Evolution Analysis with Animated and 3D-Visualizations

Sven Wenzel, Jens Koch, Udo Kelter
Software Engineering Group
University of Siegen
{wenzel,–,kelter}@informatik.uni-siegen.de

Andreas Kolb
Computer Graphics & Multimedia Systems
University of Siegen
kolb@informatik.uni-siegen.de

Abstract

Large software systems typically exist in many revisions. In order to analyze such systems, their evolution needs to be analyzed, too. The main challenge in this context is to cope with the large volume of data and to visualize the system and its evolution as a whole. This paper presents an approach for visualizing the evolution of large-scale systems which uses three different, tightly integrated visualizations that are 3-dimensional and/or animated and which support different analysis tasks. According to a first empirical study, all tasks are supported well by at least one visualization.

1. Introduction

Many different questions arise when analyzing the evolution of a software system. Some questions can be answered exactly by computing appropriate metrics. Other analysis goals are defined more vaguely. For instance, we want to analyze the architectural quality of a software systems and look for components which are unstable in the sense that they are changed over an over again. But what kind of changes are we interested in? Which metrics should be inspected? In these cases an exploratory approach is needed. Especially, if the system is very large and very complex, we first need an overview of the system which shows us the big trends and anomalies which need to be considered in greater detail.

Large tables of metric values will hardly provide us with an overview. Visualization allows us to capture trends and irregularities very fast, since our cognitive skills allow us to quickly identify graphical patterns. Hence, we propose to solve evolution analysis problems with visualization.

However, a single visualization cannot cover all aspects of software evolution; the user would be overwhelmed with information. It is better to have different visualizations for different aspects and analysis goals. Each visualization can focus on a particular subject and exploit the human visual perception. Since different visualizations can confuse the user if they are not consistent and well integrated, we aim at different visualizations which can interact. E.g. if an entity is selected in one visualization it should also be selected in the other visualizations.

2. Related Work

Graphs have traditionally been used for direct visualization of dependencies between software entities, other graph types are used to visualize refinement hierarchies [6, 8]. In general, purely graph-based systems do not scale properly to large systems if information about the *complete* population of system entities is sought. Additionally, graph-based techniques are not able to integrate substantial information about the evolution of a system.

Polymetric views can visualize several software metrics in parallel using standard visual attributes like position, width, height, and color. Coarse-grained views aim at visualizing a complete system. Fine-grained views show internal class structures. Evolutionary views display the changes over time, i.e. the evolution matrix [4]. Because of the two dimensions, only a very small number of entities and versions can be visualized. The concept of polymetric views can also be used to visualize the amount of change between revisions of models [13].

Code-line-based visualization considers software as a sequence of statements and maps the evolution of a code fragment onto pixel lines forming a 2D diagram, e.g. the *SeeSoft* metaphor [2]. In [15] the line-based technique is applied to system entities instead of code lines and *evolution spectrographs* are generated, which give a more global impression of the evolution of a system. In [5] the metaphor is extended to 3D to enlarge the number of visualized information.

Visualizations based on real-world metaphors, e.g. [1], map software attributes to parameters of virtual objects in cities or landscapes. All these metaphors cannot be extended to incorporate evolutionary aspects in a natural manner. In [14] the city metaphor is extended by *timelines*, similar to our evolution view. Colors are used to indicate the aging of entities; other evolutionary aspects are not covered.

978-1-4244-4828-9/09/$25.00 2009 IEEE Proc. ICSM 2009, Edmonton, Canada

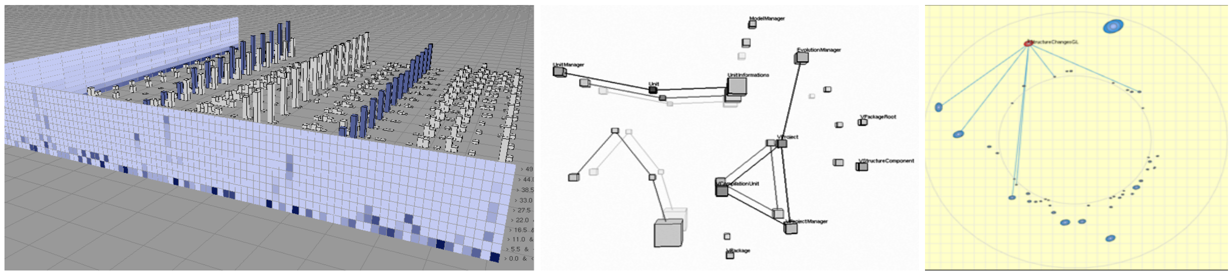

Figure 1. Different views: evolution view & spectrograph, structural changes, and animation view

3. The Visualization Concepts

We developed three integrated visualizations which provide different views on the analyzed system: the evolution view, the animation view, and the structural changes view.

3.1. Evolution View

The evolution view shows all selected entities of the evolving software system as a three-dimensional bar chart. The concept is based on the idea of Lanza's *Evolution Matrix* [4], but extended into the third dimension. Each version of an entity is represented by a column. The height of each column represents the value of a metrics. Metrics can be software metrics, difference metrics [11], or any other metric. The columns are ordered along the x-axis according to a sort criterion (e.g. the class name). The different versions of the entities are arranged consecutively in the direction of the z-axis. Corresponding entities, i.e. the same entity in different versions, have the same position on the x-axis. The evolution view may contain gaps, if entities are inserted in later versions, deleted, or moved. The visualization allows us to analyze different entities in detail. However, it is often not clear in the first place, which entities should be analyzed, especially when maintaining unknown systems.

A *relief extension* computes a surface on top of the columns of the different entities and versions respectively. It is colored according to the changes applied to the subsequent version. If the metric value increases from one version to the next the surface is colored green (black). Decrease is colored red (gray). Unchanged metric values lead to gray (lined). The colored areas can quickly be identified as entities and/or versions where many changes occurred.

While the relief extension enables us to quickly identify changed entities or system versions, the *spectrograph extension* visualizes the distribution of metric values on two walls along the x-axis and the z-axis. Our spectrographs resemble spectrographs in sound analysis, which display the frequency content over time. Each spectrograph is a wall consisting of a sequence of columns. A column represents an entity on the x-axis wall and a system version on the z-axis wall. All columns are divided into cells. Each cell is associated with a range of a metric value, it is colored with different tones of blue to indicate the number of entities being in that particular range of metric values. The darker a cell is, the more entities have a metric value within that range. A column of the x-axis wall shows the distribution of metric values of an entity over time. Similarly, a column of the z-axis wall shows the distribution of metric values in the different entities within one version of the system.

The spectrograph can be used as a "query facility": if cells on one of the walls are selected, only entities having metric values within the corresponding ranges are shown.

3.2. Animation View

The animation view uses animation to visualize changes of metric values from on version to the next. The animation can either be watched in form of a movie, or the user can manually browse through the versions.

An entity in one specific version is drawn as an ellipse in the two-dimensional space. The ellipses encode two different metrics by their size and their direction, and a third metric by the distance to the center of the circle. Thus, up to three metrics can be visualized.

If an entity changes its metrics from one version to the next, the corresponding glyph changes its size, direction, or location. Larger changes translate into faster movement.

Additionally, the user can select entities whose relationships with other entities are to be drawn (e.g. associations between classes). Simple lines connect the different glyphs. The restriction to show only relationships of selected entities is necessary because the view would become too confusing otherwise. Changes of relationships are also shown in the animation.

3.3. Structural Changes View

A central aspect of evolution visualization of large software systems are structural changes, i.e. changes which modify the structural relationships between different entities, e.g. associations between classes. In order to visualize these changes we show abstract, diagram-like representations of different versions of the software system. Each version is represented by one diagram. A diagram is a set of cubes (one for each entity) and lines that express relationships between entities. The color and the size of a cube encode different metrics. We do not use standard graphical notations as, e.g., defined in the UML, because they often contain too many details which would obscure the evolution visualization. However, layout properties (e.g. positions from a class diagram) can be used to arrange the cubes. Beside import of layout data, we provide different standard layouters such as string layout or circle layout, and the user can arrange the cubes manually. Different diagrams are generated for the different versions of the system and arranged behind each other along the z-axis in the three-dimensional space. The x- and y-positions of the cubes is the same for corresponding entities, i.e. all versions of an entity lay on one line, and looking from the front we only see the newest version.

In order to avoid visual overload, a blend factor can be defined to make subsequent diagrams more transparent, so that only the desired number of versions can be seen and more distant versions become invisible. A navigation controller enables users to browse through the history of versions and to select the version which displayed in the front. The user can freely navigate within this 3D-visualization.

4. The Analysis Tool

Each visualization comes with advantages and drawbacks. Hence, we designed an analysis tool in which we can easily switch between different visualization concepts. The tool has been realized as Eclipse plug-in and the visualizations have been realized with the OpenGL Bindings for Java (JOGL). All visualizations are embedded in a main window (perspective) as shown in Fig. 2. The screen is divided into four regions. The (global) **model tree view** on the left hand side shows a tree representation of all entities that exist at any time within the evolution of the analyzed software system. Each entry has a checkbox which, when ticked, selects these entities for visualization. The **information view** on the right hand side provides information about the currently selected entity: its name and identifier, context information if existent, and analysis information such as metric values. The **settings view** enables us to configure the current visualization, e.g. to select metrics to be displayed, scale factors, or layouts. The **main view** contains the visualizations of the software evolution either in different tabs, or side by side, when looking for relations between different visualized aspects.

All views and especially the different visualizations are tightly integrated. Whenever entities are selected during analysis, the selection is applied to all views and visualizations.

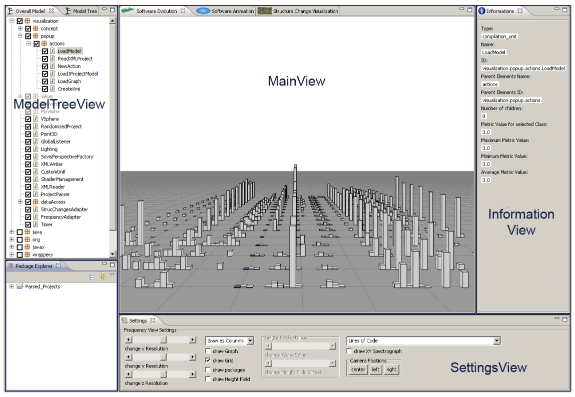

Figure 2. The analysis tool.

5. Evaluation

Based on the tool implementation, we evaluated our visualization concept in a first empirical case study. The study involved 15 test persons, which have mainly been undergraduate students. Each test person had to analyze the evolution of two software systems; one project which was unknown to the test persons and one project which had been developed by themselves. The projects ranged from 30 to 100 classes with up to 20000 lines of code. On average, 19 versions have been analyzed in the different projects. The testers had to analyze the projects regarding different aspects, such as stable code, error-prone code, or dead-code. With a questionnaire we asked for a rating from 1 (bad) to 5 (excellent) points for each visualization concept. Afterwards, we asked for an assessment of the quality of each visualization concept (e.g. scalability, navigation, etc.), and questions regarding the tool (e.g. model tree view, displayed information, etc.).

The study provided very good results; on average, each visualization concept had a rating of more than 3 points for the different analysis problems, i.e. the test persons gave mainly positive answers.

For the unknown project, the evolution view offered the best solution for the location of stable code. Here, the average rating almost reached the maximum. The animation view got a good rating, too, with more than 4 points in average. The structural changes view was rated with 2.6 points, which was the worst rating of all. The search for error-prone code and entities with a short life time was supported by all our visualization concepts similarly. The location of dead code was best supported by the structural changes view.

For the known project, the study delivered similar promising results. Here, the testers had to solve tasks such as locating the entities with most changes (task A), finding the versions with most changes (task B), and finding the versions with most structural changes (task C). On average, the evolution view (3.97 points) and the animation view (3.94) got the best ratings, however, all visualization concepts provided good results. Tasks A and B were supported equally well by the evolution view (4.09 points). The spectrograph extension provided a good support here. However, for task A, the animation view still performed better (4.46 points) because the high number of changes in the test data was well reflected in the continuous change during the animation. The overview about the changes of versions (task B), however, is not obvious in this view. The structural changes view did not sufficiently support task A; it got a rating of 2.64 points only. Surprisingly, the version with the most structural changes was not located at best with the structural changes view either. It was only rated as good as the evolution view. This weakness of the structural changes view is due to the problem that we cannot display all versions at a time; if we did so, the visualization became unclear.

The test persons were also asked about general properties of the different visualizations. The testers have been asked again to give up to 5 points in different categories. None of the concepts got a rating lower than 3 in any category. The evolution view got the best results for scalability, support of complex systems, and support of evolution analysis. The animation view got the best results for navigation and interaction. The structural changes view got the worst rating in all categories.

The users also assessed the extensions of the evolution view. 33% of the test persons saw a benefit in the spectrograph, the relief extension was rated positively by 40% of the testers. In general, the evolution view was rated best due to its intuitive behavior.

We were initially afraid that the encoding of three different metrics in the animation view might overload the user. Thus we asked the test persons about their impression: only three persons felt to have been overloaded. Only two test persons had a problem to recognize changes in metric values within the animation view. Some test persons requested an option to switch off the interpolation between metric values of different system versions.

The structural changes view was often assessed to be very confusing. The layout generated by the predefined layouters, which have their origin in graph visualization, was not preferred; it got a rating of only 2.8. The manual arrangement of cubes, however, was often used and got a better rating of 3.5.

The overall usage of the tool was rated with 3 points. The feature that the selection of entities is synchronized between the model tree view and all visualizations was in average rated with 4 points. Many test persons, however, missed a function to directly switch from the visualization into the source code.

6. Outlook

In this paper we presented an approach for visual evolution analysis, which integrates three novel visualization concepts. The approach has been implemented in an Eclipse-based tool, which was evaluated to be very convenient in a first empirical case study. Further material such as videos showing the tool in action can be found at [16].

Currently, we run another case study with different open-source projects to evaluate the applicability to larger projects. We further work on the integration with more advanced features such as difference analysis [10] and automated trace recovery [12].

References

[1] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz. Software landscapes: Visualizing the structure of large software systems. In *Proc. Symp. on Visualization (VisSym)*, 2004.

[2] S. Eick, J. Steffen, and E. Sumner. Seesoft - a tool for visualizing line oriented software statistics. *IEEE Trans. Softw. Eng.*, 18(11):957–968, 1992.

[3] C. Knight and M. Munro. Comprehension with[in] virtual environment visualisations. In *Proc. IWPC*, pp. 4–11, 1999.

[4] M. Lanza. The evolution matrix: Recovering software evolution using software visualization techniques. In *Proc. IWPSE*, 2001.

[5] A. Marcus, L. Feng, and J. Maletic. 3D representations for software visualization. In *Proc. ACM Symp. on Software Visualization (SoftVis)*, pp. 27–ff, 2003.

[6] H. Muller and K. Klashinsky. Rigi: a system for programming-in-the-large. *Proc. ICSE*, pp. 80–86, 1988.

[7] T. Panas, R. Berrigan, and J. Grundy. A 3d metaphor for software production visualization. In *Proc. Intl. Conf. on Information Visualization (IV)*, pp. 314–320, 2003.

[8] M. Storey and H. A. Muller. Manipulating and documenting software structures using SHriMP views. In *Proc. ICSM*, p. 275, 1995.

[9] A. Telea, A. Maccari, and C. Riva. An open toolkit for prototyping reverse engineering visualizations. In *Proc. Symp. on Data Visualisation (VisSym)*, pp. 241–248, 2002.

[10] C. Treude, S. Berlik, S. Wenzel, and U. Kelter. Difference computation of large models. In *Proc. ESEC/FSE*, pp. 295–304, 2007.

[11] S. Wenzel. Scalable visualization of model differences. In *Proc. of the International Workshop on Comparison and Versioning of Software Models (CVSM'08)*, May 2008.

[12] S. Wenzel, H. Hutter, and U. Kelter. Tracing model elements. In *Proc. ICSM*, pp. 104–113 , 2007.

[13] S. Wenzel and U. Kelter. Analyzing model evolution. In *Proc. ICSE*, pp. 831–834, 2008.

[14] R. Wettel and M. Lanza. Visual exploration of large-scale system evolution. In *Proc. WCRE*, pp. 219–228, 2008.

[15] J. Wu, R. Holt, and A. Hassan. Exploring software evolution using spectrographs. In *Proc. WCRE*, pp. 80–89, 2004.

[16] Project homepage. `http://pi.informatik.uni-siegen.de/projects/evolver`, 2009.