Chapter 26
Software Evolution Visualization:
Status, Challenges, and Research Directions

Renato Lima Novais
Federal Institute of Bahia, Brazil

Manoel Gomes de Mendonça Neto
Fraunhofer Project Center for Software and Systems Engineering at UFBA, Brazil

ABSTRACT

Software Visualization is the field of Software Engineering that aims to help people to understand software through the use of visual resources. It can be effectively used to analyze and understand the large amount of data produced during software evolution. Several Software Evolution Visualization (SEV) approaches have been proposed. The goals of the proposed approaches are varied, and they try to help programmers and managers to deal with software evolution in their daily software activities. Despite their goals, their applicability in real development scenarios is questionable. In this chapter, the authors discuss the current state of the art and challenges in software evolution visualization, presenting issues and problems related to the area, and they propose some solutions and recommendations to circumvent them. Finally, the authors discuss some research directions for the SEV domain.

INTRODUCTION

Software evolution generally deals with large amounts of data that originates from heterogeneous sources such as Software Configuration Management (SCM) repositories, Bug Tracking Systems (BTS), mailing and project discussion lists. One of the key aspects of software evolution is to build theories and models that enable us to understand the past and present, as well as predict future properties related to software maintenance activities, and hence support software maintenance tasks.

Software Visualization (SoftVis) is the field of Software Engineering (SE) that aims to help people to understand software through the use of visual resources (Diehl, 2007), and it can be

DOI: 10.4018/978-1-4666-6026-7.ch026

effectively used to analyze and understand the large amount of data produced during software evolution. For this reason, many researchers have been proposing Software Evolution Visualization (SEV) tools (Kuhn, Erni, Loretan, Nierstrasz, 2010)(Voinea, Lukkien & Telea, 2007)(Fischer & Gall, 2004)(German, Hindle & Jordan, 2006)(Cepda, Magdaleno, Murta & Werner, 2010)(Eick, Steffen & Sumner Jr, 1992). In general, these tools analyze the evolution of the software with respect to a set of software maintenance related questions.

Despite the goals of the software evolution visualization approaches, most have yet to be used in industrial environments. SEV approaches usually provide good and attractive visual metaphors, but how to use them within the software development process remains an open question. Several SEV tools are proposed as proof of concepts that is not evolved anymore.

This chapter covers Software Evolution Visualization (SEV) approaches, providing information about how SEV research is structured, synthesizing current evidence on the goals of the proposed approaches and identifying key challenges for its use in practice. This text is based on a mapping study that was carried out to analyze how the SEV area is structured (Novais et al., 2013a).

In the following sections we will discuss the current state and challenges in software evolution visualization. We will present issues and problems related to the area, and propose some solutions and recommendations to circumvent them. Finally, we will discuss some research directions for the SEV domain.

BACKGROUND

Software Visualization

Software visualization (SoftVis) can be defined as the mapping of any kind of software artifact in graphic representations (Koschke, 2003) (Roman & Cox, 1992). SoftVis is very helpful because it transforms intangible software entities and their relationships into visual metaphors that are easily interpretable by human beings. Consider coupling among software modules as an example. Using a graph as a visual metaphor, these modules can be represented as nodes and the coupling information can be represented as directed edges to build an intuitive visual metaphor for their dependency. Without a visual representation, the only way to analyze this information would be to look inside the source code or at a table of software metrics, a laborious task or one of great cognitive effort.

There are several classification taxonomies for SoftVis. Some divide SoftVis according to type of visualized object. Diehl (2007), for example, divides software visualization into visualizing the structure, behavior and evolution of the software. Structure refers to visualizing static parts of the software. Behavior refers to visualizing the execution of the software. Evolution refers to visualizing how software evolves (Diehl, 2007). SoftVis can also be classified according to the metaphors it uses to represent software. Among others, visualizations can use iconographic, pixel-based, matrix-based, graph-based and hierarchical metaphors (Keim, 2002) (Ferreira de Oliveira & Levkowitz, 2003).

Software can also be visually analyzed from different perspectives (Carneiro et al., 2008)(Carneiro, Santanna, & Mendonça, 2010)(Carneiro et al., 2010)(Carneiro & Mendonça, 2013). In this case, visualization can be classified according to the point of view it provides to engineers to explore a software system. The perspectives concern to the way in which we look to the software. In the context of software, the perspective may be represented by a set of coordinated views designed to represent a group of properties of the software.

There are several software perspectives (Novais et al., 2013a). Common perspectives in object-oriented programming are Structural, Inheritance and Coupling. Common perspectives in software evolution are Change and Authorship. For example, one might be interested in investigating