University of Groningen

Combined Visualization of Structural and Metric Information for Software Evolution Analysis

Gonzalez, Antonio; Theron, Roberto; Telea, Alexandru; Garcia, Francisco J.

Published in:
EPRINTS-BOOK-TITLE

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2009

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):

Gonzalez, A., Theron, R., Telea, A., & Garcia, F. J. (2009). Combined Visualization of Structural and Metric Information for Software Evolution Analysis. In *EPRINTS-BOOK-TITLE* University of Groningen, Johann Bernoulli Institute for Mathematics and Computer Science.

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

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 viewing of source code correlated to 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 propose and discuss several design elements supporting these tasks, including interaction patterns and linked views.

Categories and Subject Descriptors

H.5 [INFORMATION INTERFACES AND PRESENTATION]:
User InterfacesScreen design (e.g., text, graphics, color)

General Terms

Design, Experimentation, Measurement

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 [3]. Software evolution analysis is concerned with understanding software changes, their causes, and their effects. Such analyses enable 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 software projects for team and productivity assessments; and reporting activities to upper management. Evolution analyses support developer tasks such as process improvement, fault prediction, productivity estimation, comparing the actual and desired architectures of a product, and planning future 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 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 four 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, as well as associating structure elements and source code. Within these, we propose 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. Together with these, we integrate two variations of existing evolution visualizations: a metric view and an indirect class coupling representation that integrates source code viewing.

This paper is structured as follows. Section 2 reviews related work. Section 3 discusses our proposed visualization designs. Section 3.5 discusses the proposed designs and outlines ongoing work directions. Section 4 concludes the paper.

2. RELATED WORK

In software evolution, just as other dynamic analysis, change events can be seen at different scales, or granularities [9]. Many visualizations address the problem of getting insight in time series [7]. However, this is not enough for software-related problems, as stakeholders need to correlate time events with data changes,

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

IWPSE-Evol'09, August 24–25, 2009, Amsterdam, The Netherlands.
Copyright 2009 ACM 978-1-60558-678-6/09/08 ...\$10.00.

Figure 1: Multilevel timeline view correlating commit events and evolution metrics

such as structural or metric changes [12, 11]. As such, researchers have investigated ways to combine time events with data changes in *timelines*, i.e. the representation or exhibit of key events within a particular historical period, often consisting of illustrative visual material accompanied by written commentary, arranged chronologically.

In information visualization, several visualizations focus on correlating time events with structure or metrics (values). For example, SemTime uses a set of stacked timelines for the same or different time ranges, decorated with arrows to show relationships between data elements in the timelines [8]. Continuum combines the use of a timeline with a scalable histogram overview and allows the navigation through a complete hierarchical data set [1]. Treevolution uses a tree ring metaphor to represent hierarchical time-based structures to browse and discover relationships in the history of computer languages [14]. Spiral Graph uses a spiral metaphor to show a timeline to visualize of large time data sets, compare values along time, and detect periodic behaviors and trends [19]. The Spiral Semantic Timeline [6] uses a radial layout and accumulates activities in each time period. This visualization uses semantic zoom to allow pattern discovery by years, months, weeks, days and hours.

The use of timelines in the visualization of software evolution is often linked to the representation of software project hierarchies shown with treemaps, graphs, radial layouts and cone trees. Two

related approaches to our work that have been applied to visualize software project hierarchies are [2] and [5]. At a higher level of granularity are the evolution visualizations of Pinzger *et al.* [12] and Voinea *et al.* [17], which scalably show the evolution of metrics over thousands of software artifacts and hundreds of changes.

Without detailing further due to space limitations, however, we see several unsolved, yet general, challenges in software evolution visualization. First, we may want to visualize different types of data at different time scales (years, months, hours days) and also correlate the different scales. Second, visualizing software structural changes is still hard, although much work was done in graph animation and graph evolution. Last but not least, it is crucial to *intuitively* combine different types of visualizations when addressing a real-world task, as outlined by visual analytics research [20]. While not claiming to fully address such goals, our work here presents several steps in designing such a solution for software evolution analysis purposes.

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 [20]. Central to this process is the creation of

Figure 2: Structure comparison view showing a small software package. Lines represent sub-packages and circles show classes

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 four linked views. In decreasing level-of-detail order, these are: a timeline view, a structure evolution view, a metrics view, and an indirect class coupling representation that integrates source code viewing. These views and their interactions are described next.

3.1 Timeline view

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 related layout was used by Holten *et al.* to show software project hierarchies [2]. This type of layout compactly presents large quantities of data and provides an overview + detail view. We used shading to emphasize structure (dark along the ring borders, bright in the middle). This is slightly similar to [2], with the difference that they shade all borders of each cell separately to emphasize fine-grained structure, whereas we use shading to emphasize the multilevel time scales.

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 [4]. 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 such as [2], 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 (see Sec. 3.3 for concrete examples). 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 [13].

The ‘months’ ring embeds a different chart: a height plot showing the number of committed revisions per day-of-month. Next, the

‘days’ ring shows revisions within each day, so this ring has 30 or 31 cells. In each cell, a matrix dot plot is drawn in polar (ρ, ϕ) coordinates, where ρ 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 ‘hours’ ring shows revisions by hour, so it has 24 cells. Within a cell, the radial position of a revision shows the creation minute. The shapes used for revisions in the ‘hours’ ring indicate the change of a user-selected metric among such revisions. For example, the triangle shape shown for November 6th, 2008 indicates an increase of a user-selected metric of interest for that revision. Shape colors show, again, developer identity as for the ‘days’ ring.

Figure 3: Metrics visualization, project-wide (left) and per-class (right), over all revisions

3.2 Structure evolution view

The second visualization targets the task of analyzing the evolution of the structure of a given subsystem, which is selected by clicking a dot (a revision) in the circular timeline overview (Section 3.1). For this task, we propose a variation of the binary H-V tree [15] for the representation of n-ary trees, as shown in Fig. 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

Figure 4: Indirect-coupling view

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 top level nodes). This type of interaction supports common structural evolution analysis 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 view

The third visualization targets the display of the evolution of metrics. The user can select in the timeline view either a period of time and view project-wide aggregated metrics over that period (Fig. 3 left) or a specific class and view class-level metrics over the entire class evolution (Fig. 3 right). The metric views use the well-known overlapped graphs metaphor: the x axis shows time; the y axis shows metric values; each metric is drawn as a line graph with a different color; finally, revisions and their author IDs are indicated as vertical lines marked by colored dots, just like in the 'days' and 'hours' timeline rings (Sec. 3.1). Project-wide metrics are useful to monitor overall system quality trends [18]. Class-level metrics help more fine-grained development tasks. Besides metric values, the class-level view (Fig. 3 right) also indicates the appearance (1), deletion (2), or re-addition (3) of a class to a revision - class deletions are indicated by graph portions falling below the zero metric level.

3.4 Visualization of the indirect class coupling integrating source code viewing

The fourth visualization design addresses the representation of indirect coupling integrated with source code viewing. The indirect coupling consists of the indirect associations of a given class with other classes. Let's say that class A creates an instance of B and then B creates an instance of C. Then, A has a direct coupling relationship with B and an indirect coupling relationship with C. Our visualization proposal for indirect coupling allows seeing the classes and methods that affect the tasks carried out by a particular class. Thus, when a class has been selected, a graph is built including only the source code fragments to which that class has direct and indirect references. Figure 4 shows the visualization when the class `LineartimelineFrame` has been selected: that class

is coupled to the class `LinearTimeline`, which at the same time is coupled to the class `Tree` and finally, the class `Tree` is coupled to the classes `Node` and `Timeline`. This visualization allows a fine-grained examination of the indirect class coupling. The large central yellow frame shows the code of a method that has been selected for zoom-in `Tree::addActivityTree` which calls two other methods, `Node::addChild` and `Timeline::addElement`. The respective calls are highlighted in color, and the method bodies are shown in the two frames at the right. The two left frames indicate the indirect class coupling up to the selected class containing, as leaf, the method `Tree::addActivityTree` that is called by the `LinearTimeline` class. Hence, the x axis in this view indicates coupling dependencies: from a selected class (to the left) to a called method (center) and to its called methods (right). Although not yet done, this view could be easily combined with color mapping techniques to show the commit events, or authors, related to each line of code, like in [16]

3.5 Discussion

Figure 5 shows a snapshot of our tool with the 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 [10], computed directly from the (Java) source code in the examined repository. The metric view in the lower panel shows the selected metrics' evolution (Sec. 3.3). 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, or zoom in to the code level with the indirect-coupling view. 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).

This design offers several interaction modes. For example, when the user selects a revision, other revisions created by the same developer are highlighted in the timeline and the metrics views. Standard navigations *e.g.* filtering metrics by value ranges, open or close packages, and showing an inheritance class path, are also included.

Although we have not conducted a formal validation of the effectiveness of our linked-view evolution visualization, we have designed and used different types of evolution visualizations, targeting very similar tasks and users for a period of over four years, several tens of users, and around 15 academic and commercial analyses [17, 18]. It is interesting to compare the two approaches. The metrics visualization in [18] 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 [18] 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 offer a multilevel aggregation of information as we do here. The structure evolution view shown here is new and supports finer-grained tasks that the design in [18] does not address yet. Finally, the hierarchy-and-code view offers possibilities for correlating three types of data: class inheritance, dependencies, and code, in a compact way.

Figure 5: Visualization design integrating the timeline, structure evolution, and metrics views

4. CONCLUSIONS

In this paper, we proposed a four-view design of an exploratory visualization for combined metrics-and-structure data for software evolution. The four views focus on different tasks and use-cases: overview of the project commits structure and related metrics (time-line view), comparison of evolving package or class hierarchy structures over time (structure evolution view), trend analysis of metrics (metric view), and detailed code inspection (visualization of the indirect class coupling integrating source code viewing). The views are linked by interaction, selection, and navigation functions. Although a formal evaluation lacks, we believe from our previous experience with designing and using evolution visualizations that this type of combination provides added value in cross-role tasks, such as managers that need to overview, discuss and report activity and project quality correlations, or developers that need to learn new code bases, development history, and track structural changes.

Future work includes validating the proposal by analyzing several software projects by comparison with existing evolution visualizations, as well as the inclusion of more visualization designs for providing more details at the source code level.

5. REFERENCES

- [1] P. André, M. L. Wilson, A. Russell, D. A. Smith, A. Owens, and M. C. Schraefel. Continuum: designing timelines for hierarchies, relationships and scale. In *Proc. 20th Symp. on User Interface Software and Technology*, pages 101–110. ACM, 2007.
- [2] B. Cornelissen, D. Holten, A. Zaidman, L. Moonen, J. van Wijk, and A. van Deursen. Understanding execution traces using massive sequence and circular bundle views. In *Proc. ICPC*, pages 49–58. IEEE, 2007.
- [3] J. Fernandez-Ramil, A. Lozano, M. Wermelinger, and A. Capiluppi. *Empirical Studies of Open Source Evolution*. Springer, 2008.
- [4] FreeCol-team. Freecol. <http://www.freecol.org>, May 2009.
- [5] J. Garcia, A. Gonzalez, D. A. Gomez, R. Theron, and F. J. Garcia. A visual analytics tool for software project structure and relationships among classes. *Proc. Smart Graphics*, 2009.
- [6] D. A. Gomez, R. Theron, and F. J. Garcia. Semantic spiral timelines used as support for e-learning. *J. Universal Computer Science*, 2009.
- [7] H. Hochheiser and B. Shneiderman. Dynamic query tools for time series data sets: timebox widgets for interactive exploration. *Information Visualization*, 3(1):1–18, 2004.
- [8] M. Jensen. Visualizing complex semantic timelines. Tech. Report NBTR2003-001, NewsBlip, 2003.
- [9] U. D. Lago, A. Montanari, and G. Puppis. On the equivalence of automaton-based representations of time granularities. In *Proc. 14th Intl. Symp. on Temporal Representation and Reasoning*, pages 82–93. IEEE, 2007.
- [10] M. Lanza and R. Marinescu. *Object-oriented metrics in practice*. Springer, 2006.
- [11] M. Lanza and M. Pinzger. *Analysing Software Repositories to Understand Software Evolution*. Springer, 2008.
- [12] M. Pinzger, H. Gall, M. Fischer, and M. Lanza. Visualizing multiple evolution metrics. In *Proc. SoftVis’05*, pages 67–75. ACM, 2005.
- [13] Smartmoney. Map of the market. <http://www.smartmoney.com>, May 2009.
- [14] R. Theron. Hierarchical-temporal data visualization using a ring tree metaphor. *Proc. Smart Graphics*, 2006.
- [15] I. Tollis, G. DiBattista, P. Eades, and R. Tamassia. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice Hall, 1998.
- [16] L. Voinea and A. Telea. CVSgrab: Mining the history of large software projects. In *Proc. EuroVis’06*, pages 123–131, 2006.
- [17] L. Voinea and A. Telea. Multiscale and multivariate visualizations of software evolution. In *Proc. SoftVis’06*, pages 115–124. ACM, 2006.
- [18] L. Voinea and A. Telea. Visual querying and analysis of large software repositories. *Empirical Software Engineering*, 2008.
- [19] M. Weber, M. Alexa, and W. Muller. Visualizing time-series on spirals. *IEEE TVCG*, pages 7–13, 2001.
- [20] P. C. Wong and J. Thomas. Visual analytics. *Comp. Graphics & Applications*, 24(5):20–28, 2004.