Chronos: Visualizing Slices of Source-Code History

Francisco Servant and James A. Jones
Department of Informatics
University of California, Irvine, USA
{fservant, jajones}@uci.edu

***Abstract*—In this paper, we present CHRONOS—a tool that enables the querying, exploration, and discovery of historical change events to source code. Unlike traditional Revision-Control-System tools, CHRONOS allows queries across any subset of the code, down to the line-level, which can potentially be contiguous or disparate, even among multiple files. In addition, CHRONOS provides change history across all historical versions (*i.e.,* it is not limited to a pairwise "diff"). The tool implements a zoom-able user interface as a visualization of the history of the queried code to provide both a high-level view of the changes, which supports pattern recognition and discovery, and a low-level view that supports semantic comprehension for tasks such as reverse engineering and identifying design rationale. In this paper, we describe use cases in which CHRONOS may be helpful, provide a motivating example to demonstrate the benefits brought by CHRONOS, and describe its visualization in detail.**

I. INTRODUCTION

The ability to efficiently answer questions about source-code history is a key factor in a software developer's productivity. LaToza and Myers [10] found that developers often ask questions such as *When, how, by whom, and why was this code changed or inserted?* or *How has it changed over time?*

A wide variety of tasks may benefit from the ability to answer questions about source-code history. For example, observing how the source code pertaining to a feature changed over time might help a developer understand the rationale for its current implementation. Also, finding out who was involved in modifying a functionality over time would help developers identify other knowledgeable developers about that functionality. As a final example, discovering multiple artifacts that are modified over the history of the code in a synchronized fashion may suggest an implicit relationship among these artifacts, and further may be used as an exemplar for future edits. Zimmermann *et al.* defined this implicit relationship as *evolutionary coupling* [16].

Revision Control Systems (RCS) (*e.g., CVS*, *Subversion*, *Git*) afford many benefits in terms of supporting software-development tasks. However, they provide limitations for exploration, querying, and discovery of even moderately complex historical events in the evolution of the source code.

First, current RCS tools operate at only two different levels of granularity for the dimension of the program: a source code *file* or the whole *project*. When a developer needs to query the history of source code at finer levels of granularity than a file, she will have to manually filter the historical information that is not related to her lines of code of interest. Additionally, if a developer needs to query the history of source code that involves multiple files, she either needs to query the history of the whole project, or separately query the history of each file of interest and then manually synthesize the results.

Second, current RCS tools allow only two operations over the dimension of history: obtain *one* previous revision, or obtain *all* the revisions. As previously mentioned, a developer may be interested in the history of source code at a different granularity than a whole file, *e.g.,* the history of a set of lines of code. In such cases, only a subset of revisions will contain changes to the lines of interest, and the developer will have to manually filter those revisions that do not contain changes to them. Also, obtaining only one previous revision of the source code file at a time involves a repetitive process that has to be manually performed until all the revisions of the lines of interest are obtained.

To addresses the limitations of current RCS tools, we present our tool CHRONOS. CHRONOS is an implementation of our History Slicing approach [12], which allows developers to select any set of lines of interest, contiguous or disparate, from any set of files. For the selected set of lines of interest, CHRONOS visualizes their complete history, including only the revisions that modified them. With its visualization, CHRONOS facilitates (1) interactive querying of the history of any set of code at any point in time, (2) exploration of the query results to support both a high-level view of the evolution and a detailed view of the changes, and (3) discovery of patterns of change among multiple components of the code.

II. MOTIVATING EXAMPLE

Consider a scenario in which we need to understand how two methods co-evolved, for example, to understand how they are related and thus how they should be co-modified.

To demonstrate the benefits brought by CHRONOS, we use actual results from a real-world program, ASPECTJ. Table I defines the two code selections (S_A and S_B) from two files (F_A and F_B) and the revision. Next, we describe the effort that would be spent in obtaining the complete evolution of S_A and S_B by using a state-of-the-practice tool, Eclipse's Git plug-in, and by using CHRONOS.

Traditional Tool: Eclipse's Git Plug-in. By using Eclipse's Git plug-in, we could request all the revisions of F_A with the *Show History* command, and then manually call the *Compare* command to see the changes performed by each one of the revisions. With this approach, we would have to inspect the whole contents of the file for 81 revisions of F_A and 120 revisions of F_B. However, only 11 revisions of F_A and 21

TABLE I
HISTORY SLICING CRITERION FOR MOTIVATING EXAMPLE.

Program:	ASPECTJ (http://www.eclipse.org/aspectj/), with 8+ years history and ~510KLOCs
Revision ID:	`6c59333620d99c4eed53c17f70d9ba66d157bf64`
Selection S_A:	all lines in `makeJoinPointSignatureForMethod-Invocation()` in `BcelWorld.java` (*i.e.,* File F_A)
Selection S_B:	all lines in `mungeNewField()` in `BcelTypeMunger.java` (*i.e.,* File F_B)

revisions of F_B contained changes to the code in which we are interested, *i.e.,* S_A and S_B. With this first approach, we would be manually navigating the dimension of the program by inspecting the contents of all the revisions for a file, and we would be manually navigating the dimension of history by manually identifying the revisions that actually modified the lines of code of interest.

An alternative approach would be to use the *Show Annotations* command instead of the *Compare* command. An annotation shows the latest revision that modified each line of code, therefore allowing us to skip revisions that did not modify S_A and S_B. However, we would still have to check the annotation for every line of code in S_A and S_B — which can be tedious when there are many of them — and manually call the *Show Annotations* command again for every visited revision. As a result, we would still be manually navigating both the dimension of the program and history. In a previous user study [12], when using Eclipse, only 2 participants were able to answer a question about the history of S_A and S_B within a time frame of 10 minutes, with an average time of 9:57 minutes.

***Our New Tool:* CHRONOS.** By using CHRONOS, we would simply select the lines of code in S_A and S_B in Eclipse's editor and select the *Add to History Slicing Seed* command. Then, we would run the *Display History Slice* command. CHRONOS would then show a single visualization with the complete history of S_A and S_B. In a previous user study [12], when using CHRONOS, all of the participants were able to answer a question about the history of S_A and S_B in an average time of 5:19 minutes.

III. VISUALIZATION

We implemented our tool CHRONOS in Java as an Eclipse plug-in and it currently supports the CVS, Subversion and Git revision-control systems. In its visualization, CHRONOS can display the history of any set of lines of code. Also, it displays multiple selections of groups of code lines. Figure 1 depicts an example of how CHRONOS displays the history of two code selections. Below, we describe the components of the visualization and the interaction options in CHRONOS.

Components. Our visualization in Figure 1 is composed of two main parts. At the top, the two horizontal gray bars represent the global timeline. Below the global timeline, are two individual areas, each of which displays the history of one code selection.

Global, Multi-selection Timeline Visualization. The top area of the visualization displays a set of timelines—one for each one

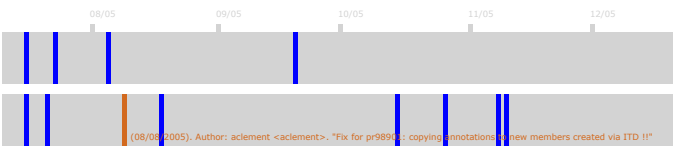

Fig. 2. Global, multi-selection timeline visualization supports pattern recognition and discovery.

of the code selections. A zoomed-in view of the global timeline is depicted in Figure 2. The goal of the global timeline is to allow developers to discover patterns in the evolution of the selected snippets of code. In the global timeline, a developer could quickly answer questions such as: "Was a code snippet modified at a particular point in time but not in others?", or "When were two code snippets changed together, and when were they changed separately?"

The top part of the global timeline is annotated with a *ruler* of equidistant ticks that represent months in time. Time is represented in increasing order from left to right, having the latest dates on the rightmost side of the timeline. The upper gray bar represents the history of the first code selection and the lower gray bar represents the history of the next code selection. A blue mark inside a gray bar denotes a change to its code selection at that time. Also, users can highlight any of the blue marks to show meta-data about a change, particularly its date, author and commit message. Figure 2 shows a change highlighted in orange and its meta-data also in orange text.

A developer could quickly refer to the meta-data of specific changes for a better understanding of an observed change pattern. Additionally, if the developer wants to perform a more in-depth investigation of the code's evolution, she could also explore the actual contents of a change by looking at the individual histories that are depicted below the global timeline.

Individual Histories. The second component of the visualization of CHRONOS are the individual histories for the code selections. We show in Figure 3 a zoomed-in view from Figure 1 into the individual history of a code selection. Each individual history is in turn composed of two parts: the individual timeline at the top and the snapshots of code at the bottom. A *snapshot* of code depicts the contents of the selected lines of code at a point in time when they were changed.

By including the individual timeline at the top, CHRONOS allows users to get a sense of the time that passed between changes. At the same time, by displaying the snapshots of code side by side, CHRONOS allows users to easily compare corresponding lines of code between revisions.

Since the spacing between changes in the timeline and in the snapshots areas is different, CHRONOS connects with a blue line changes in the timeline with their corresponding snapshot. This connection is the only difference between the timeline for a code selection inside the individual history and inside the global timeline.

Similarly to the global timeline, changes can be highlighted in the individual timeline for a quick view of their meta-data and for an easier identification of their corresponding snapshot. A highlighted change and its connection to its corresponding

Fig. 1. Visualization of CHRONOS.

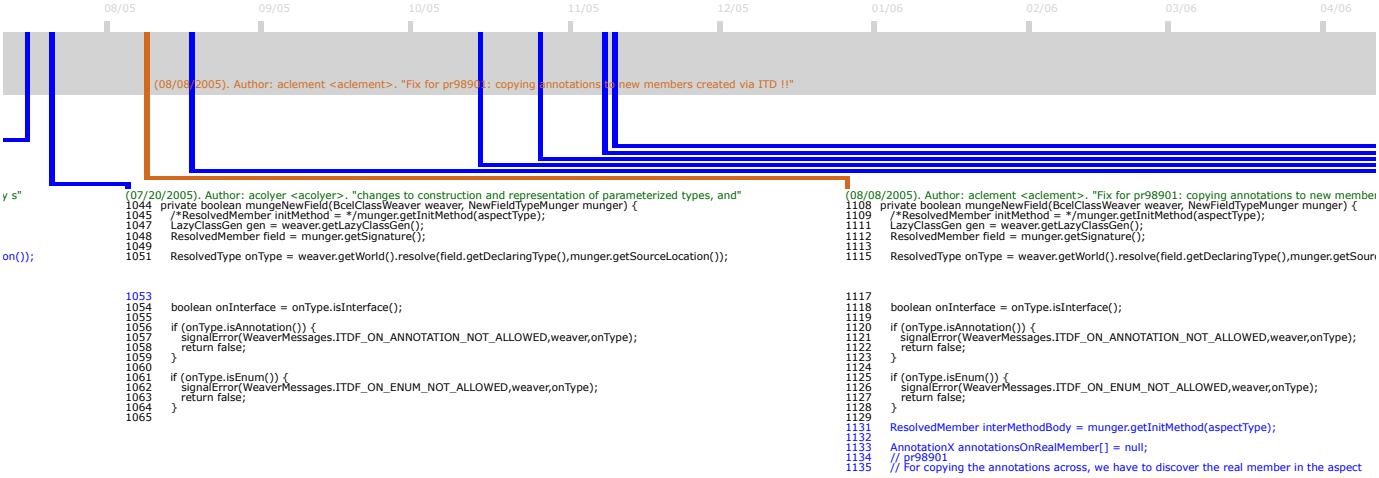

Fig. 3. The individual history of a code selection provides semantically rich detail of code changes and supporting meta-data.

snapshot can be seen in Figure 3 in orange. The individual timeline and the global timeline are connected, so that if one of them is highlighted, the other one is also highlighted, which can be seen in Figure 1.

The individual history of a set of lines of code contains as many snapshots as the number of revisions that contain changes for them. If a file has a revision in which the selected lines of code were not changed, that revision is not displayed by CHRONOS. Each snapshot is represented with gray text for those lines of code that were selected but not modified in that snapshot, and with blue text for the lines of code that were selected and were modified in that snapshot. Consecutive snapshots are represented side by side and aligned to each other for an easier comparison. For example, in Figure 3, lines 1131 to 1135 were added in August 8th, 2005. We know that these lines were added since there are no lines that correspond to them in the previous snapshot of July 20th, 2005.

Finally, each snapshot is also annotated with meta-data of the change in green: the date, author, and commit message. The goal of including the meta-data of a change on top of the snapshot is to aid in the understanding of the rationale of changes. By looking at a snapshot, developers can see at a glance not only the changes that were performed in that revision, but also additional information that may help them understand why the changes were performed.

Interaction. CHRONOS allows three mechanisms for interaction: zooming, panning, and highlighting.

Zooming and Panning. In order to provide users with both a summarized view of the history of source code and a detailed view of the specific changes performed as well as their meta-data, CHRONOS supports panning and zooming on a scalable vector graphic visualization. When CHRONOS is executed, it provides a complete view of the individual histories that correspond to all the sets of lines of code selected, and all the snapshots that exist for them. This summarized view allows developers to have a first quick view of the timing of changes for each individual history and how many lines were changed in each snapshot. An example of this view is depicted in Figure 1. Then, CHRONOS allows zooming without loss of quality. In the same manner, CHRONOS allows users to move to any part of the visualization to focus on and explore different aspects of it. Examples of detailed views are Figures 2 and 3.

Highlighting. As mentioned before, CHRONOS allows the user to highlight a change when they hover the mouse over it.

When a change is highlighted, CHRONOS shows the meta-data corresponding to it and changes the color of the link to its corresponding snapshot to orange. With this feature, users can see properties of changes quickly from the timeline without having to pan and zoom to re-focus to the snapshot. In addition, highlighting changes makes it easier to follow the connection and identify their corresponding snapshot.

IV. Related Work

Multiple visualizations have been proposed for visualizing the evolution of source code. In terms of the dimension of the program, many authors propose visualizations that show the evolution of the whole software system with a level of detail of source code files, *e.g.,* [5], [9], [13], [15]. In terms of the dimension of history, such techniques allow the visualization of the whole history of the software system.

In order to provide a more detailed understanding, other authors have proposed techniques that visualize the history of source code at finer levels of granularity. Some techniques display evolution information about source code methods [6]–[8]. Hassan and Holt [6] and Holmes and Begel [8] annotate source code methods with additional information taken from commit operations. Hattori *et al.* [7] list all the changes that affected a source code method. Then, users need to select one change at a time to see the *diff* that it caused. In terms of the dimension of history, Hassan and Holt's and Holmes and Begel's approaches display the complete history of the source code method. Hattori *et al.*, however, offer a finer-grained visualization of the dimension of history, by displaying changes captured in the IDE between commit operations.

Other techniques provide a visualization at the line-level granularity, *e.g.,* [1]–[3], [11], [14]. Ball and Eick [1] display aggregated evolution information for every line of code. They annotate a SeeSoft [4] visualization with color codes to indicate historical properties, such as last author, code age, and ratio of bug-fixing changes to feature-addition changes. Chen *et al.* [3] display those lines in the current revision of code which have ever been changed with a commit message that matches a user-specified query. Voinea *et al.* [14] join multiple SeeSoft views of the source code to represent the evolution of a source-code file with line-level granularity. They also allow users to see the *diff* between two consecutive revisions. Lommerse *et al.* [11] extend Voinea *et al.*'s approach by allowing the inclusion of the evolution of multiple files. Bradley and Murphy [2] display, for each line of code, information about its last change, such as its author, date, and commit message. In terms of the dimension of history, Ball and Eick, Voinea *et al.*, and Lommerse *et al.* display information about the whole history of a file (Ball and Eick do so in an aggregated form), and Chen *et al.* and Bradley and Murphy display information only about the last change to each line of code.

In contrast, CHRONOS is the only tool that allows the visualization of all and only those revisions that affected a specified set of lines of code and that shows the meta-data as well as the contents of the code for said revisions in a single visualization.

V. Conclusions

In this paper, we described our visualization, which is implemented in our CHRONOS tool. CHRONOS provides a zoomable user interface, global and local timeline visualizations, detailed semantic source-code change information and development meta-data, and interaction mechanisms to support software-development tasks that require understanding of complex change-history events. With these mechanisms, CHRONOS enables: (1) precise, granular, and flexible querying of source-code changes; (2) exploration of the source-code history to identify detailed and semantically rich information about the changes and relevant meta-data; and (3) discovery of event patterns across the evolution for source code of interest.

VI. Acknowledgements

This material is based upon work supported by the National Science Foundation under award CCF-1116943.

References

[1] T. Ball and S. G. Eick. Software Visualization in the Large. *Computer*, 29(4):33–43, 1996.

[2] A. W. J. Bradley and G. C. Murphy. Supporting Software History Exploration. In *International Working Conference on Mining Software Repositories*, pages 193–202, 2011.

[3] A. Chen, E. Chou, J. Wong, A. Y. Yao, Q. Zhang, S. Zhang, and A. Michail. CVSSearch: Searching through Source Code using CVS Comments. In *International Conference on Software Maintenance*, pages 364–373, 2001.

[4] S. Eick, J. L. Steffen, and E. E. Sumner Jr. Seesoft-A Tool for Visualizing Line Oriented Software Statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992.

[5] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse. How Developers Drive Software Evolution. In *Workshop on Principles of Software Evolution*, 2005.

[6] A. E. Hassan and R. C. Holt. Using Development History Sticky Notes to Understand Software Architecture. In *International Workshop on Program Comprehension*, pages 183–192, 2004.

[7] L. Hattori, M. Lungu, and M. Lanza. Replaying Past Changes in Multi-developer Projects. In *Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 13–22, 2010.

[8] R. Holmes and A. Begel. Deep Intellisense: a Tool for Rehydrating Evaporated Information. In *International Working Conference on Mining Software Repositories*, pages 23–26, 2008.

[9] M. Lanza and S. Ducasse. Understanding Software Evolution using a Combination of Software Visualization and Software Metrics. In *Langages et Modèles à Objets*, pages 135–149, 2002.

[10] T. D. LaToza and B. A. Myers. Hard-to-Answer Questions about Code. In *Evaluation and Usability of Programming Languages and Tools*, pages 8:1–8:6, 2010.

[11] G. Lommerse, F. Nossin, L. Voinea, and A. Telea. The Visual Code Navigator: An Interactive Toolset for Source Code Investigation. In *IEEE Symposium on Information Visualization*, pages 24–31, 2005.

[12] F. Servant and J. A. Jones. History Slicing : Assisting Code-Evolution Tasks. In *Foundations of Software Engineering*, pages 43:1–43:11, 2012.

[13] L. Voinea and A. Telea. Visual Querying and Analysis of Large Software Repositories. *Empirical Software Engineering*, 14(3):316–340, 2009.

[14] L. Voinea, A. Telea, and J. J. van Wijk. CVSscan: Visualization of Code Evolution. In *ACM Symposium on Software Visualization*, pages 47–56, 2005.

[15] R. Wettel and M. Lanza. Visualizing Software Systems as Cities. In *International Workshop on Visualizing Software for Understanding and Analysis*, pages 92–99, 2007.

[16] T. Zimmermann, P. Weibgerber, S. Diehl, and A. Zeller. Mining Version Histories to Guide Software Changes. In *International Conference on Software Engineering*, pages 563–572, 2004.