eCITY: Evolutionary Software Architecture Visualization – An Evaluation

Taimur Khan[1(✉)], Henning Barthel[2], Liliana Guzman[2], Achim Ebert[1], and Peter Liggesmeyer[1]

[1] Computer Graphics and HCI Group, University of Kaiserslautern, Gottlieb-Daimler-Str., 67663 Kaiserslautern, Germany
{tkhan,ebert,liggesmeyer}@cs.uni-kl.de
[2] Fraunhofer IESE, Fraunhofer-Platz 1, 67663 Kaiserslautern, Germany
{Henning.Barthel,Liliana.Guzman}@iese.fraunhofer.de

Abstract. An essential component in the evolution and maintenance of large-scale software systems is to track the structure of a software system to explain how a system has evolved to its present state and to predict its future development. Current mainstream tools facilitating the structural evolution of software architecture by visualization are confined with easy to integrate visualization techniques such as node-link diagrams, while more applicable solutions have been proposed in academic research. To bridge this gap, we have incorporated additional views to a conventional tool that integrates an interactive evolving city layout and a combination of charts. However, due to a limited access to the stakeholders it was not possible to solicit them for a formal modeling process. Instead, an early prototype was developed and a controlled experiment was conducted to illustrate the vital role of such in-situ visualization techniques when aiming to understanding the evolution of software architecture.

Keywords: Software architecture visualization · Software comprehension · Software evolution · Experiment

1 Introduction

Mainstream software systems undergo continuous changes in order to adapt to new technologies, to meet new requirements, and to repair errors [1]. Inevitably, the software in question expands in both size and complexity, often leading to a situation where the original design gradually decays unless proper maintenance is performed [2]. However, due to the "complex, abstract, and difficult to observe" nature of software systems performing visually supported maintenance can be quite complicated [3]. The field of software visualization aims to ease this task by providing visual representations and techniques that make the software more comprehensible. A key ingredient of these visualizations is a visual representation of the software structure that assists in creating a mental map of the system. Such a mental map provides a means to examine product properties such as size and quality indicators and process events such as errors found or changes made [4].

A. Ebert et al. (Eds.): HCIV Workshops 2011, LNCS 8345, pp. 201–224, 2014.
DOI: 10.1007/978-3-642-54894-9_15,

With respect to the analysis of the evolution of software, it is essential to track the structure of the software system to explain and document how a system has evolved to its present state and to predict its future development [5]. There are a number of free and commercial tools that can be found in both academic and industrial research with the sole purpose of improving software architecture evolution comprehension through the use of visualization. On the one hand, industrial applications are confined to easy to integrate visualization methods and metaphors that lack the sophistication to handle informative large-scale software architecture evolution visualization. While on the other, academic researchers have developed numerous solutions that have not made it to the mainstream [6–8]. The work of Telea et al. [7] indicates two inter-related reasons for this phenomenon and we agree with their findings; (1) stakeholders do not have the time to try new tools to see if it fits in their context, and (2) tool developers cannot create an product that satisfies all possible needs. Our aim is to bridge this gap by addressing the following two factors: (1) propose visualization methods and metaphors that do not significantly deviate from current solutions - analysts should be comfortable with the software architecture representations visually, and (2) the ability to monitor, visualize, and interact with large-scale software systems in real-time - be able to deal with the scale and complexity of real-world software applications.

The goal of this project is to utilize research ideas in the area of software architecture evolution visualization and to apply these modern techniques in the context of mainstream software architecture maintenance and evolution tools. To achieve this goal we have been working together with the Fraunhofer IESE, to enhance their conventional SAVE (Software Architecture Visualization and Evaluation) tool that evaluates software architectures while they are constructed as well as after their construction [9]. However, due to a limited access to the stakeholders it was not possible to solicit them for a formal modeling process. Instead, a prototype was developed to augment SAVE through the use of different views to further the daunting task of large-scale software architecture evolution analysis. While the experimental results show that some details were missed through this non-formal approach, they also show that an improved configuration of the visualization influences the efficiency and effectiveness of basic software architecture evolution tasks significantly. More specifically, a gain of efficiency by 170 % and a gain of effectiveness by 15 % in these basic tasks were realized simply by selecting a different set of views. Based on these results we claim that considerable benefits can be attained by incorporating such in-situ visualization methods and metaphors to a conventional software architecture maintenance and evolution tool.

The organization of this paper is as follows: related work is examined in Sect. 2, the eCITY tool is described in Sect. 3, and the experiment setup and results are reported in Sect. 4. Finally, we conclude this paper and look at possible future work in Sect. 5.

2 Related Work

As software maintenance is mainly performed at code level, majority of the visualizations employ a 2D line-based approach to represent software evolution [10–12]. In such scenarios, the adopted procedure is to visually map a code line to a pixel line and to utilize color to depict the age of a code fragment [10]. Additional focus has been to enhance interaction techniques to improve navigation and exploration of the underlying data [11,12]. While these techniques thrive in tracking the line-based structure of software systems and reveal change dependencies at given moments in time, they lack the sophistication to offer insight into attribute and structural changes made throughout the development process.

In contrast, there are only a small number of visualizations that represent structural changes of a system architecture over time [8]. Holten and van Wijk present a technique that compares the software hierarchies of two software versions [13]. The algorithm positions matching nodes opposite to each other to better compare the two versions. They utilize shading to highlight nodes that are present in one version but not the other. Further, Holten employs his well-known edge bundling technique to highlight and track the selected hierarchy.

Collberg et al. illustrate the use of a graph drawing technique, that has a temporal component for the visualization of large graphs, to visualize the evolution of a software system [14]. They employ force-directed layouts to plot call graphs, control-flow graphs, and inheritance graphs of Java programs. Changes that the graphs have gone through since inception are highlighted through the use of color. Nodes and edges are initially given the color assigned to its author (red, yellow, or green) and progressively age to blue.

A recent research focus is to utilize and extend intuitive metaphors to aid in the visualization of software systems. Our work is in fact inspired by the original contribution of Steinbrückner et al., where they propose the idea of stable city layouts for evolving software systems [4]. They describe a three-staged visualization approach, where the first stage constitutes of a primary model responsible for capturing the software system structure and its evolution details. The second stage refers to a secondary model that initially adds geometric information to the primary model. This model is then further enhanced through the use of elevation levels to directly depict a software systems development history in the layout. Finally the third stage is comprised of tertiary models that are derived from secondary models by applying projections, coloring, or the imposition of symbols or diagrams. While we employ the above-mentioned basic secondary model, we utilize a combination of animated transitions to grow or shrink the city instead of using different elevation levels and color interpolations to highlight the evolution of components.

3 eCITY: An Evolving City

The eCITY tool follows the methodology of Eick et al. [15] to employ the well-known idea of multiple architectural views of large and complex software

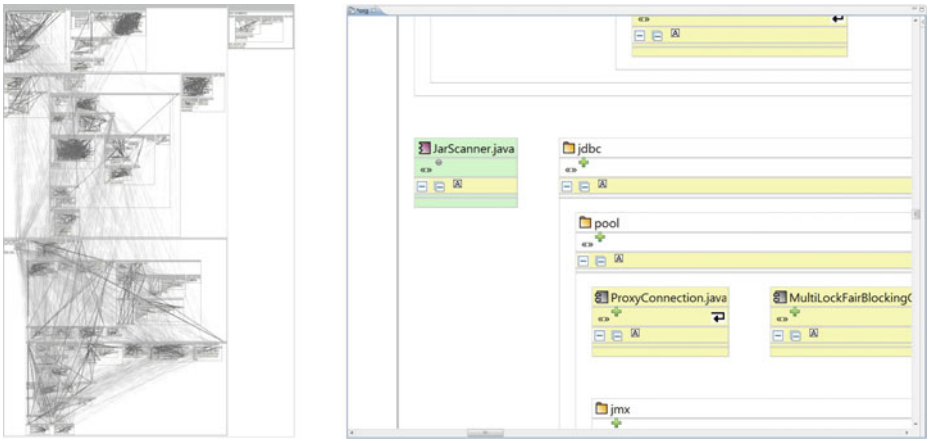
systems with a focus on evolution tasks. This approach was adopted to enhance the architects' current workflow that relies solely on the SAVE Diagram View (Sect. 3.1) - a view that provides extensive evaluation possibilities but one that does not have the best design to handle the complexities of exploring the systems evolution. To address this shortcoming, we have implemented a combination of views; a Timeline View (see Sect. 3.2) and an evolving city layout (see Sect. 3.3).

eCITY exploits the core functionality provided by the Fraunhofer SAVE tool. One of these features is responsible for extracting the underlying architectural model from the software systems source code. This model is generated through a combination of an initial fact extraction and a number of delta fact extractions, to produce a data model that contains architectural data about a software system over a period of time. In this section, we describe the three main views of the eCITY tool that utilize this data model to perform evolutionary software architecture tasks; namely the *SAVE Diagram View*, the *Timeline View*, and the *City View*. Here, it is important to note that although the SAVE Diagram View is part of the original SAVE tool, it has been incorporated in the eCITY tool so that the users have access to their original workflows. Both the Timeline and City Views have been designed to help the architects in accessing the structural changes of the software system in a more effective and efficient manner than using the SAVE Diagram View alone. We support this claim through the results of our experiment that are presented in Sect. 4.

3.1 SAVE Diagram View

The SAVE Diagram View (Fig. 1a) is the main view used by the software architects at the Fraunhofer IESE to explore and assess a software systems architecture. The main features of this view are: its configurability (enabling and/or disabling certain graphical elements), the expressiveness of its graphical elements, and its rich-features that allow for extensive evaluations. Projects, packages, and classes are all represented as components in the SAVE Diagram View, while edges between components depict their relationships to one another. In general, SAVE has a nested approach in which high-level components have rectangular representations that may be either expanded one-level deeper into the hierarchy or expanded completely to show the entire underlying hierarchy. Conversely, components may be collapsed in a similar manner. The work of Knodel et al. [9] discusses these elements and their features in further detail.

The complexity of a structural diagram is typically reduced by either creating new diagrams of selected components or through the use of various filters. For the purpose of analyzing the underlying software structure over time, the analysts rely on two distinct filters: the *Relation Type* filter and the *Point-In-Time* diagram filter. The user may reduce visual clutter, while solely examining the software structure, by applying a *Relation Type* filter to hide some or all of the displayed dependencies. Additionally, the user may apply a *Point-In-Time* diagram filter to explore the system at a particular point in time. As shown in Fig. 1b applying a *Point-In-Time* diagram filter updates the SAVE Diagram View with icons to depict the modification status at a certain point in time



(a) Complete Diagram (b) Applying filters and scrolling

Fig. 1. SAVE diagram view

according to the chosen parameters: a triangle depicts modifications, addition symbol depicts insertions, minus symbol depicts removals, and a circle depicts no change.

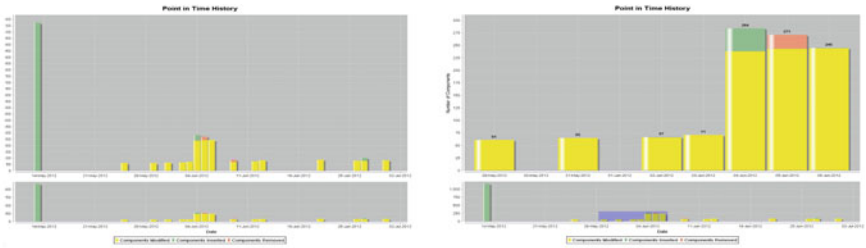
It was a conscious decision to incorporate the same view into eCITY, thereby not completely replacing the architects' normal workflow. Instead, we provide them with the views described in Sects. 3.2 and 3.3 to have additional perspectives of their evolution data.

3.2 Timeline View

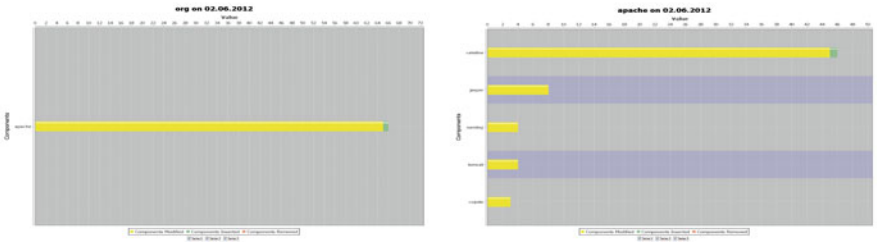
In the original SAVE workflow, architects apply a *Point-In-Time* filter to update the SAVE Diagram and manually track the number and type of changes made - a process which is deemed not only tedious but also error prone. The first step of this workflow requires the user to load the underlying data model that represents the software system being analyzed. It is during this process that data regarding the modification status of both the individual components as well as the overall hierarchy is stored; i.e. the distribution of these changes over time are stored in a convenient and easy to access manner.

As such, the main purpose of the Timeline View is to provide the user with an overview of changes made to the system over time. A combination of interactive bar charts that represent the number of modifications, insertions, and removals made to packages and classes are employed to achieve this task. A typical color scheme was employed to depict these changes; modifications, insertions, and removals are represented using yellow, green, and red colors respectively.

The initial view consists of a combined plot (Fig. 2a left); one of which is an overview where the user can select and manipulate a rectangular region, the other provides details on the selection (Fig. 2a right). Further, the user may



(a) Timeline overview and detail plots



(b) Top components and org.apache on 02.06.2012

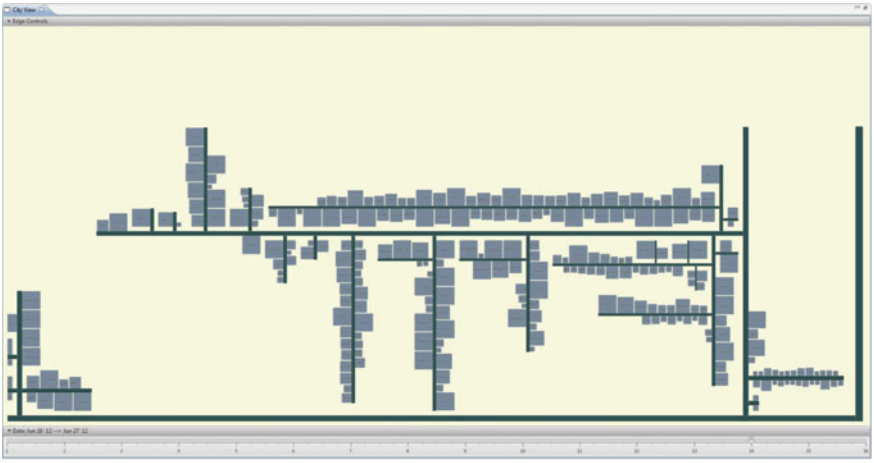
Fig. 2. Timeline view

interactively select a point in time and update all three views simultaneously. Such a selection, changes the plot and provides details of the top level component at the chosen time stamp (Fig. 2b left). Further, the user may recursively explore the distribution of changes over the hierarchy (Fig. 2b right). The user may also navigate back to the previous chart or directly to the overview chart of Fig. 2a. There are additional interaction possibilities with the charts; hovering over a chart component highlights the relevant subtree in the City View and selecting the chart component zooms onto the graphical representation of that particular component in the City View.

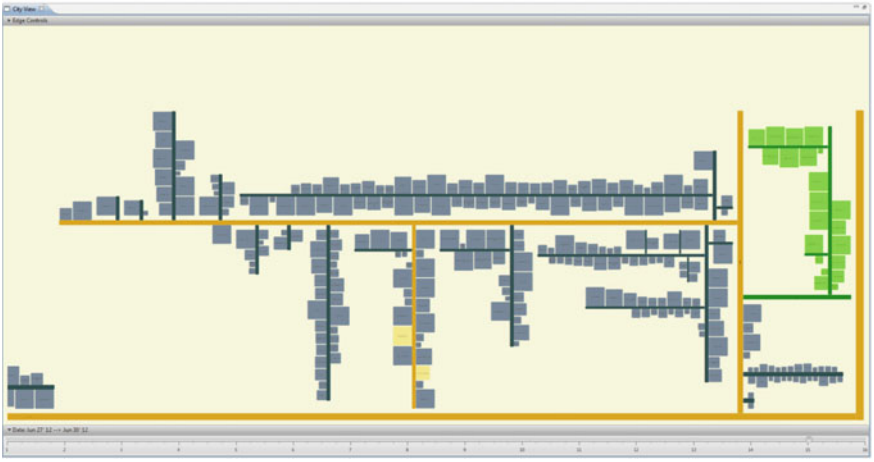
3.3 City View

An additional mode of analysis while monitoring the structural changes made to a software system is to track where these changes are made, i.e. where are packages and classes added, modified, or removed over time. In order to achieve this goal in the original SAVE workflow, architects apply multiple *Point-In-Time* filters and have to keep a mental map of these changes - this process is also deemed tedious, error prone, and depending on the size of the system hierarchy quite difficult. The City View (Fig. 3) addresses these concerns by providing an overview of the entire system architecture at a particular point in its evolution process and provides the user a means to interactively explore these changes over the system hierarchy.

It was mentioned in related works that our approach is inspired by the work of Steinbrückner et al. [4] and in particular the secondary model they present.



(a) Package org.apache.tomcat on 26.06.2012



(b) Package org.apache.tomcat on 27.06.2012

Fig. 3. Exploring org.apache.tomcat with eCITY (see Appendix C for details) (Color figure online)

While they focus on geometrically mapping different details of the development history onto this model, we choose to manipulate this layout to highlight structural changes over time. Figure 3a depicts our implementation of their secondary model, where packages are represented by streets and classes are symbolized by plots. The vertical or horizontal orientation of a street alters depending on its parent street/package orientation. Plots are positioned along the street representing the package they belong to; to reduce space requirements, they are positioned on both sides of the street. The layout algorithm adjusts the length

of the street to hold all its plots and all its subpackages. Further, the size of a plot may vary according to a given scaling parameter; because connectivity is an important measure for an architect, we scaled plots according to its connectivity to other artifacts.

While the user loads the SAVE data model, we calculate the initial layout as described above and sequentially go through the *Point-In-Times* to update the layout by adding and removing both classes and packages. Such details of the city layout as well as the modification status of its components are stored in key frames to allow for real-time interaction. Utilizing a slider and key frame animation techniques colors are interpolated and the city suburbs grow and shrink to represent changes made between two points in time.

The city may be explored using the mouse to pan and zoom to examine its suburbs or different parts more closely. Further, the user may interactively invoke a slider and manually update the city to another point in time. This interaction mechanism animates the city through the use of colors that depict the modification status and the growing and shrinking of suburbs to represent the addition and/or removal of classes and package. This functionality may be observed by comparing Fig. 3a with Fig. 3b, where certain classes are modified (color changes to yellow) and a package is inserted in the top right corner (colored green). The color scheme employed is quite similar to that of the Timeline View charts, where yellow, green, red, and grey are used to depict the modification, insertion, removal, and unchanged modification status of each component.

Besides panning and zooming capabilities, two distinct modes of operation have been incorporated into this view; one of these modes is triggered when the user selects a point in time in the combined plot of Fig. 2a, while the other mode requires the user to interact with a time slider. In the former mode the city may either animate or instantaneously update itself to the chosen point in time depending on whether the *Time Animation* is enabled or disabled. On the other hand, interacting with the time slider interactively updates the city to an older or a newer version depending on the direction chosen. As soon as the slider is released, the other two views are updated to reflect the active point in time. The user also has the possibility to reset the views or toggle certain animations using the popup menu. Finally, with mouse-over he is able to see the name of the underlying component, which comes in handy when he is zoomed out of the city.

4 Experiment

We proposed eCITY for improving the analysis of software architecture evolution over time. In particular, we assume that through the use of the Timeline and City views, software architects would be more efficient and effective in analyzing architectural changes. We also expect that eCITY will be better accepted and perceived as more useful than SAVE. For testing these assumptions, we designed and performed a controlled experiment.

4.1 Research Purpose and Hypotheses

The first goal was to compare the SAVE and eCITY configurations with respect to the efficiency and effectiveness achieved by analysts when they analyze architectural changes. Thus, we defined the following research hypotheses:

- H_1: Analysts using the eCITY configuration are more efficient than analysts using the SAVE configuration when they analyze the evolution of software architectures.
- H_2: Analysts using the eCITY configuration are more effective than analysts using the SAVE configuration when they analyze the evolution of software architectures.

The second goal was to compare SAVE and eCITY with respect to their acceptance and usability. Therefore, we define the two additional research hypotheses:

- H_3: Analysts accept the eCITY configuration more than the SAVE configuration when they analyze the evolution of software architectures.
- H_4: Analysts consider the eCITY configuration more useful than the SAVE configuration when they analyze the evolution of software architectures.

4.2 Operationalization

In order to test the above hypotheses, we operationalized the four variables of interest, selected a software system to be analyzed, and designated the tasks to be performed. First, the variables of interest were operationalized as follows:

i. Efficiency is the time required for accomplishing a set of given tasks.
ii. Effectiveness is the difference between the true and actual score related to a task.
iii. Acceptability is measured using the Technology Acceptance Model (TAM), which is a valid and reliable questionnaire for assessing technology acceptance and use [16]. Out of 31 questions in the original TAM, we selected 7 questions focused on performance and effort expectancy (Appendix A). All questions were rated using a five-point Likert scale (1: I strongly disagree, 5: I strongly agree).
iv. Usability is measured using the questionnaire proposed by Nestler et al. [17]. Considering the purposes of the eCITY configuration, we selected 4 out of 5 defined dimensions and 15 out of 269 questions. The selection was discussed with evaluation experts. The dimensions and related questions are listed in Appendix B. Each question is rated using a five-point Likert scale (1: I strongly disagree, 5: I strongly agree).

Second, we selected the Apache Tomcat system as the system to investigate because it is a real software system and its architectural models were available in SAVE. Finally, three types of architectural evolution tasks were identified with the support of experts. These are:

i. *Counting:* Identifying changes made to the system on a specific date or changes made to a subcomponent in a time period.
ii. *Find Date:* Identifying the time period with the most number of changes or when a component has been changed the most.
iii. *Find Package/s:* Identifying the subcomponents of a chosen component that have been changed in a time period, find modules/subcomponents that are present since the earliest version and have not been changed since, or find modules/subcomponents that have changed a lot in a particular time period.

4.3 Pilot Study

A pilot study was conducted with two experts in the field of software architecture. Our purpose was to get an early feedback regarding the experimental design, the selected tasks, and the time required for solving them.

The first expert completed the tasks using the SAVE configuration in an hour and fifteen minutes. He considered the tasks to be realistic. However, he suggested that while asking the user if certain components are changed often or not at all, we should focus on a smaller sub-package rather than the entire system. Hence reducing the time required to perform the tasks related to finding packages.

The second expert solved the tasks using the eCITY configuration in forty five minutes. He also considered the set of selected tasks to be realistic. Additionally, he made recommendations regarding the visualization. These included: (1) keeping the date format and color coding uniform, (2) adding the option to enable and disable animations, and (3) adding feedback regarding the enabled/disabled features in the City popup menu. These issues were discussed and solved.

Consequently, the original set of tasks were tweaked to address the experts' feedback.

4.4 Controlled Experiment

Subjects. The controlled experiment was conducted with 38 participants. Out of the 38 participants, 3 were software engineering experts of the Fraunhofer Institute for Experimental Software Engineering (IESE) and 35 were graduated students in computer science from the Technical University of Kaiserslautern. The three experts involved have a deep knowledge in the field of software architecture and prior experience in SAVE, while all the students have theoretical knowledge about software architectures.

Due to the lack of experts available and their previous experience in SAVE, experts were asked to work with SAVE and then with eCITY. Students were randomly distributed in two groups working either with SAVE or eCITY. This was intended for avoiding learning effects.

Experimental Setup. The same software architecture was explored using either SAVE or eCITY. Thus, two computers were prepared for the experiment:

one with Eclipse and the SAVE Configuration and the other with Eclipse and the eCITY configuration. The Eclipse workspace was prepared for both installations to contain the required data: a SAVE data model containing 16 fact extractions of the Apache Tomcat system between the time period 14.05.2012 and 02.07.2012. Each instance of the Apache Tomcat system was extracted from its public repository. Four of these instances were complemented with the insertion, modification, and removal of some fictitious packages and classes to create some interesting artifacts.

Each workspace was completely restored at the beginning of each experiment session. A different visualization introduction handout was provided to each participant depending on the installation used. Additionally, the participants were afforded with a walkthrough of the functionalities and features that would assist them in the experiment process. The participants were then asked to perform the tasks and questionnaires described in Sect. 4.2.

Each participant was allowed unlimited time to finish the tasks and to fill the questionnaires regarding Acceptability and Usability. The resulting materials were then collected and analyzed.

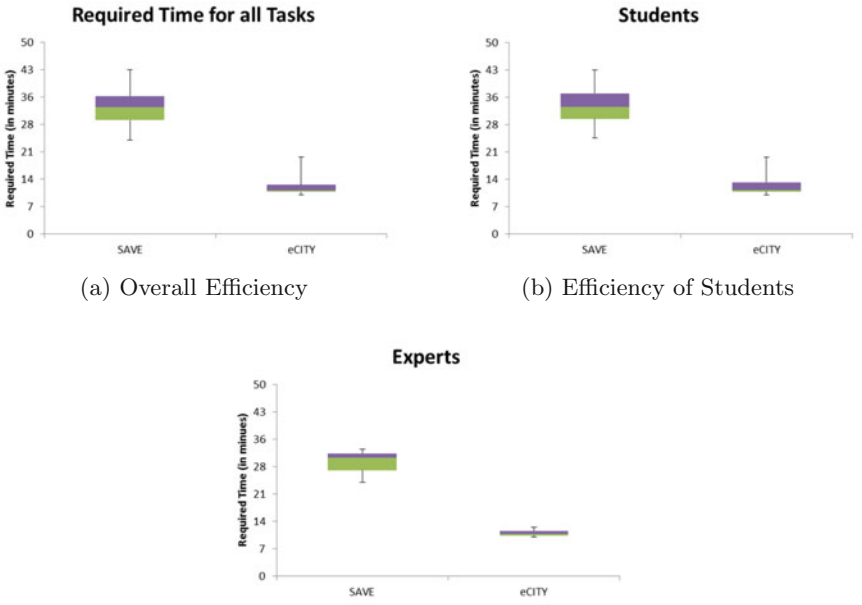
Data Collection. The time and the answer to each task was collected using an exercise sheet. Additionally, a printed questionnaire was used for eliciting the perception of the participants regarding the acceptance and the usability of the corresponding configurations, i.e. SAVE or eCITY.

Data Analysis. A transcript of the collected efficiency, effectiveness, acceptability, and usability data was compiled in excel. The subject data is kept anonymous and confidential. Regarding effectiveness, tasks were weighted according to their difficulty: Counting and Find Date tasks were awarded 3 points each and Find Package/s tasks were awarded 4 point each. The latter sets of tasks were weighted higher as they were deemed to be more complex.

We applied descriptive statistics methods such as the sample mean, standard deviation, median, and range to the experimental data. Further, we used the *MegaStat* excel plug-in to statistically test our hypotheses using the Mann-Whitney-U-Test. It is appropriate for our scenario due to the size of the data and its ability to handle both normal and non-normal distributions. We performed the tests with a confidence level for rejecting the null hypotheses at 99 %.

4.5 Results

The statistical evaluation of the experiment is presented in this section. As such, the time required for tasks, the results of the tasks, the acceptability of the configuration, and the usability of each configuration is presented. Further, we argue for the earlier stated hypotheses by performing hypotheses testing on the collected data. We conclude this section by examining the threats to validity



(a) Overall Efficiency

(b) Efficiency of Students

(c) Efficiency of Experts

Fig. 4. Efficiency box-plots (x-axis: left SAVE, right eCITY, y-axis: required time in minutes)

Efficiency Results. The time required for each participant to complete a task was measured to compare the efficiency with which the participants performed using each one of the two configurations. The results are presented in the box-plots of Fig. 4; Fig. 4a depicts the efficiency of the participants with respect to the SAVE and eCITY configurations, while Fig. 4b and c examine the efficiency of students and experts respectively. Based on these results, the following observations can be made:

- Independent from being experts or students, members of the eCITY group were on average more efficient.
- There is a larger variance from the average required time in the SAVE group than the eCITY group.
- Independent from the configuration used, students were on average as efficient as experts.
- The variance of the results is larger for students than experts

The mean, median, and standard deviation values of the efficiency with which the participants performed are presented in Table 1. The results presented have been broken down to the type of tasks, to inspect the level of efficiency reached for each type. It is not surprising to see such a significant improvement in the efficiency of counting tasks as the SAVE configuration required the participant

Table 1. Participants efficiency (time required in mm:ss)

		Mean	Median	Std. deviation
Counting	SAVE	11:20	11:15	2:11
	eCITY	1:59	1:58	0:32
Find Date	SAVE	8:09	7:29	2:46
	eCITY	2:12	2:07	0:42
Find Package/s	SAVE	13:50	13:19	2:11
	eCITY	8:10	7:50	1:39

to pan and zoom on an extremely large viewing area and physically count the changes while the eCITY configuration required them to look up charts and interact with sliders. The participants needed to interact with the eCITY configuration in much the same manner for the Find Date tasks, however, for the SAVE configuration they had to apply filters and compare the densities of each updated diagram. The biggest challenge they faced was having to keep a mental map of these changes. Similarly, the participants were more efficient in the Find Package/s tasks using the eCITY configuration as they found it easier to interactively update the city and locate both stable and heavily constructed areas.

In Table 2, we further examine the results from the SAVE (A) and eCITY (B) configurations. Using this table we can compare the efficiency gain of the eCITY configuration to that of the SAVE configuration. The following efficiency gains were recorded: 470 % for counting tasks, 271 % for finding significant dates, and 69 % for finding packages with certain changes. This equates to an overall efficiency gain of 170 % for the eCITY configuration compared to the SAVE configuration. Knodel et al. [18] conduct a similar empirical experiment where they also evaluate the results with respect to the effect size, a representation of the difference in mean values as compared to the standard deviation. Similarly, we calculate the standard deviation in Table 2 using the formula of Hedges et al. [19] and claim an overall effect size of 5.77 standard deviations to be highly significant [20].

The corresponding efficiency hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is significantly more efficient than the SAVE configuration.

Table 2. Efficiency gain and effect size (mm:ss)

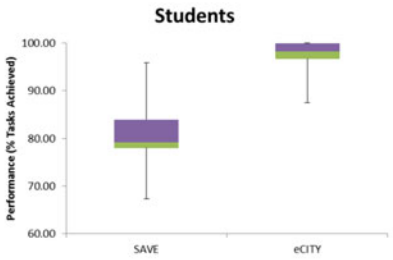
	Mean A	Mean B	Dev.	Gain	Effect
Counting	11:20	1:59	1:20	469.91	6.99
Find Date	8:09	2:12	1:42	270.78	3.49
Find Package/s	13:50	8:10	1:55	69.37	2.97
Overall	33:18	12:21	3:38	169.65	5.77

– $\mathbf{H_{0,1}}$: $\text{Time}_{\text{SAVE}} <= \text{Time}_{\text{eCITY}}$
Test input: the time required to complete tasks
Test: Mann-Whitney U test (one-tailed)
Result: $Z = 5.46$, $p < 0.001$; thus, the null hypothesis is rejected

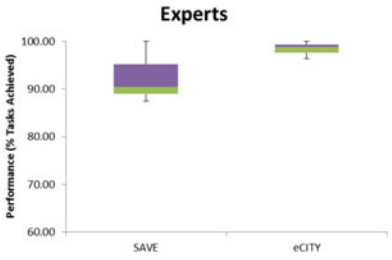
Effectiveness Results. The accuracy with which each participant completed a task was measured to compare the effectiveness with which the participants performed using either one of the two configurations. As such, the answers to the tasks were evaluated against the expected results. The maximum score was 24 points (6 points for Counting tasks, 6 points for Find Date tasks, and 12 points for Find Package/s tasks). The results are presented in the box-plots of Fig. 5. A similar pattern to the efficiency evaluation is followed: Fig. 5a depicts the effectiveness of the participants with respect to the SAVE and eCITY configurations, while Fig. 5b and c examine the effectiveness of students and experts respectively. The following observations can be made from the results presented:

– Independent from being experts or students, members of the eCITY group were on average more effective.
– There is a larger variance from the average value in the SAVE group than the eCITY group.

(a) Overall Effectiveness



(b) Effectiveness of Students



(c) Effectiveness of Experts

Fig. 5. Effectiveness box-plots (x-axis: left SAVE, right eCITY, y-axis: % tasks achieved)

Table 3. Participants effectiveness (% of task results achieved)

		Mean	Median	Std. deviation
Counting	SAVE	70.00	66.67	16.75
	eCITY	99.21	100.00	3.64
Find Date	SAVE	70.00	50.00	29.91
	eCITY	97.62	100.00	10.91
Find Package/s	SAVE	95.95	97.62	5.34
	eCITY	96.94	97.62	3.03

- Experts were on average more effective than students using the SAVE configuration, however, the students performed almost as well as experts with the eCITY configuration.
- The variance of the results is larger for students than experts.

The mean, median, and standard deviation values of the effectiveness with which the participants performed are presented in Table 3. Again, to examine the level of effectiveness reached for each type of tasks, the results have been broken down for each type. Our findings show that panning and zooming on a large view to count the changes leads to more errors than using the chart and slider combination of the eCITY configuration. It was difficult for the participants to keep track of which components they had already counted. The participants also made more mistakes with the SAVE configuration while looking for dates with the most changes as they found it difficult to compare the mental maps of the densities of each updated diagram. It was hard for them to distinguish between the relatively few changes between two different time stamps. For the last category of tasks, the participants performed equally well to find certain packages that were modified in a particular time frame. However, there was a larger variance from the average effectiveness using the SAVE configuration as compare to the eCITY configuration.

We further examine the effectiveness results from the SAVE (A) and eCITY (B) configurations in Table 4. The following effectiveness gains were recorded: 29 % for counting tasks, 28 % for finding significant dates, and 1 % for finding packages with certain changes. This equates to an overall effectiveness gain of 15 % for the eCITY configuration compared to the SAVE configuration. The effect size for all tasks is 2.52 standard deviations, which according to Cohen [20] is considered to be a large effect size.

The corresponding effectiveness hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is significantly more effective than the SAVE configuration.

- $\mathbf{H_{0,2}}$: $\text{Effectiveness}_{\text{SAVE}} >= \text{Effectiveness}_{\text{eCITY}}$
  Test input: the effectivess (accuracy) of the task results
  Test: Mann-Whitney U test (one-tailed)
  Result: $Z = -4.81$, $p < 0.001$; thus, the null hypothesis is rejected

Table 4. Effectiveness gain and effect size (%)

	Mean A	Mean B	Dev.	Gain	Effect
Counting	70.00	99.21	10.04	29.44	2.91
Find Date	70.00	97.62	20.18	28.29	1.37
Find Package/s	95.95	96.94	4.16	1.02	0.24
Overall	82.98	97.68	5.83	15.05	2.52

Acceptability Results. The participants of the experiment were asked to give their personal assessment on the acceptability of the two configurations. As mentioned in Sect. 4.2, we applied certain parameters of the performance and effort expectancy dimensions of the Technology Acceptance Model. The participants responses ranged from 1 (Strongly Disagree) to 5 (Strongly Agree) for each parameter. The aggregated mean, median, and standard deviation values for the dimensions mentioned above and the average overall acceptability score are presented in Table 5. These results indicate that the average values in all of these cases are very similar for both configurations.

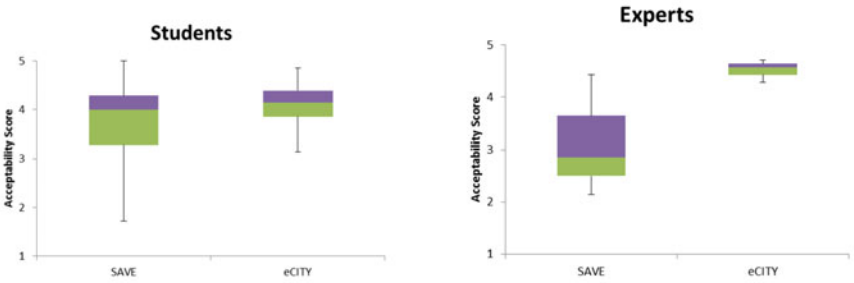
Using the box-plots of Fig. 6, we can make the following observations regarding the acceptability scores for each configuration by participant type:

- The average acceptability score for the experts differs more than the score of the students.
- Independent from being experts or students, the acceptability score of the eCITY group was higher on average.
- Students voted higher for the acceptability of the SAVE configuration than experts, whereas experts voted higher still for the eCITY configuration.
- The difference in expert acceptability scores for the two configurations is higher than the difference in student acceptability scores.

At first the last observation was a bit surprising, however, after careful consideration it was in fact quite a reasonable result. We claim this to be the case due to the fact that the student evaluations were blind, that each student used only one configuration, and most importantly the experts were more aware of the shortcomings in the SAVE configuration.

Table 5. Acceptability response for the comparison of configurations

		Mean	Median	Std. deviation
Performance E.	SAVE	3.72	4.33	1.20
	eCITY	4.38	4.33	0.43
Effort E.	SAVE	3.56	3.75	0.86
	eCITY	3.94	4.00	0.52
Overall	SAVE	3.63	3.93	0.94
	eCITY	4.13	4.14	0.42



(a) Students Acceptability Score (b) Experts Acceptability Score

Fig. 6. Acceptability response of students and experts (x-axis: left SAVE, right: eCITY, y-axis: acceptability score)

The corresponding acceptability hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration achieves the same acceptability as SAVE configuration.

- $\mathbf{H_{0,3}}$: $\text{Acceptability}_{\text{SAVE}} >= \text{Acceptability}_{\text{eCITY}}$
  Test input: results of the acceptability questionnaire
  Test: Mann-Whitney U test (one-tailed)
  Result: $Z = -1.55$, $p = 0.0608$; thus, the null hypothesis cannot be rejected

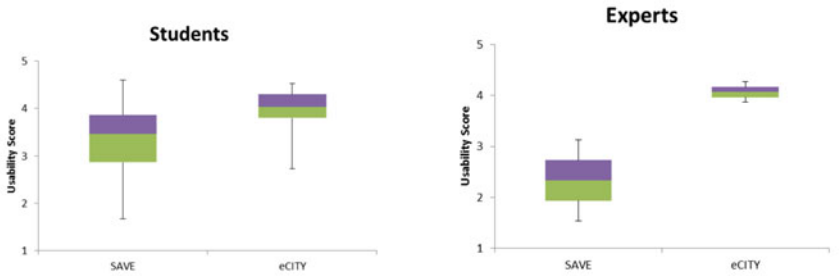
Usability Results. The participants of the experiment were also questioned for an assessment on the usability of the two configurations. Earlier in Sect. 4.2, we defined Usability in terms of various Utility, Intuitiveness, Learnability, and Personal Effect parameters. Usability was evaluated in the same manner as Acceptability where the participants used an ordinal scale with five values for their responses. The aggregated mean, median, and standard deviation values for the dimensions mentioned above and the average overall usability score are presented in Table 6. These results indicate that the average values in three of the four dimensions are marginally higher for the eCITY configuration, while the average value in the fourth dimension (Personal Effect) is significantly more. Overall, the average usability score of the SAVE configuration leans towards *neutral* on the scale, while the eCITY configuration tilts towards *agree* on the scale.

By examining Fig. 7 we can make the following remarks about the usability scores for each configuration by participant type:

- The average usability score for the experts differs more than the score of the students.
- Independent from being experts or students, the usability score of the eCITY group was higher on average.

Table 6. Usability response for the comparision of configurations

		Mean	Median	Std. deviation
Utility	SAVE	3.48	3.67	0.97
	eCITY	4.22	4.00	0.53
Intuitiveness	SAVE	3.29	3.75	0.84
	eCITY	3.81	4.00	0.61
Leanability	SAVE	3.30	4.00	1.22
	eCITY	4.14	4.14	0.65
Personal Effect	SAVE	3.02	2.90	0.94
	eCITY	4.03	4.20	0.43
Overall	SAVE	3.24	3.30	0.79
	eCITY	4.99	4.07	0.40



(a) Students Usability Score (b) Experts Usability Score

Fig. 7. Usability response of students and experts (x-axis: left SAVE, right: eCITY, y-axis: usability score)

- On average, the students voted a whole ordinal scale higher for the usability of the SAVE configuration than experts.
- The difference in expert usability scores for the two configurations is higher than the difference in student usability scores.

The corresponding usability hypothesis was tested using the Mann-Whitney-U-Test. These results indicate that on average the eCITY configuration is perceived to be more useful than the SAVE configuration.

- $\mathbf{H_{0,4}}$: $\text{Usability}_{\text{SAVE}} >= \text{Usability}_{\text{eCITY}}$
  Test input: results of the usability questionnaire
  Test: Mann-Whitney U test (one-tailed)
  Result: $Z = -3.33$, $p < 0.001$ thus, the null hypothesis is rejected

Threats to Validity. Threats to internal and external validity are discussed in this section. Threats to internal validity refer to conditions that constrain the confidence level of the results, while threats to external validity are conditions that limit the generalizability of the study.

Internal Validity: We tried to prevent the effects of confounding variables, between a tool (independent variable) and a dependent variable (efficiency, effectiveness, acceptability, and usability). Student participants received the same training for either the SAVE or eCITY configurations. None of the thirty five students had any prior experience with SAVE or eCITY and were randomly distributed in to one of the two study groups. We then ran the experiment with SAVE experts to make sure that the results were representative. The three experts had no experience with the eCITY configuration and were given the same training as the students. As our statistical results show, there was not a lot of difference in the dependent variable within each configuration. Further, we verified that the tasks for each configuration were performed with similar performance and time. The acceptability and usability aspects we choose are quite reliable, but certainly there are other measures. We applied the widely accepted Technology Acceptance Model and the usability study of Nestler et al. to break down the two aspects into 22 distinct measures. Lastly, we argue that the Apache Tomcat servlet container is representative of a real-world software system. It is used in a diverse range of industries and organizations to power numerous large-scale web applications.

External Validity: A real and complete system, the Apache Tomcat, was evaluated by the participants using the two configurations. Although, our results show that our solution was as acceptable as the original configuration, but more efficient, effective, and perceived as more useful, it would be interesting to repeat the experiment with other representative systems to check if the validity holds. Further, since a small number of participants were involved, the experiment should be repeated with more participants. These two measures should ensure that our approach has a practical value for other projects that look to improve analysis of software evolution.

5 Summary and Outlook

The quality of a software system's architecture is one of the most significant factors for its success, a characteristic that is even more prevalent in the development of large and complex software systems. A precursor to accessing the quality of the underlying software architecture is to comprehend the structure of its elements. Further, the structure of the architecture continuously undergoes changes to adapt to functional or quality requirements, making the comprehensibility of the existing architecture an effort intensive activity. These problems are addressed through the use of visualization tools that examine the evolution of software architectures. The experiment we conducted showed the significance of

employing appropriate visualization techniques and metaphors to conduct such analyses. By means of a more appropriate configuration, participants achieved an average gain of 170 % in the efficiency and an average gain of 15 % in the effectiveness in basic software architecture evolution tasks.

For the realization of our goals, we worked with Product Line Architects at the Fraunhofer IESE to augment their workflow with additional views of their data. In this paper, we introduced an implementation of an interactive evolving city view through a non-formal modeling process that proved to be both efficient and effective. Initially, the stakeholders were reluctant with the idea of a city layout as it deviated from traditional node-link layouts. However, they revised this opinion by working with the eCITY prototype and even found the interactive city to be both natural and intuitive. Having said that, by not following a formal modeling process there are certain details that we need to address further. The most critical limitation is that we currently do not examine how the software architecture's interdependency evolves over time. While the experts can still use their traditional workflows to examine these dependencies, "it would be quite nice to add them to the city view". Other possible improvements include a better mechanism for locating packages and classes such as an integrated search engine, highlighting of suburbs undergoing change in a time interval using multiple fish-eye views, the implementation of an alternate color scheme to address colorblindness, and an interface scheme that allows users to perceive and interact with focused and contextual views through overview-and-detail or focus-and-context. We are also aware that at times the analyst needs to directly compare the architecture at two disjoint points in time and will be addressing this in the near future.

Overall, the results show that our solution was in average as acceptable as the original configuration, but was more efficient, effective, and perceived as more useful. These results are a positive indication of the quality of our solution in terms of efficiency, effectiveness, acceptance, and usability. However, further empirical studies are required for confirming these results and deriving conclusive outcomes; i.e. using a larger sample, analyzing different systems, etc.

Acknowledgment. The authors wish to thank the members of both the Computer Graphics and Human Computer Interaction Group at the University of Kaiserslautern and the Product Line Architectures at the Fraunhofer IESE for their cooperation. This work was supported by the Innovation Center Applied System Modeling through the Applied System Modeling for Embedded Software Systems project.

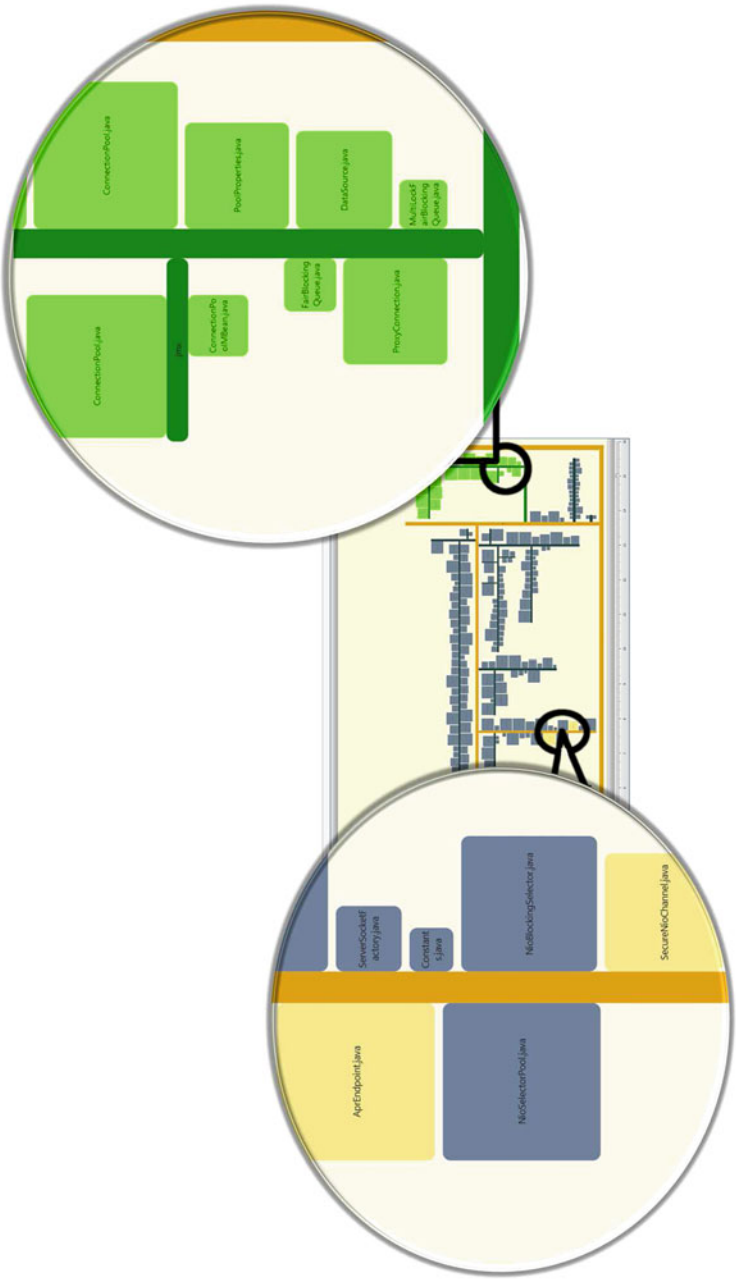
A Acceptability Questions

- **Performance ExpectanQuestiocy:**
  i. *I would find the visualization useful in my job*
  ii. *Using the visualization enables me to accomplish tasks more quickly*
  iii. *Using the visualization increases my productivity*
- **Effort Expectancy:**
  i. *My interaction with the visualization would be clear and understandable*
  ii. *It would be easy for me to become skillful at using the visualization*
  iii. *I would find the visualization easy to use*
  iv. *Learning to apply the visualization is easy for me*

B Usability Questions

- **Utility**; Productivity:
  i. *I find the visualization highly appropriate to get details on the software's architectural evolution*
  ii. *The visualization supported the handling of all the tasks I needed to perform*
  iii. *I was highly successful in accomplishing the given tasks with the visualization*
- **Intuitiveness**; Affordance, Transparency, Memorability, and Perspicuity:
  i. *The visualization clearly indicated all the possible inputs to me*
  ii. *I find the terms, abbreviations, and symbols used in the visualization easy to understand*
  iii. *I find the visualization to be highly understandable*
  iv. *I find the effects of actions to be highly transparent*
  v. *I find the visualization helped reduce my memory load in accomplishing the given tasks*
  vi. *It was easy for me to find the important commands and actions*
- **Learnability**; Feedback:
  i. *The visualization provided appropriate feedbacks to me when I interacted with it*
- **Personal Effect**; Novelty, Satisfaction, and Stress
  i. *I feel that the visualization provided a novel approach*
  ii. *I was totally comfortable with using the visualization*
  iii. *I find interaction with the visualization to be pleasant*
  iv. *I am satisfied with the information provided to me by the visualization*
  v. *I felt insecure, discouraged, irritated, or stressed while using the visualization*

C Detailed eCITY View (rotated) –
Left: Classes modified and Right: Package inserted



References

1. Lehman, M.M., Belady, L.A. (eds.): Program Evolution: Processes of Software Change. Academic Press Professional Inc., San Diego (1985)
2. D'Ambros, M., Lanza, M.: Visual software evolution reconstruction. J. Softw. Maint. Evol. **21**, 217–232 (2009)
3. Petre, M., Quincey, E.: A gentle overview of software visualization. PPIG News Letter, 1–10 (September 2006)
4. Steinbrückner, F., Lewerentz, C.: Representing development history in software cities. In: Proceedings of the 5th International Symposium on Software Visualization, SOFTVIS '10, pp. 193–202. ACM, New York (2010)
5. D'Ambros, M., Lanza, M.: Reverse engineering with logical coupling. In: Proceedings of the 13th Working Conference on Reverse Engineering, pp. 189–198. IEEE Computer Society, Washington, DC (2006)
6. Ghanam, Y., Carpendale, S.: A survey paper on software architecture visualization. Technical report, University of Calgary, pp. 1–10, June 2008
7. Telea, A., Voinea, L., Sassenburg, H.: Visual tools for software architecture understanding: a stakeholder perspective. IEEE Softw. **27**, 46–53 (2010)
8. Khan, T., Barthel, H., Ebert, A., Liggesmeyer, P.: Visualization and evolution of software architectures. In: Garth, C., Middel, A., Hagen, H. (eds.) Visualization of Large and Unstructured Data Sets: Applications in Geospatial Planning, Modeling and Engineering - Proceedings of IRTG 1131 Workshop 2011. OpenAccess Series in Informatics (OASIcs), vol. 27, pp. 25–42. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2012)
9. Knodel, J., Muthig, D., Naab, M., Lindvall, M.: Static evaluation of software architectures. In: Proceedings of the Conference on Software Maintenance and Reengineering, CSMR '06, pp. 279–294. IEEE Computer Society, Washington, DC (2006)
10. Eick, S.G., Steffen, J.L., Sumner Jr, E.E.: Seesoft - a tool for visualizing line oriented software statistics. In: Card, S.K., Mackinlay, J.D., Shneiderman, B. (eds.) Readings in Information Visualization, pp. 419–430. Morgan Kaufmann Publishers Inc., San Francisco (1999)
11. Telea, A., Auber, D.: Code flows: visualizing structural evolution of source code. Comput. Graph. Forum **27**(3), 831–838 (2008)
12. Voinea, L., Telea, A., Chaudron, M.R.V.: Version-centric visualization of code evolution. In: Brodlie, K., Duke, D.J., Joy, K.I. (eds.): EuroVis, pp. 223–230. Eurographics Association (2005)
13. Holten, D., van Wijk, J.J.: Visual comparison of hierarchically organized data. Comput. Graph. Forum **27**(3), 759–766 (2008)
14. Collberg, C., Kobourov, S., Nagra, J., Pitts, J., Wampler, K.: A system for graph-based visualization of the evolution of software. In: Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis '03, pp. 77–86. ACM, New York (2003)
15. Eick, S.G., Graves, T.L., Karr, A.F., Mockus, A., Schuster, P.: Visualizing software changes. IEEE Trans. Softw. Eng. **28**(4), 396–412 (2002)
16. Venkatesh, V., Morris, M.G., Davis, G.B., Davis, F.D.: User acceptance of information technology: toward a unified view. MIS Q. **27**(3), 425–478 (2003)
17. Nestler, S., Artinger, E., Coskun, T., Yildirim, Y., Schumann, S., Maehler, M., Wucholt, F., Strohschneider, S., Klinker, G.: Assessing qualitative usability in life-threatening, time-critical and unstable situations. GMS Med. Inform. Biom. Epidemiol., 7(1) (2011)

18. Knodel, J., Muthig, D., Naab, M.: An experiment on the role of graphical elements in architecture visualization. Empirical Softw. Eng. **13**(6), 693–726 (2008)
19. Hedges, L., Olkin, I.: Statistical Method for Meta-Analysis. Academic Press, New York (1985)
20. Cohen, J.: A power primer. Psychol. Bull. **112**, 155–159 (1992)