Combined visualization of structural and metric information for software evolution analysis

Antonio Gonzalez
Universidad de Salamanca
Facultad de Ciencias, 37008
Salamanca, Spain
agtorres@usal.es

Roberto Theron
Universidad de Salamanca
Facultad de Ciencias, 37008
Salamanca, Spain
theron@usal.es

Alexandru Telea
University of Groningen
Institute of Mathematics and
Computer Science
Netherlands
a.c.telea@rug.nl

Francisco J. Garcia
Universidad de Salamanca
Facultad de Ciencias, 37008
Salamanca, Spain
fgarcia@usal.es

ABSTRACT

This paper discusses a proposal for the visualization of software evolution, with a focus on combining insight on changes that affect software metrics at project and class level, the project structure, the class hierarchy and the indirect class coupling. The proposed visualization supports several tasks: the comparison of structural information, including class hierarchies, across several revisions; uncovering collaboration patterns between developers; and determining which classes have been added or deleted to the project during the creation of a given revision. We discuss several design elements supporting these tasks, including interaction patterns and and linked views.

Keywords

Visualization of software evolution, structure and metrics visualization, visualization design

1. INTRODUCTION

Software evolution is the process of software change and improvement over years and releases [**?**]. Software evolution analysis is concerned with understanding software changes, their causes, and their effects [**?**]. Software evolution analysis enables project managers in decision-making affected by factors such as the dynamics of software quality measured by quality metrics; controlling the contribution frequency and contribution patterns of programmers to the software project for team and productivity assessments; and reporting activities to upper management. Evolution analysis supports developer tasks such as learning new code bases, comparing the actual and desired architectures of a product, and planning development activities. While the needs of project managers and developers may differ, both groups require methods and tools that enable one to compare and correlate the evolution of *structural* and *metric* information from a software project. Structural data includes project containment trees, class hierarchies, and entity dependency graphs. Metric data includes tens of attributes measured on the evolving entities, ranging from names, types, and IDs up to derived metrics such as size, complexity, cohesion and coupling.

While numerous visualizations have been proposed to get insight in the evolution of metrics and structural data, combining several such attributes in scalable, effective, and efficient views is still an ongoing endeavor. In this paper, we propose and discuss three visualization designs for the exploration and comparison of software project revisions, project structural and class hierarchy data, and the correlation of structural data with metrics defined at revision and class level. The design contributions are the proposal of two novel visualization designs: a circular granular timeline for the visualization of all revisions of a project, and a hierarchical representation for the project structure and class hierarchies.

This paper is structured as follows. Section 2 reviews related work. Section 3 discusses our proposed visualization designs. Section **??** discusses the proposed designs and outlines ongoing work directions.

2. RELATED WORK

I am going to complete this section today (27/05/2009).

3. PROPOSED VISUAL DESIGNS

Software visualization is arguably a perfect instance of the recently emerging *visual analytics* discipline, which studies the design of interactive visual tools that help analytical reasoning about a given body of data [**?**]. Central to this process is the creation of tools and techniques based on various design, visual perception, interaction, and cognition principles to support a given set of tasks on a given set of data types. Here, we apply this design methodology to create useful visual tools for a set of tasks concerning software evolution analysis, on structure-and-metrics data mined from software repositories. The remainder of this paper describes the created tools, addressed tasks, and supported data types.

Our overall design combines three linked views. In decreasing level-of-detail order, these are: a timeline view, a hierarchy view, and a parallel coordinates view for software metrics. The views and their interactions are described next.

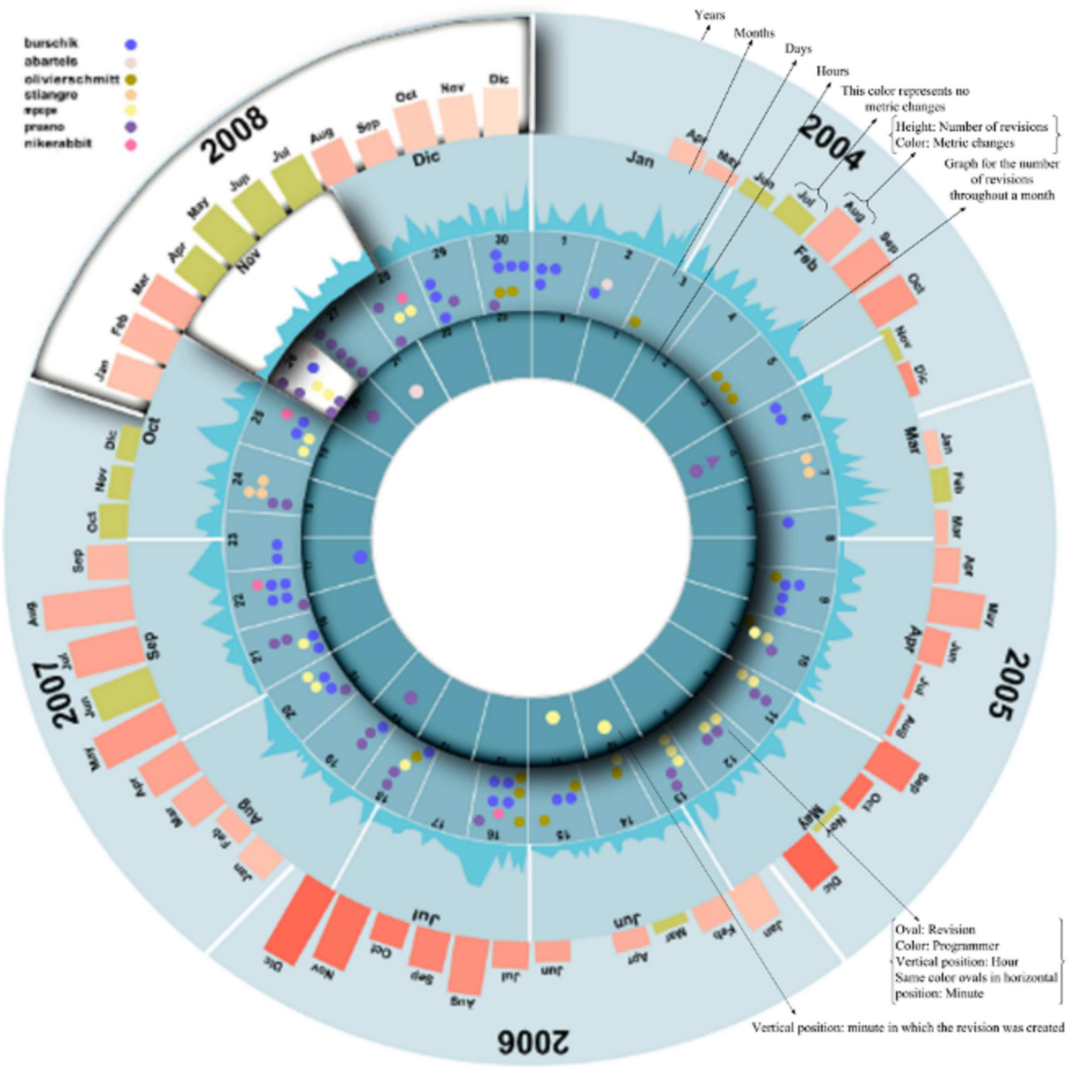

Figure 1: This timeline allows the granular representation of time.

3.1 Timeline visualization

This first visualization design uses a modified circular ring chart layout, to show an entire overview of the *time* dimension of a project (Fig. 1). Concentric rings show the different time scales that record change events, from coarse (years, outer ring) to fine (hours or finer, innermost ring). A similar layout was used by Holten *et al.* to show software project hierarchies [**?**]. This type of layout compactly presents large quantities of data and provides and overview + detail view. We augmented the basic design by drill-down selections, *i.e.* selecting a year to display data on its months, and the same for months-to-days and days-to-hours drill-downs. The details on the selected time-frame are displayed in the linked views discussed in the next sections.

Figure 1 illustrates this design for 4255 revisions of the project Freecol [**?**]. The user has selected November 26th, 2008, which is shown by highlighting the logical containment path day-month-year. As opposed to most applications using circular ring charts, we use the space within each chart cell to embed different types of visualizations that solve a task centered at that cell's level-of-detail, as follows. First, the 'years' ring cells embed bar charts that show two metrics for each month for each year. In Fig. 1, we map the number of revision to bar heights and a *change* in time of a user-chosen software metric of interest to a hue-luminance colormap. Light reddish colors indicate metric increases, while dark reddish colors indicate metric decreases. A separate golden hue indicates no changes in the metric. Similar colormap designs have been shown to be perceptually intuitive and better than pure luminance or hue colormaps in similar applications interested in showing metric fluctuations [**?**].

The 'months' ring embeds a different chart showing the number of committed revisions per month day using a height plot. Next, the 'day' ring shows revisions within each day, so this ring has 30 or 31 cells. For each cell, a matrix dot plot is drawn in polar (ρ, ϕ) coordinates. The radial ρ coordinate maps the creation hour of a revision. All revisions created within the same hour are placed at the same ρ but different angles ϕ. Finally, the innermost ring shows the placement of revisions by 'hour', so it has 24 cells. Within a cell, the radial position of the revision shows the creation minute. If more than two revisions were created in the same hour, the icons that depict them change, as *e.g.* for November 6th, 2008: the number of sides of the polygon depicts the variation of the metrics,

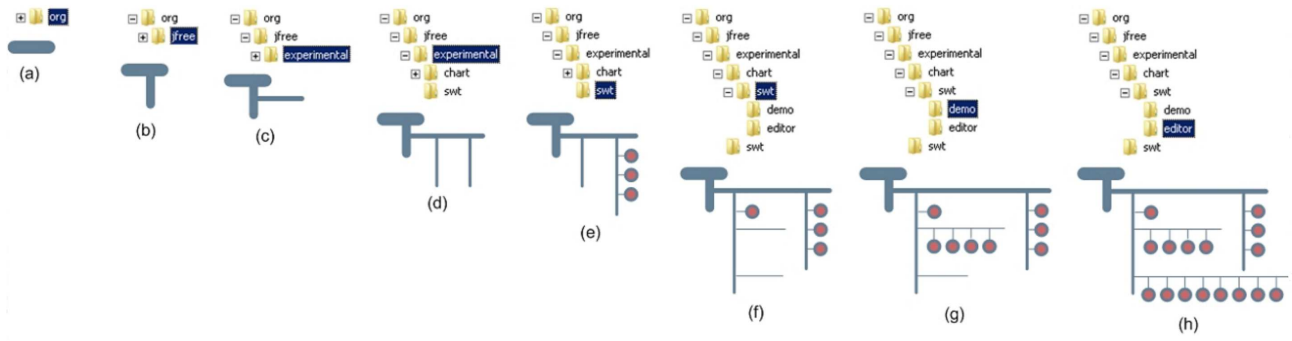

Figure 2: H-V tree navigation of a small software package. Lines represent sub-packages and circles show classes.

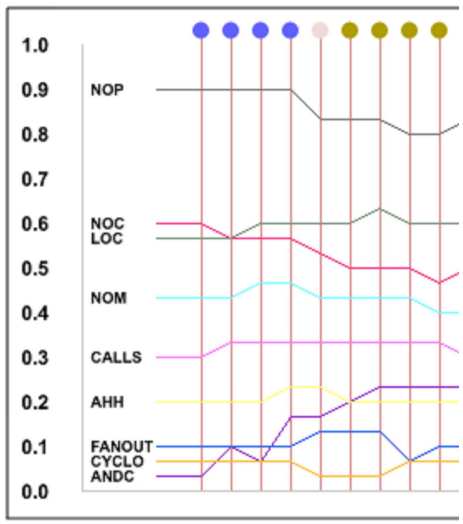

Figure 3: Parallel coordinates showing the metrics for revisions.

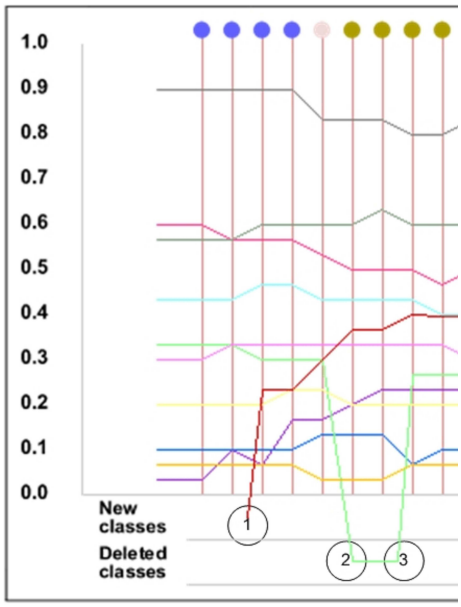

Figure 4: Parallel coordinates representing class metrics for revisions as well as the addition, deletion and re-addition of classes to revisions.

if more sides greater is the metric variation. Dot colors indicate, again, developer identity as for the 'days' ring.

3.2 Structure evolution visualization

The second visualization targets the task of analyzing the evolution of the structure of a given subsystem, which is selected by clicking a dot in the circular timeline overview (Section 3.1). For this task, we propose a variation of the H-V tree (a reference to Graph Drawing Algorithms for the Visualization of Graphs will be here), as shown in Figure 2. The advantage of this layout is a very compact usage of the 2D space. In this layout, horizontal or vertical lines represent non-leaf nodes, *e.g.* packages in a containment hierarchy or superclasses in a class hierarchy. Circles represent leaf nodes, *e.g.* functions in a containment hierarchy or classes without subclasses in a class hierarchy.

A second advantage of this structure visualization is that, combined with interaction, it allows the side-by-side comparison of a moderate number of hierarchies extracted from different project revisions. In our design, H-V trees can be navigated interactively by opening and closing non-leaf nodes. Figure 2 illustrates this, showing eight snapshots from the successively deeper drilling-down in the structure of a small software package (one version). The upper images show a classical file browser metaphor for the same navigation. Arguably, the H-V layout shows more information and more structural insight using the same amount of screen space. Exactly the same navigation can be used to compare different versions (not shown in the figure). This trades off more space for details in one or a few revisions (when opening many nodes) to showing more revisions along each other (when opening only a few toplevel nodes). This type of interaction supports common structural evolution analyses where one is interested to see the overall structure changes together with detailed changes in a few selected hierarchy sub-branches, all in the same view. If desired, events such as the addition or modification of entities can be emphasized by using different colors.

3.3 Metrics visualization

The third and final visualization targets the We propose the use of the well known parallel coordinates visualization for the representation of metrics. It is a multivariate visualization that uses vertical bars for representing variables and horizontal lines for an object that have associated several variable values. The idea behind this is that the user selects a period of time from the circular granular timeline and its associated revisions are displayed in the metrics

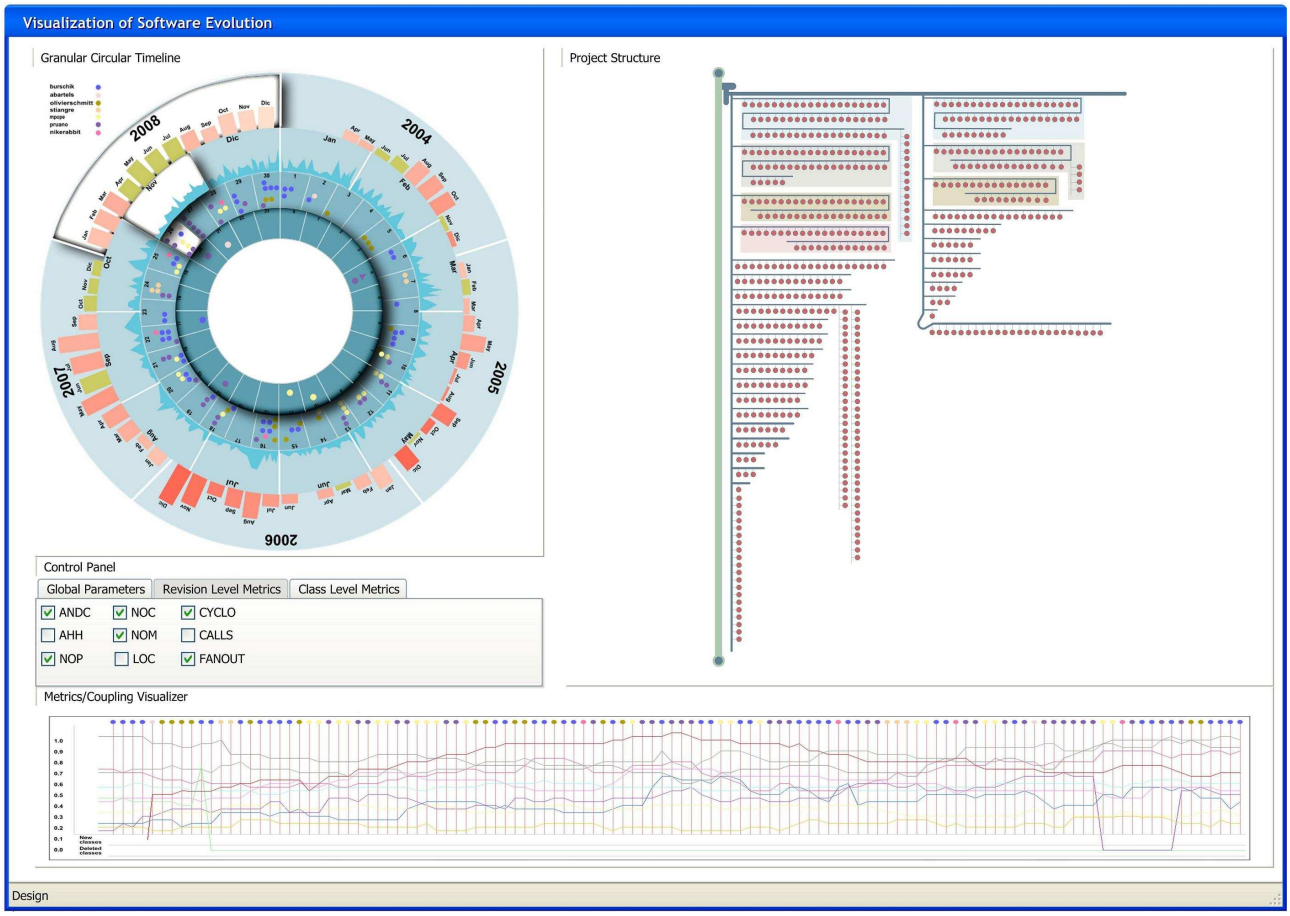

Figure 5: View of the visualization proposal integrating the three designs.

viewer. If the user selects to view the metric values for each revision, figure 3 is shown; the vertical bars represent revisions and the horizontal lines the metric values for the project at each particular revision. However, if the user is interested in analyzing the metric values associated to classes at each revision he/she can select to display figure 4. In this case vertical bars also represent revisions and horizontal lines classes. In addition, this representation also shows how classes are added (1), deleted (2) or created again and added (3) to a revision.

3.4 Discussion

Figure 5 shows a snapshot of our tool with the three linked views discussed so far. In visual analytics terms, the so-called interaction path that users follow in our tool starts with the selection of a *time period* from the timeline. The interaction path branches here. Managers, or users new to a project, will most likely ask for an overview of revision-level or class-level software metrics for the selected time period. Our current implementation supports typical object-oriented static analysis metrics such as size, number of members, LOC, cyclomatic complexity, fan-in, fan-out, coupling and cohesion [**?**], computed directly from the (Java) source code in the examined repository. The metric view in the lower panel shows the selected metrics' evolution. Developers or architects with detailed project knowledge and finer-grained maintenance or refactoring tasks will typically want to see the structure evolution across several revisions of an entity. For all selected revisions, the project hierarchy (packages-classes-methods in Java) or the inheritance tree are displayed using the H-V layout (Sec. 3.2). In addition, the user can select a class and the indirect coupling of that class is displayed. Moreover, when a user selects one of the visual elements in the indirect coupling representation the fragment of source code is displayed for a carefully analysis (this figure will will be included today).

This design offers several interaction possibilities, i.e. when the user selects a revision the other revisions created by the same programmer are highlighted in the circular granular timeline and the metrics visualizer, the user has the possibility of filtering metric by ranges, minimize packages, show the inheritance class path, and many other possibilities.

Although we have not conducted a formal validation of the effectiveness of our linked-view evolution visualization, we have designed and used a different type of evolution visualization for evolving software, targeting very similar tasks and users for a period of over four years, several tens of users, and around 15 academic and commercial analyses [**?**, **?**]. It is interesting to compare the two approaches. The metrics visualization in [**?**] is very similar in technique and purpose to the one presented here, except that in this paper we explicitly add the user identity, mapped to color, to the revision dots along the x axis. The timeline presented here is fundamentally different, *i.e.* it uses a circular layout with four hierarchy levels and different types of embedded visualizations and metrics at each level. In contrast, the timeline in [**?**] uses a flat 2D Cartesian layout with time along the x axis and files along the y axis; whereas visually more scalable, this layout does not of-

fer a multilevel aggregation of information as we do here. Finally, the structure evolution view shown here is new and supports finer-grained tasks that the design in [**?**] does not address yet.

4. CONCLUSIONS