Multiscale and Multivariate Visualizations of Software Evolution

Lucian Voinea[*]
Technische Universiteit Eindhoven

Alexandru Telea[†]
Technische Universiteit Eindhoven

Abstract

Software evolution visualization is a promising technique for assessing the software development process. We study how complex correlations of software evolution attributes can be made using multivariate visualization techniques. We use a combination of color and textures to depict up to four artifact attributes at the same time in one view using the same spatial layout. Next, we describe an interactive navigation method of the attribute space that can extend the correlation capabilities to four or more attributes. A second issue we address is how to use clustering to reduce the complexity of evolution visualizations. We propose two new methods, *isometric* and *isorelevance*, to generate relevant abstraction levels in a hierarchical clustering of software evolution artifacts. The *isometric* method generates partitions with similar size elements. The *isorelevance* method generates partitions with elements of similar relevance. We propose a novel widget, the cluster map, which visualizes all partitions in a clustering and supports users when making size/relevance compromises when choosing a partition. We illustrate the applicability of the proposed techniques with two usage scenarios on the evolution of two real-life industry size projects.

Keywords: Evolution visualization, Software visualization, CVS

CR categories: D.2.7 [Software engineering]: Distribution, Maintenance, and Enhancement – documentation, reengineering; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval – clustering, query formulation; I.3.8 [Computer Graphics]: Applications

1 Introduction

Visualization of the evolution of software has recently emerged as a promising research direction. It uses software history recordings, e.g. from Software Configuration Management (SCM) systems, to build graphical representations of evolution of software artifacts. The main goal of such representations is to let users explore the software evolution and make visual correlations that lead to relevant findings regarding the process assessment or system understanding.

In this paper, we build on previous research efforts in the field of software evolution visualization. We address two aspects of the problem of coping with the large software size to be visualized: many data elements (e.g. files and file versions in a repository) and many attributes per element (e.g. file size, type, and author, and commit time and comments). We address the first problem by using a multiscale (or hierarchical) software decomposition and a new visual widget for displaying this hierarchy and letting users choose from its relevant levels of detail. We address the second problem by a new visual approach that enables complex visual correlations over multivariate data.

The structure of this paper is as follows. In Section 2, we review previous work on software evolution visualization. In Section 3, we outline the basic visualization model on which we build our multivariate and multiscale visualization techniques. Section 4 details our multivariate visualization approach. Section 5 describes the *cluster map*, a new widget for visualizing and choosing from several possible decompositions and several levels of detail thereof. Section 6 presents a number of usage scenarios that demonstrate the applicability of the proposed techniques. Section 7 concludes the paper with a summary of our contribution and outlines future research directions.

2 Related work

The massive growth in popularity and use of SCM systems, influenced by open source projects like CVS and Subversion, opens new possibilities for project accounting, auditing and understanding. Efforts have been focused so far in two research directions: data mining and data visualization.

Data mining research focuses on processing and extracting relevant information from the evolution data stored into SCM systems. However, most data mining approaches work by trying to fit an existing ‘data model’ on the raw information stored by the SCM systems, which is fine if the model is correct and exactly what the user wants to see, but may be of limited use otherwise. Many techniques have been proposed to offer access to higher level, aggregated information about the project evolution [Gall *et al.* 2003; Zimmermann *et al.* 2004].

Data visualization, the second research direction, takes the different path of making the large amount of evolution data effectively available to the user. Visualization techniques use a ‘weak’ data model, as the goal is to let the users discover patterns and trends by themselves, rather than hard-coding such patterns in the mining process. Many visualizations tools have been proposed to assist users in analyzing the software evolution data [Eick *et al.* 1992; Froehlich and Dourish 2004; Lanza 2001; Wu *et al.* 2004; Voinea *et al.* 2005; Pinzger *et. al* 2005; Voinea and Telea 2006]. These tools can provide insightful evolution overviews. However, they do not enable users to perform efficient correlations over the entire evolution that involve more than one or two software attributes at the same time. There are two exceptions. First is Augur [Froehlich and Dourish 2004], a tool that divides the space

[*] email: l.voinea@tue.nl

[†] email: alext@win.tue.nl

an entity is drawn over into several areas, and displays one attribute per area using color coding. This, however, creates a discontinuity in the visualization space, both in terms of layout and color. The same color can refer to different attribute types, which hinders making correlations based on color. The second exception is described in [Pinzger *et. al* 2005]. It proposes a visualization that can successfully depict the evolution of a large number of attributes using Kiviat diagrams. This approach seems to scale well for comparing up to 10 – 20 releases of a project.

There are, however, several notable efforts in other visualization fields that propose multivariate techniques which preserve entity layout continuity. Iterrante and Shenas propose a combination of 'natural' textures with color for depicting two attributes at the same time [Interrante 2000; Shenas and Interrante 2006]. Weigle *et al.* use oriented patterns and luminance to encode overlapping scalar fields while preserving their identity [Weigle *et al.* 2000]. Holten *et al.* use procedural textures and color to depict the distribution of two attributes over the structure of a software system [Holten *et al.* 2005]. The texture-color combination principles advocated by the abovementioned methods could be adapted also for software evolution visualization.

A second problem of software evolution visualization is the sheer size of the evolution data. Hundreds of versions of thousands of files are common in a single project. Size can be managed by building hierarchical clusters of these data, either manually or automatically. In case we use automatic clustering, a remaining issue is how to select a 'level of detail' from the many offered by the clustering, in order to get some desired trade-off between simplification and insight. Recently, we proposed a multiscale software evolution clustering (and visualization thereof) based on the notion of *evolutionary coupling* [Burch *et al.* 2005], which reduces indeed the visualization complexity [Voinea and Telea 2006]. However, it is not clear which level of detail, i.e. set of clusters, to select from the produced hierarchy, in order to get some desired insight. The user is left with the task of 'blindly' browsing through the cluster hierarchy until finding some 'interesting' decomposition.

3 Basic Visualization Model

We use history recordings stored in Software Configuration Management (SCM) systems as source of software evolution data. While different implementations of such systems may have specific ways of recording software development activity, we use a data model for describing it that is generic to all structure-based SCMs. The central element of the model is a *repository* R that stores all versions of all NF files in a project:

$$R = \{F_i \mid i = 1..NF\}$$

Each file F_i is defined as a set of NV_i versions:

$$F_i = \{V_{ji} \mid j = 1..NV_i\}$$

A version is a tuple containing several *attributes*: the unique version id, the time when it was committed to the repository, the author who committed it, a log message and its source code:

$$V_{ji} = \langle id, time, author, message, code \rangle$$

The first four elements (id, time, author, and message) are unstructured attributes. The code attribute can be structured in different ways, e.g. a set of lines, or set of functions, classes, modules, or other grammar constructs. We have applied this model successfully to describe evolution data acquired from CVS and Subversion repositories, the most popular Open Source SCM systems available.

To visualize these data we use the pixel-filling 2D representation from CVSgrab [Voinea and Telea 2006]. Each file is depicted along a time horizontal axis as a sequence of segments (Figure 1). Each segment shows one file version. The version creation time and the duration decide the position of the segment in the sequence and its length. The segment color shows version attributes, e.g. author ID, or functions defined on attributes, e.g. code size. To build complete visualizations of software evolution, we stack individual file representations on the vertical axis so they share the same time scale, and use the same color encoding. Figure 2 (top) shows a snapshots of CVSgrab for a two-year project with 300 files and 100 versions, where color shows the author ID. We use geometric shaded cushions [van Wijk *et al.* 1999] to segregate between vertically stacked file stripes. The order on the vertical axis can be set by users via sort and clustering operations to target different usage scenarios. Clusters are segregated using an additional shaded cushion layer. Figure 2 (bottom) shows a decomposition of the same system visualized in Figure 2 (top). The dark areas are the cushion borders of the decomposition clusters.

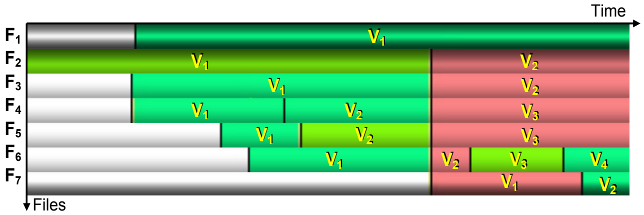

Figure 1: Visualization model

The above visualization model scales very well with industry-size projects and is useful to conduct a basic assessment of software evolution from the perspective of a given attribute.

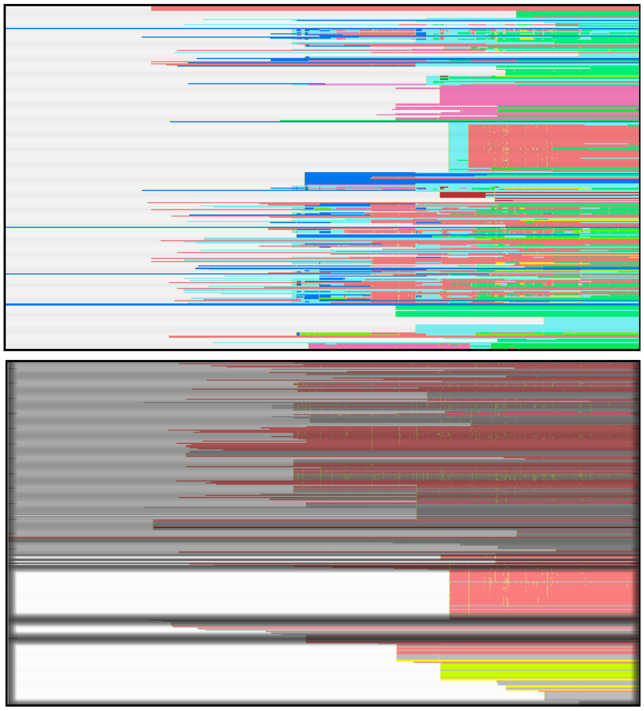

Figure 2: Evolution visualization with CVSgrab

However, this model cannot help (visually) correlate *several* version attributes, since just one is shown at a time. Moreover, detailed user feedback to our studies based on the above model showed that, while users see clustering as one of the most valued features of CVSgrab, they do not know which level of detail to choose from the ones produced by the clustering. We address these challenges by enhancing the basic visualization model with several techniques. These techniques enable users to correlate up to three version attributes at a time, and also easily navigate between various viewpoints on the software project evolution. Next, we present a new visual approach for navigating cluster hierarchies. Our approach shows an intuitive, interactive map of the clustering that enable users to easily choose relevant levels of detail and identify the importance of the cluster components. We describe all these approaches next.

4 Multivariate visualization

Software evolution data is multivariate. Every version of a file has a number of assigned attributes that characterize it: version ID, commit time, author ID, author commit comment, and version body, e.g. source code (see Section 3). Atop of these attributes, many functions can be defined, e.g. the code source size, the presence of a given word in the author comment, the membership to a given software release, and so on.

To assess the evolution of a software project, the distribution of such attribute values (and functions thereof) can be visualized, e.g. using the basic visualization model of CVSgrab. Each such visual distribution gives a *viewpoint* on the project evolution. More insightful information in the evolution can be further obtained through correlations across multiple view points. However, this typically requires visualizing the distribution of more attributes simultaneously. Moreover, the correlation making process is a dynamic activity, so a way to define, customize, and select correlation scenarios is needed.

To achieve these goals, we must address several challenges. First, the visualization model we use maps real-life projects of thousands of files with hundreds of versions to small-sized pixel strips. We must find effective ways to map several attributes on this small space. Secondly, we must find ways to enable users to construct the attribute mapping functions intuitively and quickly.

4.1 Texture Synthesis for Attribute Visualization

We depict multiple attributes on the same (small) space using a combination of color and hand-designed textures. Our approach resembles the one proposed in [Holten *et al.* 2005]. However, there are important differences. Holten *et al.* use a parameterized synthetic texture to encode one attribute besides the one encoded by color. Their texture model allows easy building of tiling textures that do not perceptually interfere with other shapes depicted in the visualization, hence do not artificially grab attention. However, this approach seems to be less suitable to encode more attributes in the same time and over the same quite small screen space. The inherent irregular texture aspect, due to the noise-based synthesis method, makes it difficult to distinguish between two (or more) superimposed patterns, e.g. used to map two (or more) attributes. We propose a different approach that allows encoding 2..3 attributes via superimposed, yet visually distinct, textures. For this, we give up the generality of the irregular texture synthesis proposed by Holten *et al.* We choose several hand-designed texture *patterns*, and encode attribute values in the pattern magnification factor. A careful pattern hand-design and selection ensures that these interfere as little as possible with each other. Figure 3 shows an example of two such textures using mirrored hatch patterns (A, B) to encode two attributes. Pattern A encodes the presence of a given word in the comment message associated with a version, and pattern B encodes the author of that version. Figure 3 shows the evolution of four files across two versions. Color encodes file type. We can easily see that file F_3 has only attribute values encoded by pattern A, and file F_4 only attribute values encoded by pattern B. File F_1 has values encoded by both patterns, since drawn with the crosshatch combination of patterns A and B. File F_2 has none of the two attributes, i.e. is not committed by the sought author, nor does it contain the sought word, as it shows no texture.

Further analysis of Figure 3 shows more correlations. Pattern B appears in both V_1 and V_2 of F_4, so F_4 was committed by the same author twice. Pattern A has different values for versions V_1 and V_2 of F_3, so different words of the searched set appear in them. File F_1 is committed by the selected author (has pattern B), and contains different searched words in its two versions. Comparing F_1 with F_3, we see that the search hits are the same in the respective versions of the two files. Version V_1 of F_1 is more similar to the (a_1) value of pattern A than to the (a_2) value. Hence, one could conclude that version V_1 of F_1 contains the word (a_1) and is committed by author (b_1), and version V_2 contains the word (a_2) and is committed by the same author (b_1).

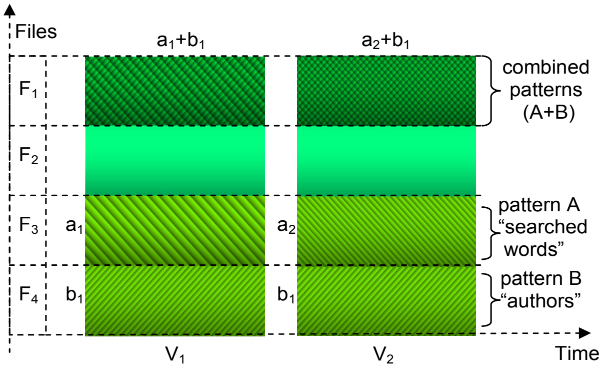

Figure 3: Combining texture patterns to show several attribute values.

Figure 4 shows a second example of visualizing several attributes. Here, we use bubble patterns to indicate revisions belonging to a given system release, and a diagonal hatch pattern for files containing the word 'tag' in their commit logs. Color shows author ID. We can easily spot files belonging to the selected release and containing the word 'tag'.

Preliminary user studies show that superimposing textures obtained by scaling perceptually largely different patterns can encode two, sometimes three, attributes simultaneously. The most effective use hereof is for showing nominal attributes with a small value range, e.g. file types, search hits from a small word set, or author IDs. Indeed, superimposing textures, even when carefully chosen, decreases the individual pattern resolution, which makes it quite hard to map continuous values with *high* precision.

After experimenting with several patterns, we designed a small set containing vertical, horizontal, and the two diagonal hatches, and also a 'bubble' pattern.

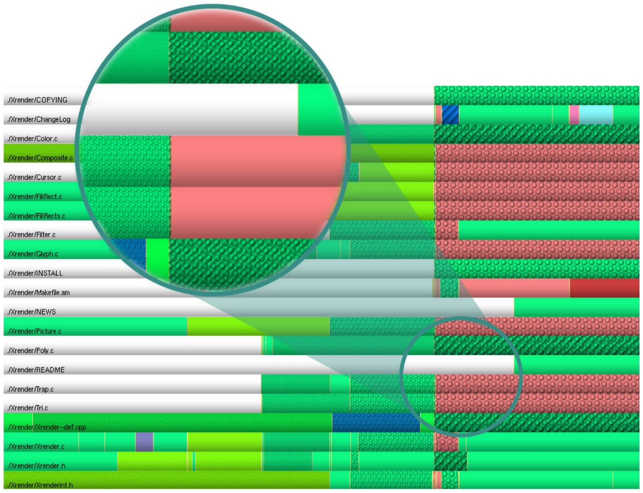

Figure 4: Texture composition: spheres = selected revisions, hatches = word 'tag' in comment

This set is quite effective since (a) the interference between any two patterns in this set is quite small; (b) the patterns are easily distinguishable, even when drawn on small areas (c) and/or scaled to small resolutions (d).Yet, there is a limit to how small an area we can texture and still see the patterns. This minimal size seems to be around 25*25 pixels on a normal 19 inch screen.

4.2 Navigation in Viewpoint Space

Our second challenge is to find an intuitive and easy way to define, customize, and navigate between different evolution views. We define a *view* as a function $f_i(\{V_{ij}\}) \rightarrow$ *Textures* × *Colors* that associates a color *rgba* ∈ *Colors* and texture $t = (scale, pattern)$ ∈ *Textures* to every file version V_{ij}., based on its attributes. Next, we use the preset controller mechanism proposed in [Van Wijk and Overveld 2000], which works as follows. Given some 2D points p_i that correspond to the views f_i, and an 'observer point' p, the user can define custom views f

$$f(V)=\sum_i d(p,p_i)f_i(V)\Big/\sum_i d(p,p_i)$$

where $d()$ is some inverse distance function, e.g. $d(x) = 1/(1+x^2)$. The custom views are generated by moving either p or p_i with the mouse in the preset controller widget.

We now refine this mechanism to make it more effective for software evolution visualization, as follows. To give users better feedback about the way each view influences the final image, we draw isolines around the *observer* glyph. This helps measuring the observer-view distance and hence estimate the 'strength' of a given view. We saw that this matches closely the way users build visualizations: It is not important to specify the *exact* contribution of one view in the final visualization, but rather to indicate the *relative* contribution of all involved views.

A second addition we propose is to use glyphs parameterized by *view attributes.* The idea is to draw some intuitive metaphor on the glyphs that suggests what kind of visual mapping the preset associated with that glyph does, so the user knows what to expect when moving the controller towards that glyph. For this, we applied design principles validated in the gamming industry by products such as Microsoft's Age of Empires (see [Age of Empires]), where various attributes (e.g. offence, defence, quality, and life values of soldier figures are drawn on a small screen area with a few colors). Figure 5 illustrates our solution on a preset controller scenario having seven possible views. Only two views contribute to the visualization, i.e. *authors* and *search text*, as the other views are beyond the furthest observer isoline.

The *authors* view colors version segments in the evolution visualization as function of the ID of the user that committed the version. The *authors* glyph contains a number of colored squares, one per user, showing the users' colors. The *search text* view colors the file versions that contain a given string in their associated commit comment. Several strings can be searched for at the same time. Each string has an associated color. Versions that contain several strings are colored with a special color (red). Versions that do not contain any of the searched strings are grayed out.

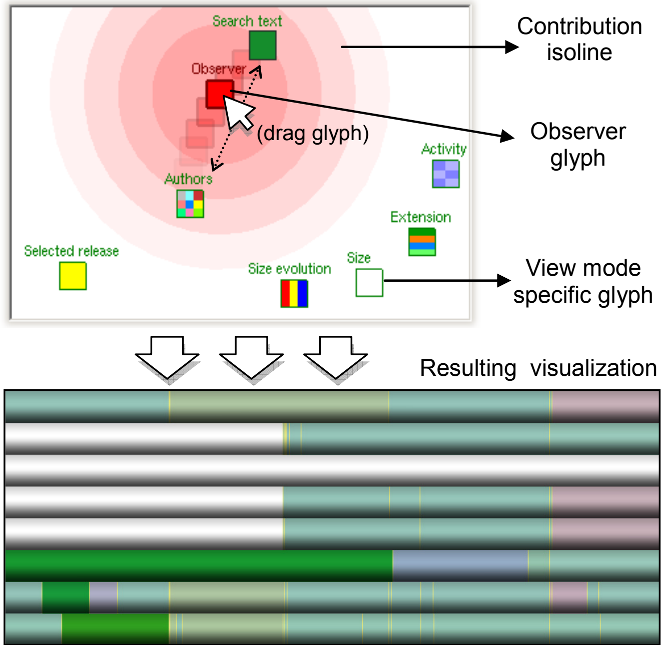

Figure 5: Preset controller based navigation among possible evolution views

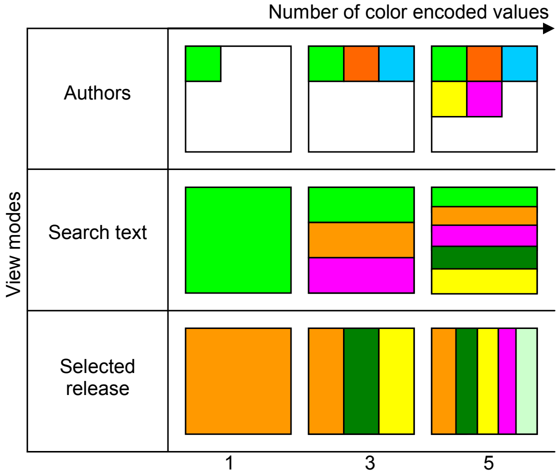

Figure 6: Parameterized glyphs for view mode identification

The associated glyph of the *search text* mode shows a vertical strip for every string in the search list, colored with the string's color. In general, we design the glyphs as small strip treemap-like areas with cells that show the colors the mapping *f* of that respective glyph can generate (Figure 6). For textures, we use a similar approach. Clearly, this approach works well only if the cardinality of both *Colors* and *Textures* is small (e.g. under 20).

As the observer is closer to the *search text* glyph (Figure 5), the final visualization will be mainly influenced by this view. The glyph associated with *search text* has only one color, i.e. green, so we search for only one string. Hence, the search hits will appear as saturated green in the resulting evolution visualization. The second active visualization mode, i.e. *authors*, has a smaller effect as its glyph is further away from the observer. Hence, the authors' colors will be less saturated, yet visible enough to distinguish between different authors or identify specific ones. The *authors* glyph contains a large number of colored squares indicating the user should expect a large number of authors to show up.

Preliminary user studies indicate that our modified preset controller is a very intuitive and fast way to understand and create the attribute mapping used in our visualization. Although only one attribute can be mapped to color at a given time, cross-view correlations are still possible. They are enabled by the seamless and fast transition between different views. By repeatedly shifting the observer's position between several views, one can correlate the color determined by the current predominant view with the previous color, stored in the 'short term memory'. Seamless transition between colors by means of blending helps focusing user's attention on an area of interest, as one is less distracted by sudden changes in other parts. Conversely, the repeated shifting of the observer glyph helps refreshing the short term memory.

5 Multiscale visualization

As already stated, one of the most valued features of the CVSgrab tool lets users to dynamically specify the file order on the vertical axis via sort and clustering operations. Industry-size projects can contain thousands of files. Following the evolution of each individual file and correlating it with the evolution of the others is simply too complex. Clustering lets users group files that are similar from a certain perspective. Clustering has two roles. First, it lets users look at less data to investigate evolution correlations, reducing the complexity problem. Second, it offers system decomposition, facilitating the software understanding process when no such decomposition is available.

The visualization model from CVSgrab uses an agglomerative bottom-up algorithm to perform clustering. It *uses evolutionary coupling* [Burch *et al.* 2005] to cluster entire file evolutions, based on the number of similar commit moments, i.e. moments that are close to each other in time. Clustering uses this distance function to find the two closest clusters in a project and merges them in a new cluster. The procedure starts with every file F_i in a cluster and is repeated until a single cluster remains. A binary system decomposition tree is created. Its leafs are all files $\{F_i\}$ in the project and its nodes are the computed clusters. Denoting the *file-set* of a node by $T(n)$, i.e. the set of leafs which are descendants of $T(n)$, a *decomposition* of the system is a set of nodes N_{sys} has the properties $\bigcap_{n\in N_{sys}} T(n) = \phi$, $\bigcup_{n\in N_{sys}} T(n) = \{F_i\}$, i.e. it is a partition of $\{F_i\}$. Once the decomposition tree is computed, the main question is: How do we let users construct and select *meaningful* decompositions?

The CVSgrab tool proposes a quite simple mechanism for selecting a decomposition: N_{sys} contains all nodes from the tree hierarchy at a given depth d_{root} from the root (Figure 7a). We shall call this the *isodepth* method. Users can specify d_{root} by moving a slider. Next, files are sorted on the vertical axis so that files in a cluster are contiguous, and all clusters in N_{sys} are visualized using plateau shaded cushions blended over the actual rendering of the file segments. Visualizing clusters by shaded cushions is quite effective, so we shall not alter this mechanism. However, specifying the decomposition via *isodepth* is far less effective. The agglomerative clustering algorithm used can generate highly unbalanced trees, which then leads to the coexistence of both very small ('skinny') and very large ('fat') clusters in the same decomposition (see e.g. Figure 2, bottom, and Figure 12a). Moreover, users have no feedback on how to choose the decomposition level, e.g. there is no indication of how similar are files in the clusters of a certain level.

We address these drawbacks by two new decomposition selection methods (Sec. 5.1). Next, we present a new technique to visualize the whole range of decompositions, so that users can select an appropriate level of detail, based not only on a desired complexity reduction, but also on a desired cluster relevance (Sec. 5.2)

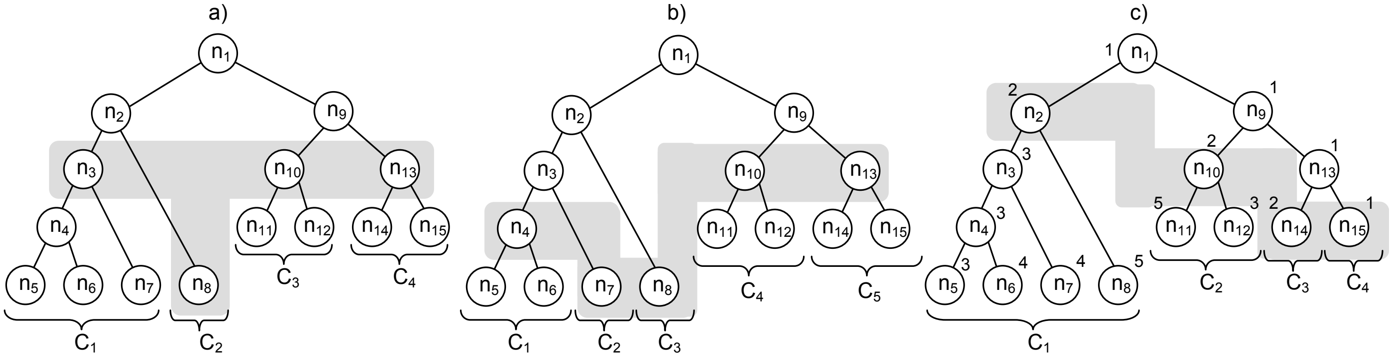

Figure 7: Decomposition selection methods: (a) *isodepth*; (b) *isometric*; (c) *isorelevance*. Gray regions show selected nodes. Numbers next to nodes in (c) give node relevance (larger means better).

5.1 Decomposition Selection Methods

To deal with the highly unbalanced trees generated by the used clustering algorithm, we propose two new approaches for selecting a cluster decomposition N_{sys}.

In the first approach, nodes are selected from the decomposition tree so that their file sets have similar cardinalities. We call this the *isometric* decomposition selection (Figure 7.b). The user can interactively specify a desired cardinality *C*. The decomposition tree is traversed in pre-order. If the cardinality of the current node's file-set $T(n)$ is smaller than *C*, *n* is added to the decomposition N_{sys}, and its children are skipped. If $T(n) > C$, the traversal continues with the children of *n*. Although this method does not guarantee that the selected clusters have exactly identical cardinalities, it provides in practice much better results than the *isodepth* method (see Figure 10.b).

The second decomposition selection we propose uses the cluster relevance (Figure 7.c). Every cluster node gets a relevance factor. The tree is traversed in pre-order, and at each step the relevance of the current node $R(n)$is compared to a user-selected value *R*. If $R(n) > R$, *n* is added to the selection and its children are skipped, else traversal continues with the children of *n*. In the resulting decomposition N_{sys}, most clusters have a similar, though not guaranteed equal, relevance with the reference value (Figure 10.c).

We call this the *isorelevance* cluster decomposition selection. Its most important part is the calculation of the relevance factor of each node. We can use as relevance the cluster cohesion, given by the cluster diameter, i.e. the distance between its two children. A drawback of this approach is that the relevance of the children does not propagate to their parents, i.e. highly relevant nodes can have irrelevant children. Clearly, a carefully designed distance metric and cluster merging criterion can take care of this problem. Another approach to ensure 'relevance inheritance' is to compute node relevance as the size-weighted average of the children relevances. Hence, the average may be biased with the node diameter so that nodes with highly relevant children are less relevant when the distance is large, compared to cases when the distance is small. The node relevance can be recursively computed using the formula:

$$R(n) = B(d_n) \times \frac{R(n_{c1}) \times |T(n_{c1})| + R(n_{c2}) \times |T(n_{c2})|}{|T(c_{c1})| + |T(n_{c2})|}$$

where d_n is the diameter of the current cluster *n*, n_{c1} and n_{c2} are its two, and $B(d_n)$ is a bias factor that depends on the diameter.

We applied the two decomposition selection methods presented above only on the decomposition tree given by the bottom-up agglomerative clustering algorithm used by CVSgrab. However, it is clear these methods can be applied on the tree resulting from any clustering algorithm.

5.2 Navigation in Decomposition Space

The second issue we address is the missing link between the desired decomposition level of detail (LOD). Both the method used by CVSgrab and the *isometric* decomposition selection we propose above work by asking the user to select a desired size simplification LOD. However, they do not indicate how relevant the selected clusters are. The *isorelevance* method allows the user to specify a desired relevance level and provides a decomposition that tries to closely match that level. However, the user still has to guess a 'good' value for the relevance. In practice, we saw that users needed to continuously adjust the input parameter until a compromise is reached between relevance and size simplification. To assist the user in making a good choice both for the size simplification LOD and the cluster relevance, we propose a new visualization: The *cluster map*. This combines a classical value-selecting slider with a 2D map of all the available decompositions (Figure 8). The horizontal axis maps the LOD (size simplification for the *isodepth* and *isometric* methods, or the relevance value for the *isorelevance* method). The vertical axis depicts the cluster decomposition for every value on the horizontal axis. Every cluster decomposition is drawn as a vertical stack of cushioned rectangles, all stacks having the same width. The height of each rectangle is proportional with the number of files contained in the associated cluster. Intuitively, each stack in this visualization is actually a mini-map of the plateau cushions used in the main visualization (e.g. see Figure 12) to show the complete system decomposition. A blue-white-red colormap encodes the relevance (low to high) of each cluster drawn as a rectangle.

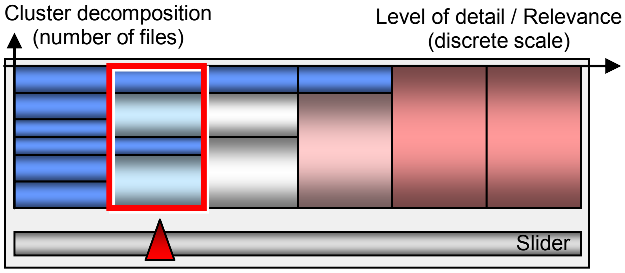

Figure 8: The cluster map widget. Clusters are drawn as cushioned rectangles. Color encodes relevance.

The widget principle (shown in Figure 8) enables users to quickly identify and make a compromise between the desired simplification level and cluster relevance. Also, it enables the user to select the desired decomposition via the slider at the bottom. Furthermore, users can correlate the clusters depicted in the main evolution visualization with the ones in the widget and therefore identify their relevance.

However, drawing all clusters of typical software decompositions in the cluster map leads to aliasing, as the cluster cushions easily become less than one pixel high. This creates the false impression that there are no clusters on the finer levels of the cluster map, e.g. at the left of Figure 9a.

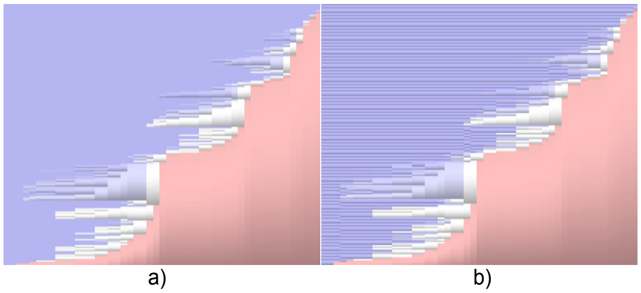

Figure 9: Cluster map without (a) and with (b) antialiasing

We solve this by drawing, on every level of the cluster map, only those clusters whose screen height exceeds 3 pixels, since this is the minimal height at which cushion textures are distinguishable. As shown in Figure 9b, small clusters are now clearly visible. An added bonus of this is that rendering the cluster map takes now constant time for arbitrarily large hierarchies.

Figure 10 shows an example of 2D cluster decomposition maps corresponding to different selection methods, for a project with 28 files spanning across up to 21 versions. All horizontal axes are normalized and sampled with a rate of 1:20, i.e. the horizontal axis displays, and allows the selection of, 20 values uniformly distributed between 0 and 1.

Figure 10a illustrates the cluster map for the *isodepth* method [Voinea and Telea 2006]. The horizontal axis gives the normalized level of detail: 0 is for 100% detail, i.e. every file is a cluster, 1 is for 0% detail, i.e. the whole project is seen as one cluster. We can now easily see that at *most* levels very small clusters coexist with large ones, so we can conclude that, for this particular project, the distance-to-root clustering is not that good.

In Figure 10b, the *isometric* selection method was used. Similar to the previous case, the horizontal axis gives the normalized level of decomposition detail. We can see that, while at most levels the cluster sizes are similar, there are few cases of extreme size differences. The cluster decomposition selections returned by this method are easier to follow.

In Figure 10c, the *isorelevance* selection method is used. The horizontal axis gives the normalized relevance: 0 is for maximum relevance, 1 for minimum. We can apply here the same reasoning as above for the *isometric* selection.

In all cases presented in Figure 10, the cluster decomposition set does not vary for every LOD value on the *x* axis. There are large LOD intervals that have the same set (see dotted highlights). In general, such long 'constant' intervals border an important system decomposition step in terms of number of displayed clusters (Figure 10.a), maximum number of files per cluster (Figure 10b) or cluster relevance (Figure 10c). The latter case is particularly important. A carefully designed relevance factor can show passing from highly coupled system components, e.g. classes, to more loosely coupled ones, e.g. packages, and therefore can give an insightful, intuitive, and simple structural view on the system.

It is true that our cluster map alone cannot show which are the *meaningful* partitions to visualize for a given system and problem. However, it shows which are those values of the level-of-detail parameter where relatively important clustering events, i.e. system simplifications, take place. The user can decide to select these levels and visualize the corresponding decompositions, without having to browse all the (usually quite many) level-of-detail values. The cluster size distribution in the cluster map shows what kind of visualization to expect if selecting that level-of-detail.

6 Usage Scenarios

To validate the techniques proposed in Section 4 and Section 5 (texture synthesis for attribute mapping, enhanced preset controller, *isometric* and *isorelevance* clustering methods, and the cluster map) we implemented them atop of the CVSgrab tool. CVSgrab allows users to retrieve software evolution recordings from CVS and Subversion repositories and visualize them using the basic version-versus-file 2D visualization model presented in Section 3. The resulting tool was used by us and other users to visualize the evolution of a number of industry size projects.

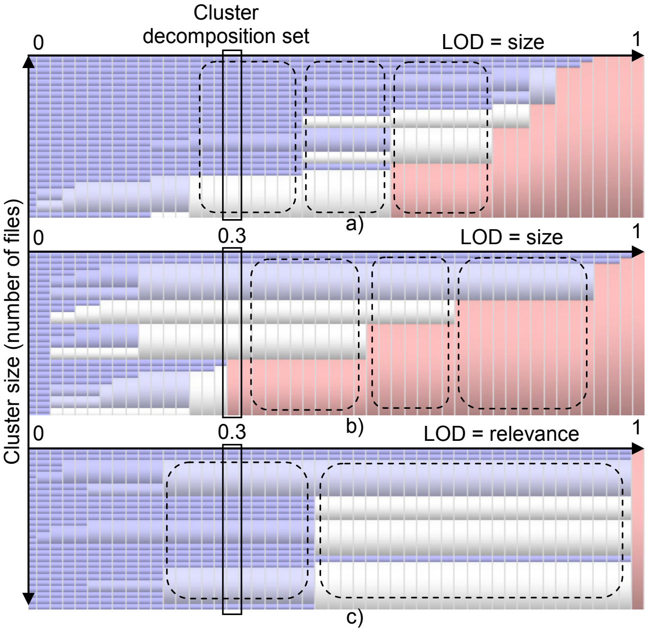

Figure 10: 2D cluster map using: (a) *isodepth* selection; (b) *isometric* selection; (c) *isorelevance* selection. Clusters are drawn as cushioned rectangles. Color shows relevance. Dotted highlights show intervals with the same cluster decomposition.

We next present two relevant use cases that profit from the novel techniques, and the associated findings.

6.1 Usage Scenario 1: Complex Queries

The texture-based attribute encoding we added to CVSgrab lets user visualize up to four attribute values at the same time (3 textures + 1 color). This supports complex evolution queries. The *preset controller* takes the correlation possibilities one step further. Figure 11 presents an example of complex query applied on the evolution of *MagnaView*, a commercial software package containing 112 versions, each of 312 files, over 16 development months (see [MagnaView]). The image answers the query: "What versions of *GUI specification files,* belonging to release *549,* and containing the word *bug* in the associated log message, have been committed by developer *tomasz?*" We answered this query with the following techniques:

- a diagonal hatch pattern texture in the direction NE-SW to show versions containig the word *bug* their commit message
- a diagonal hatch pattern texture in the direction NW-SE to show versions that belong to release *549*
- an author ID-to-color view mode, with red encoding *tomasz*
- a filetype-to-color view mode, with gray for *GUI specification files*
- a preset controller to switch between the two color view modes

Figure 11a depicts a zoomed-in area of the evolution visualization using the author ID view mode. The highlighted versions are possible candidates for the query above. The cross-hatch texture pattern shows they both contain the text *bug* and belong to release *549*. Moreover, red indicates the versions have been committed by *tomasz*. Using the preset controller to rapidly change between the two view modes, one can see that only one of the candidate versions is a GUI specification file: *UEditViewForm* (highlighted in Figure 11b). Many other similar scenarios and use cases exist, we do not present them for lack of space. In conclusion, using the proposed multivariate visualization features, one can easily give answers to complex queries by narrowing down a set of candidate solutions.

6.2 Usage Scenario 2: System Decomposition

Our second main improvement is the addition of the *isometric* and *isorelevance* methods for cluster decomposition selection, and the introduction of the *cluster map* widget. Figure 12 shows the use of the *cluster map* and presents the decomposition results for the VTK graphics library (see [VTK]). VTK is an open source project of over 2700 files, written by 40 developers in over 11 years, spanning across 180 versions. Figure 12 left shows the *cluster map* widgets for the project evolution for the *isodepth* selection method (a), the *isometric* selection (b) and the *isorelevance* selection (c). All widgets use a red-to-blue gradient color map to show (low to high) cluster relevance. In each widget, the chosen selection is indicated by a red rectangle.

Figure 12 right shows the results of the chosen cluster selection in the main evolution visualization, i.e. clusters are drawn as plateau cushions over their respective files. For the chosen LOD, the visualization in Figure 12.b shows just a few cushions, most of them of similar size, which makes the decomposition easy to understand. We can easily identify four main system components, a fifth being also rather visible. In contrast, the *isodepth* decomposition (Figure 12a) has many clusters of very different sizes on the same LOD level. Only thee main components can be identified, and a large part of the system is still impossible to categorize. Figure 12c shows a system decomposition using the *isorelevance* method. Although there are more clusters, with more similar sizes than in Figure 12a, this decomposition is not as easy to follow as the *isometric* one in Figure 12.b. However, as shown by the associated cluster map widget, all selected clusters have a similar relevance. This is not the case with the other two methods which return clusters that cover the entire relevance spectrum.

The cluster map widgets (Figure 12 left) give also an indication on the performance of the three cluster decomposition methods. We can see that the *isodepth* selection method spawns in general very few clusters and most of them evolve very fast into large ones, as the LOD increases. In contrast, the *isometric* selection tries to balance the size of the displayed clusters. For coarser LOD values, the clusters grow and equalize in size, which leads to larger intervals of constant decomposition. Finally, the *isorelevance* selection contains also large intervals of constant decomposition. They are caused by a project specific number of relevance thresholds. On the several projects we checked this method in practice, these intervals corresponded to meaningful structural decomposition views on the system.

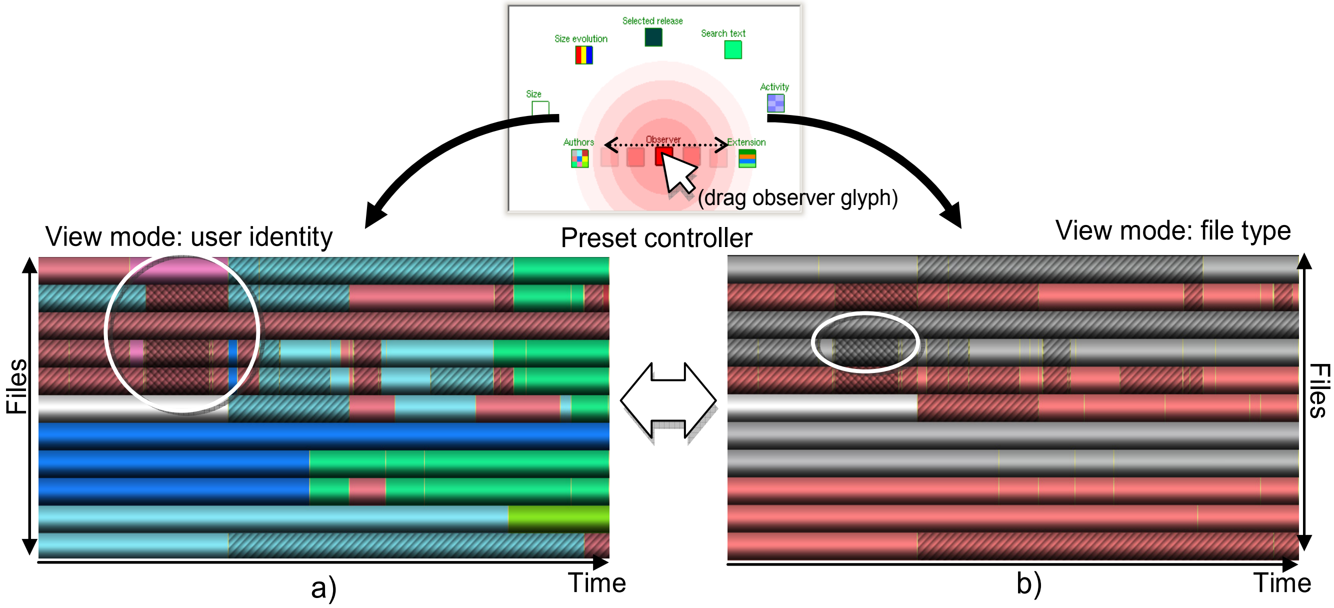

Figure 11: Complex queries usage secenario. Blended textures and colors show a set of possible solutions based on three attributes. Using the preset controller, a fourth attribute can be checked and the set of possible solutions (a) is reduced to one version (b).

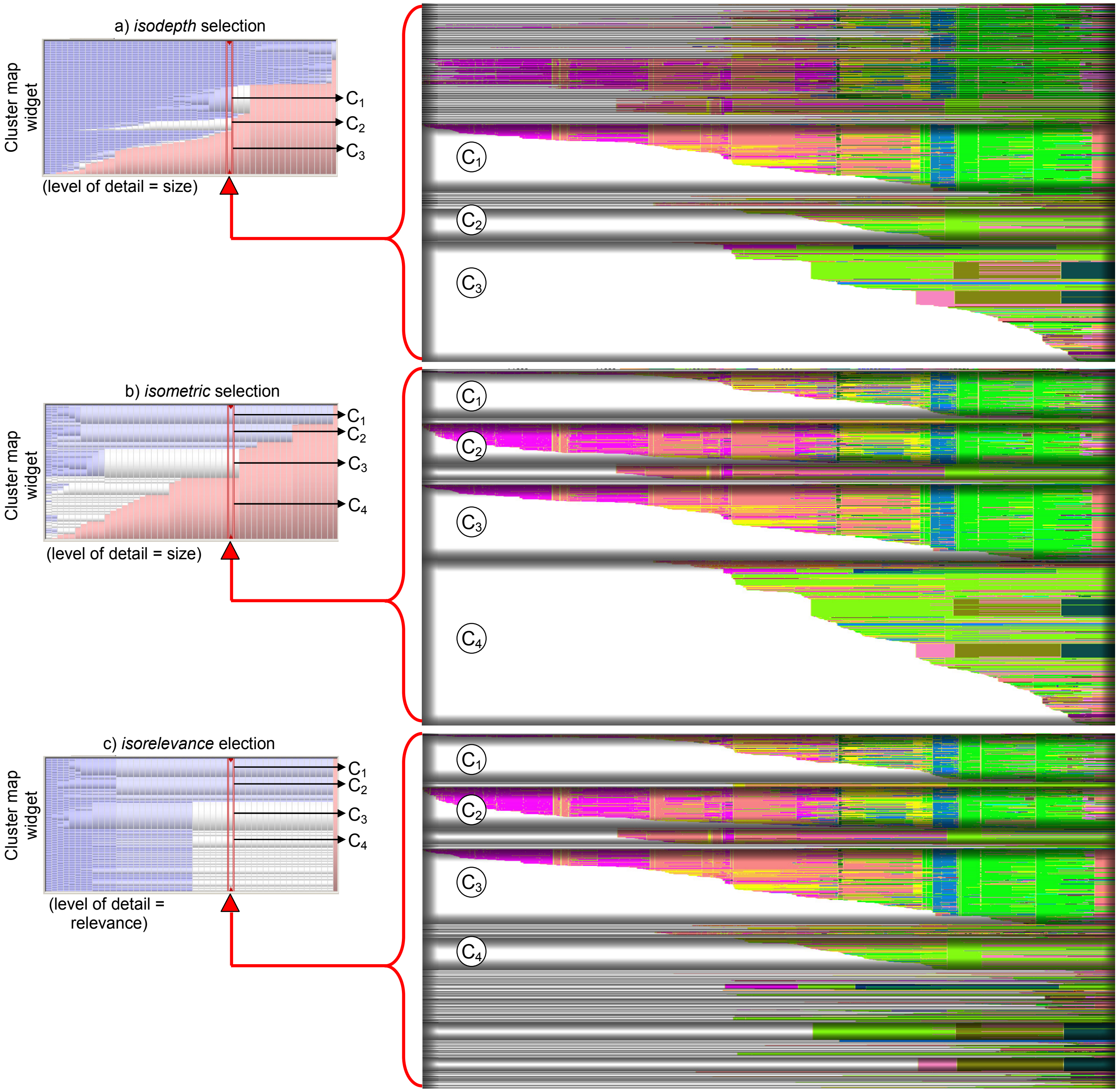

Figure 12: Usage scenario 2 - cluster decomposition selection

In conclusion, the *isometric* and *isorelevance* selection methods are better alternatives to the *isodepth* method. The *isometric* method generates views that are easy to comprehend, while the *isorelevance* method generates similar- relevance cluster decompositions. All these aspects are visible in the cluster map widget. The role of this widget is thus threefold. First, it shows to the user see a global picture of the system decomposition, thereby letting one assess the quality of meaningfulness of a decomposition method. Second, it shows constant intervals of the decomposition, which very often correspond one-to-one to different system structurings. Third, it was a useful instrument for us to compare the quality of various decomposition methods and see the effect of tuning the clustering metrics.

7 Conclusions

In this paper, we present a number of improvements to the basic visualization model for software project evolution presented in [Voinea and Telea 2006]. First, we adapt and extend two existing techniques to enable the visualization of more attributes at the same time over the same layout. We encode up to four attributes using color and hand-designed texture patterns, at the same time minimizing the visual interference. We enable making color-based correlations across several color-encoded attributes using an extended preset controller technique. Second, we propose two new methods for selecting decompositions of the software evolution: *isometric* and *isorelevance*. The *isometric* method yields easy to comprehend decompositions in which clusters have similar sizes. The *isorelevance* method generates decompositions in which clusters have similar relevance. We enable users to see an entire decomposition and select a meaningful level from it using a new widget: the cluster map. This widget enables users to quickly assess the results of a selection method in the context of a specific project, and choose the cluster decomposition that matches some desired compromise between level of detail and relevance. We incorporated all proposed techniques in the CVSgrab and used it to analyze the evolution of several industry-size projects, both open source and commercial software projects, hosted on CVS and Subversion repositories. Our tool is available at: **http://www.win.tue.nl/~lvoinea/VCN.html**

The tool scaled very well with the large size of the projects and their long history. Relevant assessments have been made on projects containing thousands of files representing the effort of tens of authors during more then 10 development years.

One open issue that has not been solved yet concerns color selection in the small glyphs in the preset controller that depict the view modes. As the value range of the color-encoded attributes increase above 15 distinct values, it is hard to show all these in a small spatial area. We want to further investigate this issue and look for better alternatives that use color as main segregation mechanism. We also consider better alternatives for computing the relevance factor. We believe a carefully designed relevance measure can lead to useful structural decompositions from the project evolution information. This would enable a static structure recovery of virtually any type of project without the need for more complex to design and use, language-dependent, reverse engineering and analysis tools.

8 Acknowledgements

We thank Danny Holten for his comments and suggestions regarding the use of texture in multivariate visualizations and Roel Vliegen from MagnaView for providing the evolution data and taking part in our user studies.

9 References

Age of Empires: www.microsoft.com/games/empires/

Burch, M., Diehl, S., Weißgerber, P., 2005. Visual Data Mining in Software Archives. *Proc. ACM SoftVis*, ACM Press, 2005, pp. 37 – 46

Eick, S.G., Steffen, J.L., Sumner, E.E., 1992. SeeSoft - A Tool for Visualizing Line Oriented Software Statistics. *IEEE Trans. on Software Engineering*, 18(11), 1992, IEEE CS Press, 1992, pp. 957 – 968

Froehlich, J., Dourish, P., 2004. Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. *Proc. ICSE*, IEEE CS Press, 2004, pp. 387 – 396

Gall, H., Jazayeri, M., Krajewski, J., 2003. CVS release history data for detecting logical couplings. *Proc. IWPSE 2003*, IEEE CS Press, 2003, pp. 13–23

Holten, D., Vliegen, R., van Wijk, J. J., 2005.Visual Realism for the Visualization of software Metrics. In Proc of VISSOFT 2005, IEEE CS Press, 2005, pp. 27 – 32

Interrante, V., 2000. Harnessing Natural Textures for Multivariate Visualization In *IEEE Computer Graphics and Applications*, 20(6), IEEE CS Press, 2000, pp. 6-11

Lanza, M., 2001. The evolution matix: Recovering software evolution using software visualization techniques. *Proc. Intl. Workshop on Principles of Software Evolution*, ACM Press, 2001, pp. 37–42

MagnaView: www.magnaview.nl

Pinzger, M, Gall, H., Fischer, M., Lanza, M., 2005. Visualizing multiple evolution metrics. *Proc. ACM SoftVis*, ACM Press, 2005, pp. 67 – 75

Shenas, H., Interrante, V., 2005. Compositing Color with Texture for Multi-Variate Visualization. In *GRAPHITE 2005*, pp. 443 – 446

van Wijk J.J, van de Wetering H., 1999. Cushion Treemaps: Visualization of Hierarchical Information. *Proc. IEEE InfoVis,* IEEE CS Press, 1999, pp. 73 – 78

van Wijk, J.J., van Overveld, C.W.A.M., 2000. Preset Based Interaction with High Dimensional Parameter Spaces. Presented at Dagstuhl Seminar Scientific Visualization, 21-26 May, 2000

Voinea L., Telea A., van Wijk J.J., 2005. CVSscan: Visualization of Code Evolution, *Proc. ACM SoftVis*, ACM Press, 2005, pp. 47 – 56

Voinea, L., Telea, A., 2006. CVSGrab: Mining the History of Large Software Projects. In *Proc. EUROVIS 2006*, IEEE CS Press, to appear.

VTK: www.vtk.org

Weigle, C., Emigh, W., Liu, G., Taylor, R., Enns, J., Healey, C., 2000. Oriented sliver textures: A technique for local value estimation of multiple scalar fields. In *Graphics Interface*, May 2000, pp. 163 – 170

Wu, J., Spitzer, C.W., Hassan, A.E., Holt, R.C., 2004. Evolution Spectrographs: Visualizing Punctuated Change in Software Evolution. *Proc. IWPSE*, IEEE CS press, 2004, pp. 57 – 66

Zimmermann, T., Weißgerber, P., Diehl, S., Zeller, A., 2004. Mining version histories to guide software changes. *Proc. ICSE*, IEEE CS Press, 2004, pp. 429 – 445