Chronia: Visualizing How Developers Change Software Systems

In Proceedings of European Conference on Software Maintenance and Reengineering (CSMR 2006)

Mauricio Seeberger Adrian Kuhn Tudor Gîrba
Software Composition Group
University of Berne, Switzerland

Stéphane Ducasse
LISTIC
Université de Savoie, France

Abstract

To understand a certain issue of the system we want to ask the knowledgeable developers. Yet, in large systems, not every developer is knowledgeable in all the details of the system. Thus, we would want to know which developer is knowledgeable in the issue at hand. In this paper we present the Chronia *tool that implements the* Ownership Map *visualization to understand when and how different developers interacted in which way and in which part of the system.*

1 Introduction

Software systems change over time, and even if the original documentation exists, it might not reflect the code anymore. In such situations, it is crucial to get access to developer knowledge. As systems grow larger, not all developers know about the entire system, thus, we need to know which developer is knowledgeable in which part of the system.

In our approach, we assume that the original developer of a line of code is the most knowledgeable in that line of code. We determine the owner of a piece of code as being the developer that owns the largest part of that piece of code. We make use of the ownership to provide a visualization that helps to understand how developers interacted with the system [4].

We implemented our approach in *Chronia*, a tool built on top of the Moose reengineering environment [2]. Our aim was to provide a solution that gives fast results. Our approach relies only on information from the CVS log without needing to check out the whole repository. As a consequence, we can analyze large systems in a very short period of time, making the approach usable in the early stages of reverse engineering.

2 Chronia

Figure 1 presents the details of the *Ownership Map* [4]: each line represents a history of a file, and each circle on a line represents a change to that file. The color of the circle denotes the author that made the change. The size of the circle reflects the size of the change, and the color of the line denotes the author who owns most of the lines of code of the file in that period.

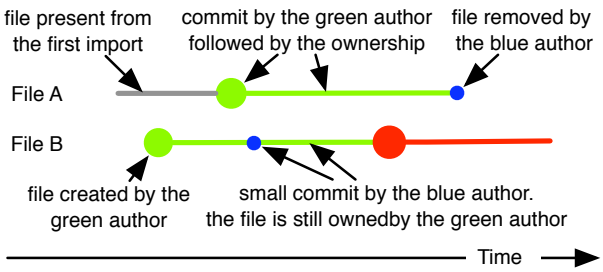

Figure 1. Details of the Ownership Map.

We implemented our approach in *Chronia*, a tool built on top of the Moose reengineering environment [2]. Figure 2 emphasizes the interactive nature of our tool. Contrary to similar approaches [5], we give a semantic order to the file axis by clustering the files based on their history of changes: files committed in the same period are related [3, 7].

On the left of Figure 2 we see *Chronia* visualizing the overall history of the project, which provides a first overview. Since there is too much data we cannot give the reasoning only from this view, thus, *Chronia* allows for interactive zooming. For example, in the window on the lower right, we see *Chronia* zoomed into the bottom right part of the original view. Furthermore, when moving the mouse over the *Ownership Map*, we complement the view by also showing the current position on both time and file axis are highlighted in the lists on the right. These lists show all file names and the timestamps of all commits. As *Chronia* is build on top of Moose, it makes use of the Moose contextual menus to open detailed views on particular files, modules or authors. For example, on the top right window we see a view with metrics and measurements of a file revision.

The visualization reveals several patterns of developer

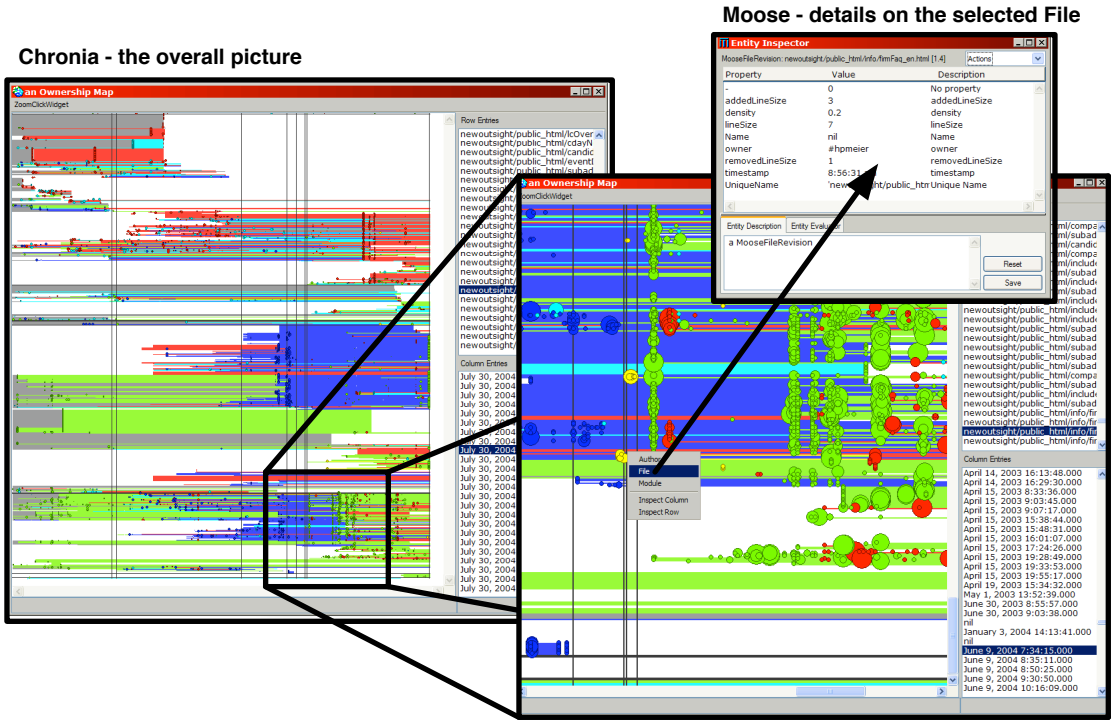

Figure 2. *Chronia* **is an interactive tool.**

behavior (please refer to [4] for more details): Monologue (one author working alone), Dialogue (two authors working together), Takeover (one author aggressively taking over from another) *etc.*. For example, in the zoomed window from Figure 2 we see how the blue lines are transformed into green, because the green author aggressively took over from the blue one. For more details regarding the visualization, please refer to [4].

3 Related Work

A visualization similar to ours is used to visualize how authors change a wiki page [6]. Ball and Eick [1] developed multiple visualizations for showing changes that appear in the source code. Rysselberghe and Demeyer use a scatter plot visualization of the changes to provide an overview of the evolution of systems and to detect patterns of change[5].

Acknowledgments. We gratefully acknowledge the financial support of the Swiss National Science Foundation for the project "RECAST: Evolution of Object-Oriented Applications" (SNF Project No. 620-066077, Sept. 2002 - Aug. 2006).

References

[1] T. Ball and S. Eick. Software visualization in the large. *IEEE Computer*, pages 33–43, 1996.

[2] S. Ducasse, T. Gîrba, M. Lanza, and S. Demeyer. Moose: a collaborative and extensible reengineering Environment. In *Tools for Software Maintenance and Reengineering*, RCOST / Software Technology Series, pages 55–71. Franco Angeli, 2005.

[3] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance 1998 (ICSM '98)*, pages 190–198, 1998.

[4] T. Gîrba, A. Kuhn, M. Seeberger, and S. Ducasse. How developers drive software evolution. In *Proceedings of International Workshop on Principles of Software Evolution (IWPSE)*, pages 113–122. IEEE Computer Society Press, 2005.

[5] F. Van Rysselberghe and S. Demeyer. Studying software evolution information by visualizing the change history. In *Proceedings of The 20th IEEE International Conference on Software Maintenance (ICSM 2004)*, Sept. 2004.

[6] F. Viégas, M. Wattenberg, and K. Dave. Studying cooperation and conflict between authors with history flow visualizations. In *In Proceedings of the Conference on Human Factors in Computing Systems (CHI 2004)*, pages 575–582, Apr. 2004.

[7] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. In *26th Inter-*

national Conference on Software Engineering (ICSE 2004), pages 563–572, 2004.