealed that there is no reference from these XUL files to this DTD, but to local DTDs, e.g. the DTD `prefwindows/locale/pref-privacy.dtd`. These local DTDs are related to the first mentioned DTD because they follow the same naming conventions (e.g. names of attributes) and define to some extent the same tags. So if the naming conventions or tags change all DTDs have to be adapted. These kinds of relations cannot be uncovered with classical text or program analyses.

Note, that the visualization shows that there is an interesting correlation between these files. However, we had to examine manually *why* these files are related by looking into the corresponding source code. Thus, it would be desirable that the visualization would allow to look directly into the source of the displayed items. But this is not as straightforward as it seems: Every item exists in different revisions and it is unclear which of these are interesting to the user.

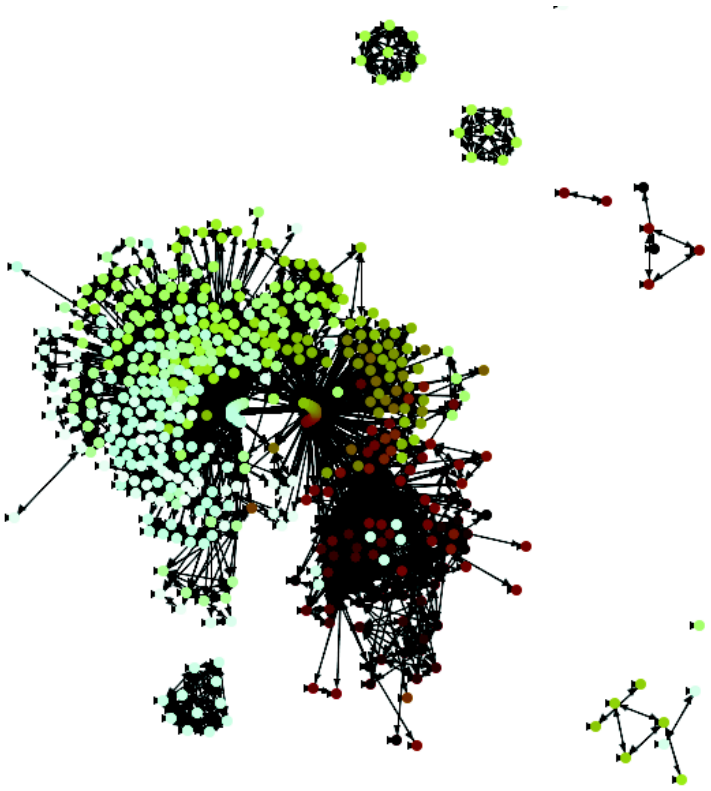


Figure 8. Support Graph of MOZILLA

In addition to the pixelmap we can also use the support graph to look for clusters and outliers. Figure 8 shows the support graph of the `/browser` directory of the MOZILLA project. Nodes represent items and are colored based on the hierarchical order of items. There is a large cluster in the middle of the drawing. The red part of this cluster corresponds to the `/base` subdirectory. In this red part are also a few light blue nodes. These outliers are the following files in directory `components/prefwindow/locale/`:

```
pref-advanced.dtd
pref-appearance.dtd
pref-applications-edit.dtd
pref-applications.dtd
```

A closer look reveals that only the file `pref-advanced.dtd` is related to files in the `/base` directory, while the remaining three DTDs are only related to the first one.

So far, we only used visualized binary rules, i.e. we only looked at dependencies between pairs of files. Additionally, we can look at simultaneous changes of possibly more than two files. Figure 9 shows the association rule matrix of the `/browser` directory of MOZILLA. As one can see easily, most of the n-ary association rules only involve items which belong to the same subdirectory.

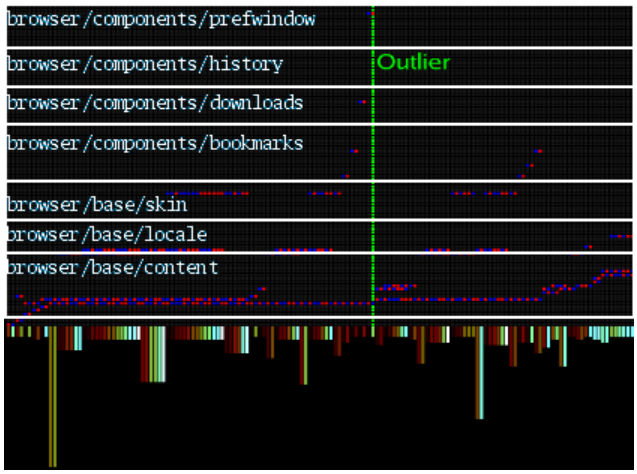


Figure 9. Association Rule Matrix of MOZILLA (only rules with a minimum support of 11)

But there are also several rules with items belonging to different subdirectories, most notably subdirectories of `browser/base` and `browser/components`. One of these outliers is marked in the picture: If `base/content/browser.js` has been changed, then `components/prefwindow/locale/pref-tabs.dtd` has been

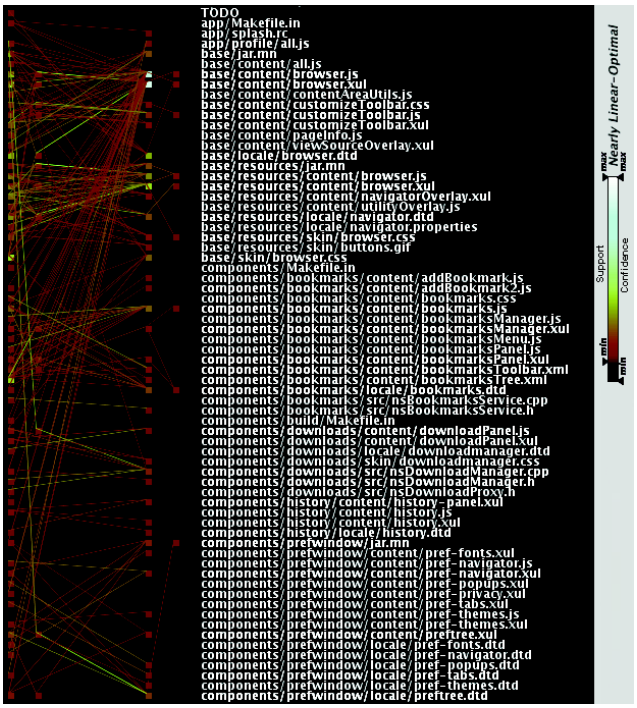


Figure 10. Parallel Coordinates View of MOZILLA (only rules with a minimum support of 11)

changed too. A remaining problem with the association rule matrix is that the rules on the x-axis have no special order. In Figure 9 we sorted the rules by the textual representation of the antecedent and consequent.

So far, we have looked at files that were often changed simultaneously. Next we look at the temporal order of changes. Figure 10 shows a parallel coordinates view of the `/browser` directory. The color of the nodes indicates the weighted sum of the support values of the prefixes of all rules which share this node, while the color of the edges indicates the weighted sum of the confidences. As the nodes are ordered with respect to the hierarchical order of the items, we see multiple clusters consisting of many edges which are only related to items in the same subdirectory. We also see that the files `base/content/browser.js` and `base/content/browser.xul` are related in a very interesting way to almost all Javascript respectively XUL files: they are typically changed after one of these other files has been changed.

Figure 11 shows a part of the parallel coordinates view in larger scale. There are two green edges leading to the light blue node representing the file `browser.xul` in the consequent column. The green edges indicate high confidence. But if we look at both items where these edges come from, we see that there are no edges with high confidence coming into these nodes. What does this mean? To inspect the rules we used the decision tree view of EPOSee and found that the rule `browser.js` ⇒ `browser.xul` has a confidence of 30%, while the rule `browser.js`→`browser.dtd` ⇒ `browser.xul` has a confidence of 61%. In other words, the confidence increases considerably as soon as the second file is changed.

8. Related Work

Data Mining Agrawal and Srikant introduced the *Apriori Algorithm* algorithm to compute *n*-ary association rules [1]. While the antecedents and consequents in their rules can be sets of items, we only consider single items to express associations between pairs of files for example.

Mining of sequence rules was described by Agrawal and Srikant [2]. However, in our work we mine in software change transactions rather than in customer transactions.

Rule Visualization There are several well-known approaches for visualizing association rules: The pixelmap technique is space-filling but only works good for association rules between exactly two items. In contrast, the visualization as directed graph supports rules between more than two items but needs a large screen.

Pak Chung Wong et al. improved the pixelmap technique to visualize *n*-ary rules with multiple items in their antecedent and/or consequent by aligning the rules on the x- and the items on the y-axis [16]. Later, they visualized sequence rules [15] using parallel coordinates as introduced by Inselberg and Dimsdale [8]. In contrast to our work, they did not deal with software evolution and did not explore relations based on hierarchically ordered items.

Visualizing Software Evolution There has been only little work on visualizing the evolution of software, so far. Tools like WinCVS [11] and VRCS [14] show the version graph in a vertical tree representation, which is sometimes called an explorer view. In three dimensional visualizations (see for example VRCS [9]) the Z axis can be used as a time axis and each vertical plane contains the versions checked in at the same time.

SeeSoft [5] introduced a space-filling visualization for metrics related to lines of code. The evolution of a system is shown as an animation of the changing values of these metrics.

The GEVOL system uses force-directed layout to draw call graphs, control-flow graphs and inheritance graphs of Java programs [3] and produces animations by showing graphs of subsequent program versions using linear interpolation for smooth transitions.

So far, case studies about the evolution of software systems are mostly using plots of metrics over time. In such plots either several metrics are shown for a single module

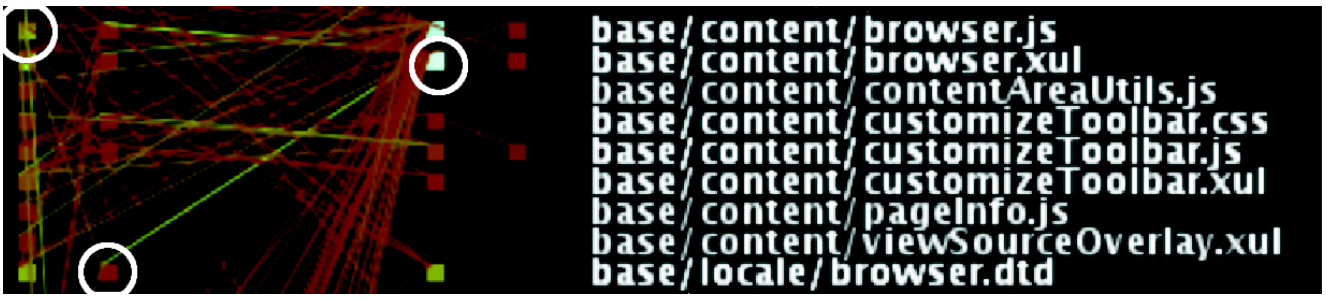


Figure 11. Enlarged Area of the Parallel Coordinates View of Mozilla

(or the whole system) or one metric is shown for a small number of modules.

9. Conclusions

Successful software changes and adapts to new situations. Configuration management systems keep track of these changes. Data mining and visual data mining techniques can help software engineers to extract rules from these archives and match them with their expectations and project rules. In this paper we presented extensions of standard visualization techniques for association and sequence rules taking the hierarchy of software artifacts into account and discussed some example visualizations computed from the software archive of the MOZILLA project.

In particular we identified certain kinds of visual patterns in the visualizations. Taking a closer look at the rules involved, we found that these patterns can be interpreted as follows and that this information could be further leveraged to support software engineers:

Clusters in Pixelmaps and Parallel Coordinates Views If clusters occur in these visualizations they are a sign that the chosen hierarchical decomposition of the system matches with the way the software system evolves, i.e. changes are mostly local. For pixelmaps some early results have been presented in [17].

Outliers in Pixelmaps An outlier indicates that items in different branches of the hierarchy are often changed together. This information can be used in two ways. First, when the programmer changes one of the items, the editor or configuration management system could suggest that he also should change the other item, this is what the ROSE prototype actually does [18]. Second, one could try to restructure the system, i.e. move the items to the same branch.

Outliers in Support Graphs The support graph shows which other items are related to the outlier and might have to be moved to the same branch as well.

Outliers in Association Rule Matrices Here sets of items are related to other sets of items. An outlier indicates that a large number of items in the same branch is related to one or a few items in a different branch. These few items might be moved to the other branch.

Outliers in Parallel Coordinates Views In the parallel coordinates view we found two different kinds of outliers. First, similar to the cases above, we have that items in different branches are changed and thus as before could be a reason to restructure the system. Second, we found that certain orders of the same changes are more likely than others, i.e. in the simplest case, that item a is often changed before item b, but b is never changed before a. One could easily extend the ROSE prototype to use this information to warn the programmer, if he does the changes in the *statistically* wrong order.

10 Acknowledgements

The work presented in this paper has been partially funded by the German Research Council (DFG) as part of the ”Evolution Patterns” project. The authors are grateful to all members of the project team, in particular Andreas Zeller and Thomas Zimmermann.

References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th Very Large Data Bases Conference (VLDB)*, pages 487–499. Morgan Kaufmann, 1994.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In Philip S. Yu and Arbee S. P. Chen, editors, *Eleventh International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, 1995. IEEE Computer Society Press.

[3] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A System for Graph-Based Visualization of the Evolution of Software. In

Proceedings of the ACM Symposium on Software Visualization, San Diego, USA, June 2003.

[4] Reidar Conradi and Bernhard Westfechtel. Version Models for Software Configuration Management. *ACM Computing Surveys*, 30(2), 1998.

[5] S.G. Eick, J.L. Steffen, and E.E. Summer. Seesoft - A Tool For Visualizing Line Oriented Software Statistics. In *Proc. of IEEE Transactions on Software Engineering*, pages 957–968, , 1992. IEEE Press.

[6] A. Endres and D. Rombach. *A Handbook of Software and Systems Engineering Empirical Observations, Laws and Theories*. Addison-Wesley, 2003.

[7] Ahmed E. Hassan, Richard C. Holt, and Audris Mockus, editors. *Proceedings of International Workshop on Mining Software Repositories MSR 2004, Edinburgh, Scotland, UK, May (collocated with ICSE'04)*, 2004.

[8] Alfred Inselberg and Bernhard Dimsdale. Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry. In *Proc. of Visualization '90*, pages 361–378, San Francisco, USA, 1990. IEEE Press.

[9] H. Koike and H-C Chu. VRCS: Integrating Version Control and Module Management using Interactive Three-Dimensional Graphics. In *Proceedings of IEEE Symposium on Visual Languages VL'97, Capri, Italy*, 1997.

[10] M.M. Lehman, D.E. Perry, J.F. Ramil, W.M. Turski, and P.D. Wernick. Metrics and Laws of Software Evolution-The Nineties View. In *Proceedings of Metrics 97 Symposium*, Albuquerque, NM, 1997.

[11] Alexandre Parenteau, Karl-Heinz Brünen, Jerzy Kaczorowski, and committers. CvsGui - A Set of GUI frontend for CVS. `http://www.wincvs.org`.

[12] Ben Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings of 1996 IEEE Conference on Visual Languages*, Boulder, CO, 1996. IEEE Press.

[13] The Mozilla Organization. Mozilla. `http://www.mozilla.org`.

[14] Walter F. Tichy. VRCE: The Visual Revision Control Engine. `http://www.aicas.com/vrce_en.html`.

[15] Pak Chung Wong, Wendy Cowley, Harlan Foote, Elizabeth Jurrus, and Jim Thomas. Visualizing sequential patterns for text mining. In *Proceedings of IEEE Information Visualization INFOVIS*. IEEE Computer Society Press, 2000.

[16] Pak Chung Wong, Paul Whitney, and Jim Thomas. Visualizing association rules for text mining. In *Proceedings of IEEE Information Visualization INFOVIS*. IEEE Computer Society Press, 1999.

[17] 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, September*, 2003.

[18] Thomas Zimmermann, Peter Weißgerber, Stephan Diehl, and Andreas Zeller. Mining version histories to guide software changes. In *Proceedings of International Conference on Software Engineering ICSE 2004*, Edinburgh, UK, May 2004.