***Proceedings of the 21st IEEE International Conference on Software Maintenance (ICSM 2005, Budapest, September 25-30), Industrial and Tool volume*, pages 89-92, 2005.**

Co-Change Visualization

Dirk Beyer
EPFL, Lausanne, Switzerland
dirk.beyer@epfl.ch

Abstract

Clustering layouts of software systems combine two important aspects: they reveal groups of related artifacts of the software system, and they produce a visualization of the results that is easy to understand. Co-change visualization is a lightweight method for computing clustering layouts of software systems for which the change history is available. This paper describes CCVISU, *a tool that implements co-change visualization. It extracts the co-change graph from a version repository, and computes a layout, which positions the artifacts of the software system in a two- or three-dimensional space. Two artifacts are positioned closed together in the layout if they were often changed together. The tool is designed as a framework, easy to use, and easy to integrate into reengineering environments; several formats for data interchange are already implemented. The graph layout is currently provided in VRML and SVG format, in a standard text format, or directly drawn on the screen.*

1 Concepts

Visualizations of the structure of software systems are helpful during reengineering and maintenance activities because they accelerate the understanding process. The extraction and comprehension of the subsystem structure is one of the most important tasks in this field, which is the more difficult the less documentation is available about the system. The change history —stored in the version control repository— is almost always available.

We propose to compute a clustering layout based on the change history of the system. Intuitively, two artifacts have close positions in such a layout if they were often changed together, and they have distant positions if they were rarely commonly changed. The method consists of two components. First, we build an abstraction of the system's change history, the so called co-change graph. Second, a force-directed graph-layout algorithm is used to compute a layout of the co-change graph, were the algorithm is driven by an energy model that is designed to produce clustering layouts. In the following, we briefly explain these two components of the method (cf. [2] for details).

Co-Change Graph. First, the tool CCVISU constructs the co-change graph, which is an abstraction of the version control repository. The (weighted) co-change graph for a given version control repository is an undirected graph $G = (V, E, w)$. The set of vertices V represents the artifacts of the system (e.g., files, classes, methods, packages) and change transactions (e.g., commits in CVS). An edge $\{c, a\}$ is contained in E if artifact a was changed by change transaction c. The weight $w(\{c, a\})$ of an edge is interpreted as the importance of the edge. For an unweighted graph, the weight is 1 for all edges. A detailed discussion on edge weights for co-change graphs is given in the technical report [3].

The motivation for using the co-change graph is threefold: First, frequently co-changed artifacts are likely to be logically coupled, and grouping them together in one subsystem restricts the scope of changes to the local context. Second, the co-change graph is not limited to program source code, unlike call graphs and other syntax-based models; the co-change graph includes also artifacts for test data, shell scripts, SQL scripts, examples, documentation, and subsystems in different programming languages. Third, the co-change graph can be efficiently and inexpensively extracted from version control repositories.

Clustering Layout. In the second stage of the method, a clustering layout of the co-change graph is computed. CCVISU is based on force-directed layout, which consists of two parts: an energy model assigns an energy to each layout for evaluation —the smaller the number, the better the layout—, and an algorithm computes a layout with minimal energy.

Such an algorithm usually works as follows: It starts with an initial layout, where the positions of the vertices are randomly assigned. Then, in every iteration, the algorithm tries to improve the layout according to the energy model (by using the first derivation of the energy function to compute a direction and a distance for the movement of each vertex). Since co-change graphs are usually large, we cannot use algorithms with complexity in $O(|V|^2)$ per iteration. The algorithm of Barnes and Hut [1] is in $O(|E| + |V| \log |V|)$ per iteration, and is therefore sufficient for our problem.

The energy model for co-change visualization has to fulfill several criteria, in particular, it should separate clusters and lead to interpretable distances. In difference to other graph-drawing applications, the energy model for co-change visualization must not enforce uniform edge length, must not be biased to the size of the clusters, and must be normalized to non-uniform degrees of the vertices. CCVISU implements the *weighted edge-repulsion LinLog* energy model:

$$\begin{aligned} U(p) &= \sum\nolimits_{\{u,v\}\in E} w(\{u,v\})\ ||p(u)-p(v)|| \\ &+ \sum\nolimits_{\{u,v\}\in V^{(2)}} -\deg_w(u)\ \deg_w(v)\ \ln||p(u)-p(v)||, \end{aligned}$$

where $p : V \to \mathfrak{R}^d$ is a layout, $U(p)$ is the energy of p, $||p(u)-p(v)||$ is the Euclidean distance of u and v in p, and $\deg_w(v)$ is the sum of the edge weights of edges incident to a vertex v. The first term of the sum is interpreted as attraction between connected vertices, because its value decreases when the distance of such vertices decreases. The second term is interpreted as repulsion between all pairs of (different) vertices, because its value decreases when the distance between any two vertices increases. The repulsion of each vertex v is weighted by the weighted edge degree $\deg_w(v)$, to avoid a bias to place vertices with heavy edge degree in the center of the layout.

The weighted edge-repulsion LinLog model is a straightforward extension of the model that we have successfully used in our initial study [2]. Noack provides a detailed introduction of the unweighted version and a comparison with the original LinLog model in the technical report [6].

Previous work. A comprehensive discussion of the related work is given in our technical report [3]. The two most related approaches are BUNCH [5] and the work of Eick and Wills [4]. The novelty of CCVISU is twofold: First, it is based on the co-change graph, not on syntax-based models. (There are other approaches using change history, but not for clustering.) Second, it is based on an energy model that is designed for clustering layout.

2 Tool Implementation

The tool CCVISU works as follows: First, the CVS logging information is parsed and analyzed to extract the file revision information. The gained information is combined to construct the co-change graph. In the second stage, the tool computes the layout for the vertices of the co-change graph. This part is done by an implementation of an energy model and an algorithm that computes a layout that has minimal energy according to the energy model. At the end, the graph needs to be displayed on the screen, or written to a file.

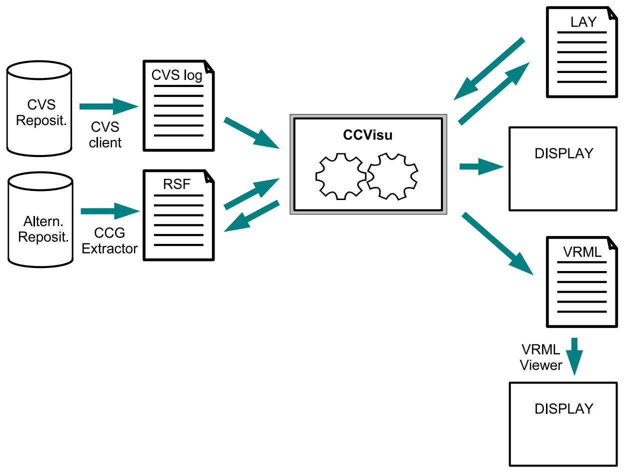

Figure 1. CCVisu's input/output interface

Input/Output (black-box view). Figure 1 shows the more general usage of the tool. The input is either (1) a CVS log file —extracted from the CVS version repository with the command `cvs log`—, or (2) a textual representation of the co-change graph in Rigi Standard Format (RSF) to compute layouts for co-change graphs extracted from other version control systems. To display a previously computed layout, the input can also be (3) a text file containing the layout (LAY).

The layout of the artifacts can be produced in three forms. (1) The text file (LAY) can later be read by CCVISU or other tools, such that the tool can be embedded in different environments. (2) The VRML format allows the use of an external VRML viewer (or a web browser with VRML plug-in) to view the layout, and is enabled for 2D as well as 3D layouts. Artifacts are drawn as spheres, and the name is annotated when the mouse pointer moves on the artifacts, or permanently annotated to the artifact via mouse click. The usage of the SVG format is similar and therefore omitted in Figure 1. However, SVG can be used to display much larger layouts, but without 3D effects. (3) The layout can be directly displayed on the screen. This form is the preferred output method for huge graphs, when VRML and even SVG viewers are not able to reproduce the layout on the screen. Artifacts are drawn as filled circles, and the names of the artifacts are shown in a separate frame when the mouse pointer moves onto an artifact, or permanently annotated to the artifact via mouse click. (4) Besides the layouts, the tool can also output the extracted co-change graph in RSF.

Framework (white-box view). CCVISU is designed as a framework to make improvements and extensions easy, and to enable integration into other reengineering tools. Figure 2 shows the components of the tool. Basically, the input graph is read by a reader component, passed to the layout algorithm, and the output is written by a writer component. The reader interface has currently three implementa-

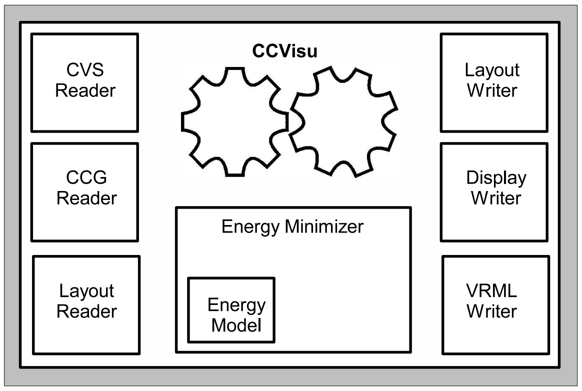

Figure 2. CCVisu's framework architecture

tions: for reading CVS log files, co-change graphs in RSF, and layouts in text format. The writer interface has five implementations so far: for writing co-change graphs in RSF format, layouts in text format, VRML and SVG layouts, and for writing the layout directly to the screen.

The version control system CVS does not directly keep the information which files were checked-in together in the repository. The transactions need to be recovered from the logged information about time, user, and log message. The current CVS reader implements the heuristic used in CVS2CL[1]: it considers a sequence of changes of files as one change transaction if the changes have the same user login, the same log message, and time stamps that differ by at most 180 s (the constant can be adjusted). The co-change graph is extracted on the file level. However, if a more fine-grained visualization is necessary (e.g., on method level), the techniques used in ROSE [7] can be integrated as additional reader. On the other hand, co-change graphs on higher levels (e.g., on package level) can be obtained by applying a technique called 'lifting'.

Using these flexible input/output formats, the tool can also be used as a force-directed layouter for other kinds of graphs, not only co-change graphs. To provide more control over the concrete layout computation, the minimizer algorithm and the energy model themselves are also abstract or generic components. Currently, CCVISU includes an implementation of the *Barnes-Hut algorithm* [1] as minimizer, and supports several energy models for the evaluation of layouts. However, to achieve clustering layouts for software graphs, we have to choose an energy model that fulfills certain clustering criteria. This is why we use the *weighted edge-repulsion LinLog model* (defined in Sect. 1) as default.

3 Application

The layouts produced by the tool CCVISU provides information on two levels:

[1]Available at http://www.red-bean.com/cvs2cl

- If the co-change graph contains clusters, then —due to the clustering quality of the energy model— the clusters are separated. Therefore, on the higher level, it reveals the subsystem structure of the system if the repository information allows so, and provides an overview over the relationships between the subsystems on the coarse level.

- Artifacts that were often changed together are placed closed together in the layout. Therefore, on the lower level, the engineer can use the visualization to find out in which context the artifact is used, which other artifacts need to be understood to understand the artifact, and if the artifact needs to be changed, it provides the artifacts that are most likely to change as well in the close neighborhood of the artifact.

The visualization can, for example, provide some guidance for answering concrete questions like the following: Which SQL query files correspond to which module of the system? Which test input file is related to which code file? Which configuration file corresponds to which module files? If we change a certain file, which files should we understand because of potential impact? If we are interested to unterstand a certain code file, which documentation file shall we read? If we want to test a certain part of the program, which example files and test cases are closely related to the source file of that part?

Example Visualization. We have applied the CCVISU method to the well-known software project MOZILLA, in particular to the *mailnews* component without the base package. The co-change graph was extracted from a CVS log file with 270 000 lines (13 MB). In this example, the artifacts of the co-change graph are files. The graph consists of 1 804 artifact vertices, 9 950 vertices for change transactions, and 30 938 edges (changes). Figure 3 shows a screenshot of the layout, which was computed within 5 min on a 1.7 GHz Pentium M machine, using only 100 iterations of the minimizer.

The vertices for the change transactions and the edges are omitted for readability. The artifact vertices were drawn in different colors, in order to compare the grouping suggested by the layout with the authoritative decomposition, according to the documentation. We considered 8 major subsystems of the *mailnews* component and assigned colors to them: AddrBook (blue), Compose (magenta), IMAP (pink), MAPI (yellow), MIME (red), Import (cyan), DB (orange), and Extensions (gray). The rest (minor components, build utils, etc.) is labeled as Misc (green) in the figure. (The subsystem labels are also annotated in gray boxes, to improve readability for gray-scale printouts.) Now we can compare whether CCVISU has positioned the 1 804 files in groups in agreement with the authoritative decomposition: Some of the subsystems are clearly separated from the rest

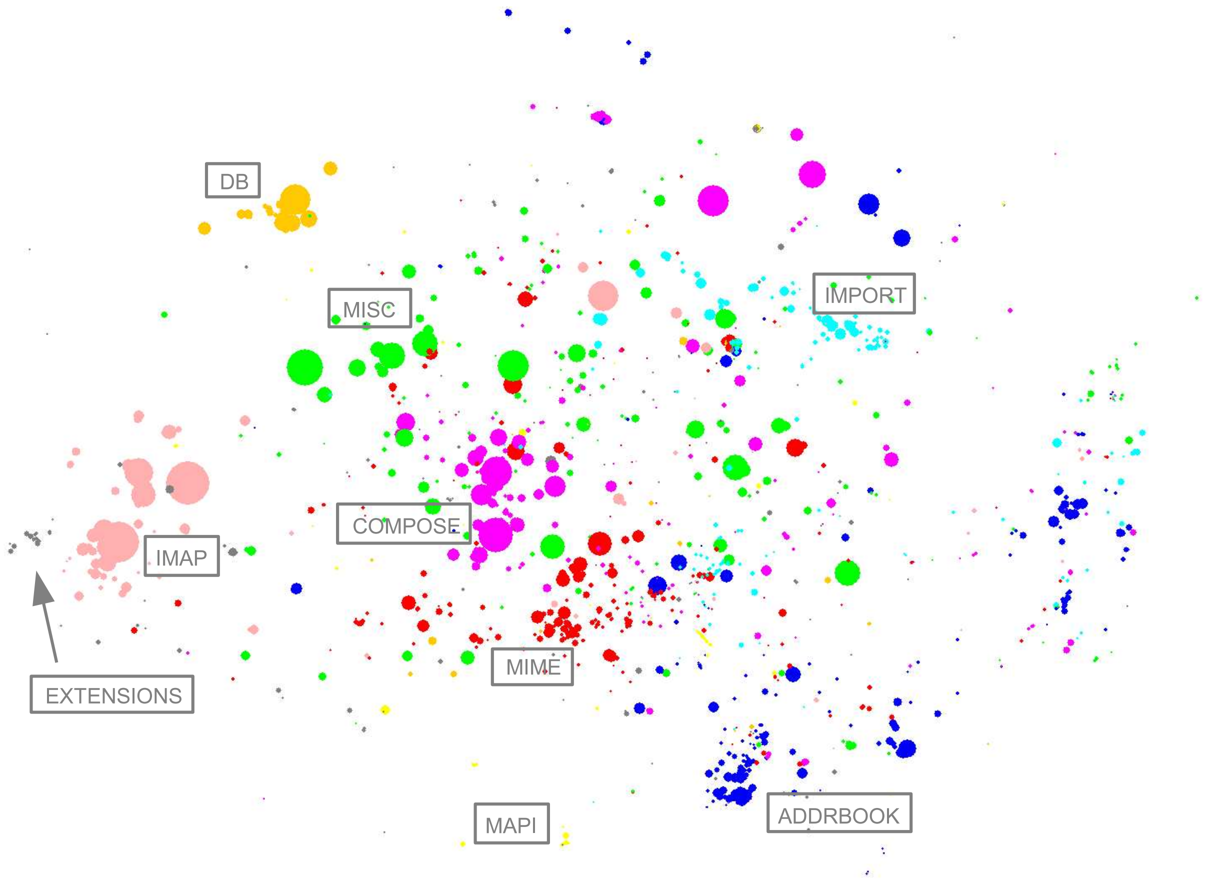

Figure 3. Co-change visualization of MOZILLA's mailnews component

(Extensions, IMAP, DB, MAPI, AddrBook), some are not separate clusters but almost all files of the same subsystem are closed together (Import, MIME, Compose), and Misc is not grouped at all (as expected). Due to space, a detailed interpretation is not possible here. However, more example visualizations together with discussion and interpretation of the layouts are given in the concept paper [2].

4 Summary

Co-change visualization is a lightweight method for visual clustering of software systems. The tool implementation CCVISU is easy to use and designed as a generic tool framework to enable easy integration of additional concepts. It is also easy to integrate into other tools. Our initial study showed that the approach of software clustering based on historical common changes produces good results and works efficiently. The layouts reveal the subsystem structure, in conformance with authoritative decompositions, and provide for each artifact the related artifacts in the close neighborhood. Using the method in reverse engineering can accelerate the understanding of the system, and during maintenance activities it can provide helpful guidance before performing changes.

More details on the concepts and a comparison of the layouts with the Fruchterman-Reingold layouts are given in the technical report [3], and all layouts and input data (inclusive raw co-change graph data) of our initial study are available on the CCVISU web page. CCVISU is released under GNU LGPL and publicly available at http://mtc.epfl.ch/∼beyer/CCVisu.

References

[1] J. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324:446–449, 1986.

[2] D. Beyer and A. Noack. Clustering software artifacts based on frequent common changes. In *Proc. IWPC*, pages 259–268. IEEE, 2005.

[3] D. Beyer and A. Noack. Mining co-change clusters from version repositories. Technical Report IC/2005/003, EPFL Lausanne, 2005.

[4] S. G. Eick and G. J. Wills. Navigating large networks with hierarchies. In *Proc. Visualization*, pages 204–210, 1993.

[5] S. Mancoridis, B. S. Mitchell, Y. Chen, and E. R. Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *Proc. ICSM*, pages 50–59. IEEE, 1999.

[6] A. Noack. Visual clustering of graphs with nonuniform degrees. Technical Report 02/04, BTU Cottbus, 2004.

[7] T. Zimmermann, S. Diehl, and A. Zeller. How history justifi es system architecture (or not). In *Proc. IWPSE*, pages 73–83. IEEE, 2003.