Asymmetric Visual Hierarchy Comparison with Nested Icicle Plots

Fabian Beck[1], Franz-Josef Wiszniewsky[2], Michael Burch[1], Stephan Diehl[2], and Daniel Weiskopf[1]

[1]VISUS, University of Stuttgart, Germany
[2]University of Trier, Germany

Abstract. The comparison of hierarchies is a data analysis task for that a number of visualization approaches already exist. Generally, this can be regarded as a special form of graph comparison. These techniques typically handle two or more compared hierarchies all in the same way. In many practical applications, however, there are reasons why one of the hierarchies is more important than others. We, hence, propose a novel visualization approach to reflect this asymmetry in importance. A focused primary hierarchy is visualized as a large icicle plot, whereas a secondary hierarchy is only shown on demand, nested in the nodes of the primary hierarchy. Similarities of the two hierarchies are color-coded. We show the applicability of the approach in a case study comparing a hierarchically organized software system to a clustering result.

Keywords: Hierarchy comparison, hierarchy visualization, graph clustering, graph visualization, software visualization.

1 Introduction

Hierarchies help abstracting large sets of entities on multiple levels of detail. Being used in file systems, for classifying species, or organizing companies, they are a ubiquitous structure for analyzing data. Hierarchies are often also used to abstract and simplify graphs, for instance using graph clustering. It is, however, not always clear how a set of entities needs to be structured as a hierarchy: different opinions of experts might exist, several clustering algorithms can be used, or hierarchies change over time. Hence, oftentimes competing variants of hierarchies exist. Finding and interpreting similarities and differences in hierarchies is a non-trivial task but that can be supported by visualization.

When comparing hierarchies, one hierarchy might be more important or acts as a kind of reference in the comparison. For instance, when a software developer compares a hierarchical decomposition of a software system to an automatically generated clustering of the same system based on the dependency graph, the original structure is probably more important: the developer knows that structure very well and might only be willing to change it gradually. In another case, when comparing the hierarchical file structure on your personal computer to a backup copy, probably the focus is on the current file structure not on the copy. Another example is biologists contrasting their classification of species or a standard one to a newly proposed classification. In general,

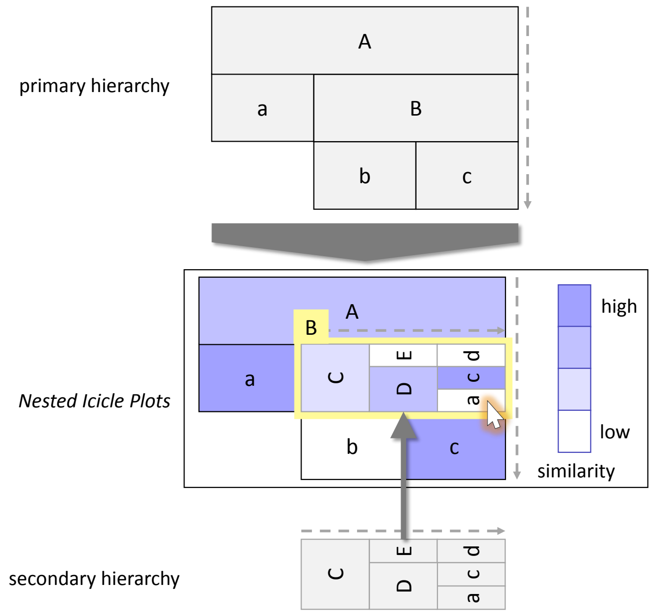

Fig. 1: Visually comparing a primary hierarchy to a secondary hierarchy based on icicle plots: on demand, a rotated version of the secondary hierarchy is nested into a node of the primary hierarchy; similarities are shown by color-coding.

it might be even more likely that the two hierarchies compared have a different level of relevance for the application at hand, rather than that they are equally important.

With these observations, we propose the novel visualization approach of *Nested Icicle Plots* for comparing two hierarchies as illustrated in Figure 1. While most previous approaches are inherently symmetric (i.e., handle and visualize the hierarchies alike), our approach is one of few approaches that focuses on an asymmetric comparison and treats the hierarchies differently. In particular, we show the primary hierarchy as a screen-filling icicle plot. The color of each node encodes its similarity to the secondary hierarchy. A detailed comparison can be obtained on demand in the form of an icicle plot of the secondary hierarchy nested within the icicle representation of the primary hierarchy. Interactions like local zooming enable the user to analyze larger hierarchies with this technique. We demonstrate the approach in a case study on restructuring a software project according to a clustering result.

2 Related Work

Hierarchy visualization can be considered as a discipline of graph visualization. However, many independent visualization techniques have been introduced that are specialized to this subproblem and were surveyed by Jürgensmann and Schulz [1, 2]. Besides node-link diagrams, standard techniques include stacking approaches like icicle plots [3] or nesting approaches like treemaps [4]. Also radial variants of these techniques were explored, for instance, bubble layouts for node-link diagrams [5, 6] or circular versions of icicle plots [7–9]. These visualizations, by default, display only one hierarchy, but some of them have been already tailored for comparison.

The visual comparison of hierarchical structures was surveyed by Graham and Kennedy [10]. They classified existing approaches into five categories: *edge drawing*, *coloring*, *animation*, *matrix representation*, and *agglomeration*. Our approach of *Nested Icicle Plots* falls into two categories of this taxonomy: it shares characteristics of an *agglomeration* approach because the focus is one hierarchy that contains information from the other hierarchy; in addition, *coloring* is used to show similarities. On a higher level of abstraction, the approach is a form of visual comparison and can be classified as an *explicit encoding* according to the taxonomy of Gleicher et al. [11]: similarities and differences of the hierarchies are explicitly encoded in the visualization through colors.

In general, most of previous work on hierarchy comparison handles the compared hierarchies symmetrically. For instance, *TreeJuxtaposer* [12] places two hierarchy representations next to each and enables a comparison based on color-coding as well as brushing and linking. Holten and van Wijk [13] use two icicle plots with leaf nodes facing each other and connect the leaf nodes by bundled edges indicating equivalent nodes. Based on a similar layout, Lutz et al. [14] discuss the editing of hierarchies in context of comparison. Beck and Diehl [15] use a matrix to indicate similarities in two hierarchies attached to the sides of the matrix. Also, the temporal evolution of hierarchies has been studied already [16].

We, however, found only few examples with an asymmetric hierarchy comparison. One example is *Trees in a Treemap* [17], where a treemap is overlaid by a node-link tree diagram. While the treemap stays readable, the layout of the node-link diagram, being dictated by the treemap, hardly is. Bremm et al. [18] compare multiple hierarchies and use, amongst others, views that compare one reference hierarchy to a set of others: a node-link representation of the reference hierarchy is color-coded in one view to illustrate which nodes are preserved in the other hierarchies; another view shows the set of other hierarchies as juxtaposed node-link diagrams and therein encodes differences to the reference. A similar view has already been described by Chevenet et al. [19]. Although those approaches work with an asymmetric representation, they do not exploit this asymmetry specifically for the application scenario at hand.

Other agglomeration-based techniques also show one main hierarchy that encodes differences of two original hierarchies, but the agglomerated hierarchy is constructed symmetrically. Guerra Gomez et al. [20] map the change of node weights to color and size of the nodes, whereas nodes only appearing in one of the compared hierarchies are marked by a special border line. Graham and Kennedy [21] construct and visualize a direct acyclic graph from multiple hierarchies; a kind of colored bar code within the nodes indicates which nodes belong to which original hierarchy. In contrast to these approaches, our approach shows the primary hierarchy as is and does not merge two hierarchies.

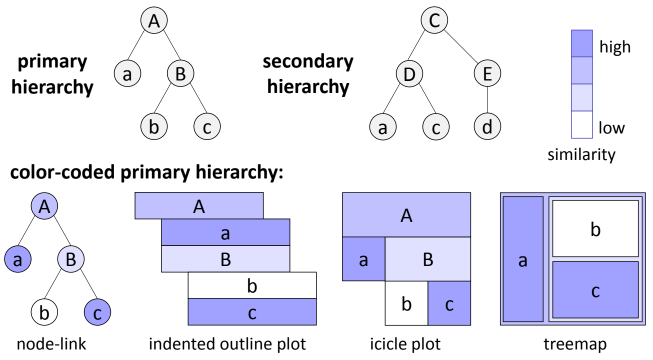

Fig. 2: Design alternatives for color-coding similarities of two hierarchies (here, the maximum similarity of a node in the primary hierarchy to any node in the secondary hierarchy) within the representation of the primary hierarchy.

3 Nested Icicle Plots

The basic idea of our approach is color-coding similarities of the two hierarchies in the primary hierarchy and showing details on demand of the secondary hierarchy nested in the nodes of the primary one. Although this idea is independent of the visual representation of each of the hierarchies, icicle plots seem particularly suitable for implementing the approach. In particular, our version of nested hierarchies, as illustrated in Figure 1 and implemented in a prototype, is based on nesting icicle plots. Interactions are important to increase the flexibility and visual scalability of the approach.

3.1 Color-Coded Similarities

Depicting a single hierarchy—as a node-link tree diagram, an indented outline plot, an icicle plot, or a treemap (Figure 2, cf. [10])—the elements of the hierarchy are represented by visual circular or rectangular nodes. The color of these nodes can be used for encoding a property of the nodes. For an asymmetric hierarchy comparison, the color-coded property for the nodes of the primary hierarchy should relate to the similarity of the node to the nodes of the secondary hierarchy. The metric needs to determine whether a node in one hierarchy has an equivalent, or at least similar node in the other hierarchy. Hence, for each node in the primary hierarchy, we want to find the most similar node in the secondary hierarchy.

While it is possible to use different definitions of similarity for this, one of the most straightforward candidates is measuring the relative overlap of pairs of nodes: In particular, we assume that there exists an equivalence relation for entities represented by leaf nodes (e.g., provided through the name of the node). Then, each node represents a set of entities. In case of leaf nodes, this set has only a single element while inner nodes aggregate descendant entities of leaf nodes. A measure of set overlap of two sets *A* and *B* is the Jaccard coefficient

$$sim(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

For each node of the primary hierarchy, the most similar node in the secondary hierarchy can be retrieved. This maximum similarity is finally mapped to the color of the nodes and creates hierarchy visualizations as depicted in Figure 2 for different types of hierarchy representations.

Hence, this approach assigns each node of the first hierarchy (at least) one most similar node in the secondary hierarchy. Please note that the same node of the secondary hierarchy can be assigned to multiple nodes of the primary hierarchy. If the primary hierarchy is identical to the secondary hierarchy, all maximum similarities of nodes of the primary hierarchy are 1.0. This also holds if additional intermediate nodes are inserted into the secondary hierarchy, but not if additional intermediate nodes are inserted into the primary hierarchy. For two hierarchies containing the same set of leaf nodes, the similarity value of the root nodes is always 1.0 because the two root nodes are a perfect match.

3.2 Nesting Hierarchies

For a detailed comparison it is, however, also important to see which elements of the secondary hierarchy cause a certain similarity. Therefore, we may embed a small representation of the secondary hierarchy into each node of the primary hierarchy, which shows the similarities between the node of the primary hierarchy and the full secondary hierarchy. We use the color of each node of the secondary hierarchy to encode its Jaccard similarity to the surrounding node of the primary hierarchy. This is demonstrated in Figure 1 for node B of the primary hierarchy using icicle plots. Doing this for the complete primary hierarchy, every node of the primary hierarchy is compared to every node of the secondary hierarchy. This approach has a clear focus on the primary hierarchy as it is represented only once in large size whereas the secondary hierarchy is represented in small multiples, each time from the perspective of a node of the primary hierarchy.

This basic approach is open to arbitrary hierarchy representations where nodes are visualized as closed-shape visual objects. If the nodes are already nested like in a treemap (Figure 2), however, there is no room for depicting a secondary hierarchy, at least not for inner nodes. For other representations that use links, spatial closeness, or indentation for encoding the hierarchy (Figure 2), a secondary hierarchy can be embedded into the nodes of the primary hierarchy. It is even possible to use different representations for the primary and secondary hierarchy.

Nevertheless, some representations are more suitable for the intended purpose than others. For instance, nodes of node-link diagrams are quite small because space is required for drawing the links, which considerably limits the space that can be used for drawing the secondary hierarchy. The nodes of indented outline plots, in contrast, are more space filling. Each node—inner nodes and leaf nodes—has the same size, which is advantageous for small hierarchies because the secondary representation always has the same space and aspect ratio; it is a problem, however, for larger hierarchies when this standard size becomes too small. In contrast, in icicle plots, the size of the inner nodes accumulates the sizes of all children so that even in larger hierarchies, the main inner nodes have reasonable size to encode a secondary hierarchy. Icicle plots are further able to completely fill every rectangular shape. Hence, we recommend using icicle plots for representing the primary hierarchy. One downside of this, however, is that the nodes have different aspect ratios.

For embedding the secondary hierarchy into the nodes of the icicle plots, also an arbitrary hierarchy representation can be selected. Again, we chose icicle plots, for two main reasons: first, they stay quite readable at for larger datasets and visualized in small size, and second, the user does not have to deal with two different paradigms to visualize a hierarchy. Instead of using the same orientation of the icicle plots for both hierarchies, we recommend flipping the orientation of the secondary hierarchy by 90° relative to the primary hierarchy—the two hierarchies can be easily discerned. Hence, either horizontally split icicle plots are contained in vertically split ones, or vice versa. For landscape format screens splitting the primary hierarchy vertically as shown in Figure 1 is most appropriate. These are just suggestions, but the general nesting approach is open for using other hierarchy setups and representations—which one works best under which circumstances will need to be evaluated in user studies.

3.3 Interactive Comparison

The approach can be improved using interaction techniques. Making the alternatives discussed above configurable, the representation can be optimized for each data set individually. In particular, our prototype implementation allows for switching the orientation of the two hierarchies individually. Other representations than icicle plots are currently not supported by our implementations.

Showing the secondary hierarchy for every node of the primary hierarchy quickly leads to many small visual elements and hides the proposed color-coding of the primary hierarchy. We therefore show the secondary hierarchy only on demand for selected nodes of the primary hierarchy. Hence, the user first sees the color-coded primary hierarchy; now, moving the mouse over a hierarchy node, a nested secondary hierarchy appears as illustrated in Figure 1. To mark its current, distinct role, a hovered node should become visually highlighted (here, with a thick yellow border). Since the secondary hierarchy completely fills the inside space, the label of the node is attached to the top of the rectangle.

Although main inner nodes of the icicle plot are usually large enough to embed a secondary hierarchy, it is difficult to retrieve information from small-sized lower inner nodes as well as leaf nodes. Interactively zooming-in on a specific node alleviates this problem. Instead of a global geometric zoom, we use a semantic zoom that enlarges a node without displacing other nodes from screen or overlapping these—no scrolling is required in this focus–context approach. To implement the zooming, weights determine the width and height of a node. Initially, the height weight is constant for each node and the width weight is based on the number of contained leaf nodes. Zooming-in on a node increases both weights by a constant factor for the respective node. To keep all nodes on one level aligned, the height weight of all other nodes at the same hierarchy level need to be increased accordingly. Zooming-out reverses the weight increases respectively. Figure 3 shows an example where the default representation (left) is zoomed in on node 6 (middle). Having mutiple zoomed foci is also possible, as shown in Figure 3 (right) for nodes A and 6.

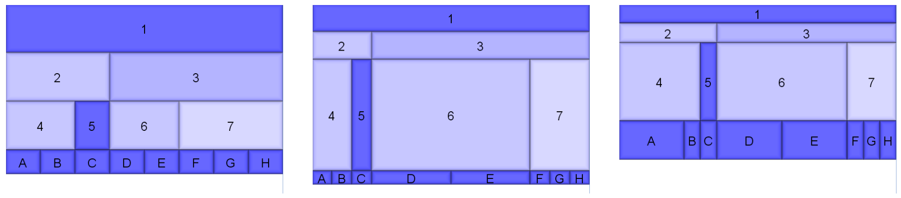

Fig. 3: Semantic zooming; left: default; middle: zoomed node 6; right: zoomed nodes A and 6.

4 Case Study

To illustrate the utility of our approach, we take up the software engineering scenario briefly mentioned in Section 1: a software developer wants to restructure the hierarchical modularization of a software system. The developer runs an automatic clustering algorithm that produces an optimized hierarchical clustering with respect to the dependency graph of the system. This clustering, however, is very different from the original modularization. The developer does not want to restructure the complete system but rather likes to compare the original modularization with the clustered one to check and take over parts of the suggested changes.

Demonstrating this application, we compare the hierarchical modularization of the *Azureus* project (a Bittorent client, now called *Vuze*; restricted to the `core3` package with 477 classes and 69 subpackages) to a clustering result based on structural dependencies within the project, as described in a study on software clustering [22] and already visualized in a matrix-based approach [15] (details on the data acquisition are described there). Since the developer is familiar with the original modularization and inner nodes have meaningful labels, the original modularization is selected as the primary hierarchy. Figure 4 (top) show this modularization with color-coding according to the similarities to the clustered modularization.

Dark colors tell that some of the modules of *Azureus* are well-matched by the clustering result, for instance, `html` or `ipfilter`. Other modules, in contrast, are colored only in light blue, indicating a low similarity to the clustered hierarchy, such as the modules `global` or `util`; these are hardly confirmed by the clustering algorithm, as looking at the nested clustering hierarchy confirms. Reasons are probably deficiencies in the clustering algorithm: these particular two modules assemble functionality that is globally used within *Azureus* and previous findings indicate that those global modules can hardly be matched by clustering based on structural dependencies [23].

While it is good to know which are the well-matched and worse-matched modules, partially matched modules might be of even larger interest: for those, the clustering leads to a slightly different structure that, however, is not too far from the current modularization. For an example of a detailed analysis of a partially matched module, we choose the `tracker` module. Figure 4 (middle) shows the nested clustered hierarchies retrieved on demand in zoomed-in versions of the `tracker` module; Figure 4 (bottom) further provides details on its main submodules `client`, `host`, `protocol`, and `server`.

The nested hierarchy within the `tracker` modules quickly reveals that the clustering algorithm organized the content of the module mainly into two clusters: `0.0.0.1` and

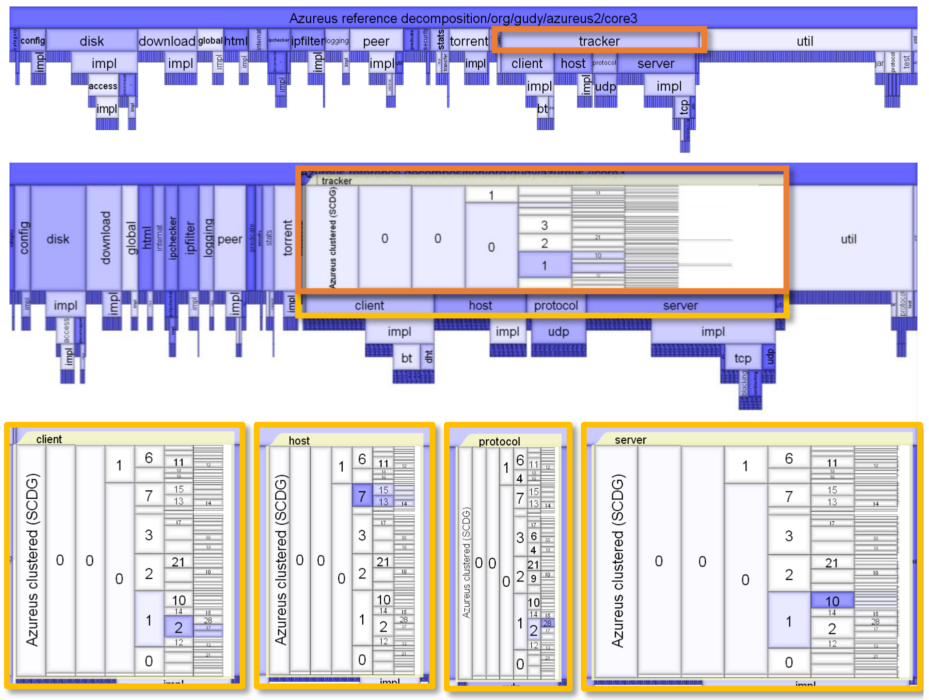

Fig. 4: Package structure of the *Azureus* project as primary hierarchy in comparison to a clustered decomposition as secondary hierarchy; top: primary hierarchy; middle: nested secondary hierarchies retrieved on demand for the `tracker` module (zoomed); bottom: direct main submodules of the `tracker` module (zoomed and clipped).

`0.0.0.7`. Hence, clustering suggests splitting the module into two parts. According to the nested hierarchies in the submodules, one part would consist of the `client`, `protocol`, and `server` modules as they are assembled in cluster `0.0.0.1`, whereas the `host` module in `0.0.0.7` should become independent of the others. Following the subcluster of cluster `0.0.0.1`, the `client` and `protocol` module belong closer together as they are both contained in cluster `0.0.0.1.2`—adding another submodule wrapping the two might further improve the modularization.

In that way, the developer might proceed with the other partially matched modules comparing the original modularization to the clustering result.

5 Discussion and Limitations

We are aware that there are several limitations of our approach:

– **Visual scalability:** The dimensions to be handled are the number of hierarchy nodes and the depth of the hierarchy. Both decrease the size of the boxes, but zooming remedies the problem.

– **Algorithmic scalability:** The comparison of hierarchies has at least a quadratic runtime complexity with respect to the number of hierarchy nodes because all nodes of one hierarchy need to be compared to all nodes of the other hierarchy (when all nested icicle plots are required). However, for just computing the coloring of the primary hierarchy, there exist faster solutions [24].
– **Multiple hierarchies:** Two different scenarios are of interest here: First, we might have to deal with one primary hierarchy and *n* secondary ones. A visualization can be applied as small multiples of the primary hierarchy (each representing a 1:1 comparison). Second, the same situation might occur but the other way round—we have to deal with *n* primary ones, but only one secondary candidate (again small multiples but this time for the *n* primary ones).

In general, the approach is not limited to the demonstrated example in Section 4, but might be useful for other clustering applications in different domains such as in the field of biology or for evaluating clustering algorithms.

6 Conclusion and Future Work

Nested Icicle Plots are a novel approach to visual hierarchy comparison. Discussing several design alternatives, we chose icicle plots as a hierarchy representation because they are space-filling and flexible enough for the suggested nesting. Since showing the nested structures at any time could produce information overload, the secondary hierarchy is only depicted on demand, whereas color-coded nodes provide a preview of the comparison by default. The case study gives a first example that demonstrated how the visualization can be used in the context of software engineering; however, our approach is general and can be applied to any hierarchy comparison problem. Being inherently asymmetric, the technique is suitable for scenarios where one hierarchy is of particular interest or importance.

Future work includes to extend the hierarchy comparison to the comparison of general graphs, i.e., a nested graph comparison. From a perceptual and a usability point of view, our visualization technique still has to be evaluated in a comparative user study. This may also be investigated by eye tracking, to give additional insights into when and where problems occur.

References

1. S. Jürgensmann and H.-J. Schulz, "A visual survey of tree visualization," *IEEE VisWeek 2010 Posters*, 2010.
2. H.-J. Schulz, "Treevis.net: A tree visualization reference," *IEEE Computer Graphics and Applications*, vol. 31, no. 6, pp. 11–15, 2011.
3. J. Kruskal and J. Landwehr, "Icicle Plots: Better displays for hierarchical clustering," *The American Statistician*, vol. 37, no. 2, pp. 162–168, 1983.
4. B. Shneiderman, "Tree visualization with Tree-Maps: 2-D space-filling approach," *ACM Transactions on Graphics*, vol. 11, no. 1, pp. 92–99, 1992.
5. S. Grivet, D. Auber, J. Domenger, and G. Melançon, "Bubble tree drawing algorithm," in *Proceedings of International Conference on Computer Vision and Graphics*, 2004, pp. 633–641.

6. C. C. Lin and H. C. Yen, “On balloon drawings of rooted trees,” *Graph Algorithms and Applications*, vol. 11, no. 2, pp. 431–452, 2007.
7. K. Andrews and H. Heidegger, “Information Slices: Visualising and exploring large hierarchies using cascading, semicircular disks,” in *Proceedings of IEEE Symposium on Information Visualization*, 1998, pp. 9–11.
8. J. T. Stasko and E. Zhang, “Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations,” in *Proceedings of the IEEE Symposium on Information Visualization*, 2000, pp. 57–65.
9. J. Yang, M. O. Ward, E. A. Rundensteiner, and A. Patro, “InterRing: A visual interface for navigating and manipulating hierarchies,” *Information Visualization*, vol. 2, no. 1, pp. 16–30, 2003.
10. M. Graham and J. Kennedy, “A survey of multiple tree visualisation,” *Information Visualization*, vol. 9, no. 4, pp. 235–252, 2010.
11. M. Gleicher, D. Albers, R. Walker, I. Jusufi, C. D. Hansen, and J. C. Roberts, “Visual comparison for information visualization,” *Information Visualization*, vol. 10, no. 4, pp. 289–309, 2011.
12. T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou, “TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility,” *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 453–462, 2003.
13. D. Holten and J. J. van Wijk, “Visual Comparison of Hierarchically Organized Data,” *Computer Graphics Forum*, vol. 27, no. 3, pp. 759–766, 2008.
14. R. Lutz, D. Rausch, F. Beck, and S. Diehl, “Get your directories right: From hierarchy visualization to hierarchy manipulation,” in *Proceedings of the 2014 IEEE Symposium on Visual Languages and Human-Centric Computing (to appear)*. IEEE, 2014.
15. F. Beck and S. Diehl, “Visual comparison of software architectures,” *Information Visualization*, vol. 12, no. 2, pp. 178–199, 2013.
16. A. Telea and D. Auber, “Code Flows: Visualizing structural evolution of source code,” *Computer Graphics Forum*, vol. 27, no. 3, pp. 831–838, 2008.
17. M. Burch and S. Diehl, “Trees in a Treemap: visualizing multiple hierarchies,” in *Visualization and Data Analysis*, vol. 6060. International Society for Optics and Photonics, 2006, pp. 60 600P–60 600P–12.
18. S. Bremm, T. von Landesberger, M. Hess, T. Schreck, P. Weil, and K. Hamacherk, “Interactive visual comparison of multiple trees,” in *Proceedings of the 2011 IEEE Conference on Visual Analytics Science and Technology*. IEEE, 2011, pp. 31–40.
19. F. Chevenet, C. Brun, A. L. Banuls, B. Jacq, and R. Christen, “TreeDyn: towards dynamic graphics and annotations for analyses of trees,” *BMC Bioinformatics*, vol. 7, no. 1, pp. 439+, 2006.
20. J. Guerra Gómez, A. Buck-Coleman, C. Plaisant, and B. Shneiderman, “Interactive visualizations for comparing two trees with structure and node value changes,” Human-Computer Interaction Lab, University of Maryland, Tech. Rep., 2011. [Online]. Available: http://cgis.cs.umd.edu/localphp/hcil/tech-reports-search.php?number=2011-22
21. M. Graham and J. Kennedy, “Exploring multiple trees through DAG representations,” *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1294–1301, 2007.
22. F. Beck and S. Diehl, “On the impact of software evolution on software clustering,” *Empirical Software Engineering*, vol. 18, no. 5, pp. 970–1004, 2013.
23. P. Andritsos and V. Tzerpos, “Information-theoretic software clustering,” *IEEE Transactions on Software Engineering*, vol. 31, no. 2, pp. 150–165, 2005.
24. L. Zhang, “On matching nodes between trees,” Systems Research Center Laboratory, HP Laboratories Palo Alto, Tech. Rep., 2003. [Online]. Available: http://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2003/HPL-2003-67.pdf