The Journal of Systems and Software 128 (2017) 56–71

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Experimentally assessing the combination of multiple visualization strategies for software evolution analysis

Renato Novais[a,c,*], José Amancio Santos[b], Manoel Mendonça[c]

[a] Federal Institute of Bahia, Salvador-Bahia, Brazil
[b] State University of Feira de Santana, Feira de Santana-Bahia, Brazil
[c] Fraunhofer Project Center at UFBA, Salvador-Bahia, Brazil

A R T I C L E I N F O

Article history:
Received 10 August 2016
Revised 13 February 2017
Accepted 7 March 2017
Available online 9 March 2017

Keywords:
Visual strategies
Software evolution visualization
Experimental evaluation

A B S T R A C T

Software engineers need to comprehend large amounts of data to maintain software. Software Visualization is an area that helps users to analyze software through the use of visual resources. It can be effectively used to understand the large amount of data produced during software evolution. A key challenge in the area is to create strategies to consistently visualize the many software attributes, modules and versions produced during its lifecycle. Most of the current visualization strategies seek to present data as a whole, including all available versions of the software in one visual scene. The area lacks strategies visualizing software in detail through the analysis of the evolution of specific software modules. Both strategies are useful, and should be selected according to the task at hand. This work focuses on combining software evolution visualization strategies, experimentally validating the benefits of the approach. Its goal was to build empirical evidence on the use of the combined multiple strategies for software evolution comprehension. It presents an experimental study that exploits the benefits of combining multiple visual strategies of software evolution analysis. The results show that combined visualization strategies perform better in terms of correctness and analysis time.

1. Introduction

Software Evolution has been highlighted as one of the most important topics in software engineering. During software evolution, software engineers need to comprehend large amounts of data. Software Visualization is the area of software engineering that aims to help users to understand the software through the use of visual resources. Our work starts from the premise that software visualization can be effectively used to analyze and understand the large amount of data produced during software evolution (Voinea, 2007; Novais and de Mendonça Neto, 2014). A key challenge in the area is creating strategies to visualize many software versions, modules (packages, classes and methods), and attributes (e.g. metrics) in a two-dimensional computer screen (Novais et al., 2013a).

Currently, software evolution visualization (SEV) approaches focus on presenting data as a whole, including all available versions, showing general information about the evolution process without giving access to fine-grained module information (Gall et al., 1999; D'Ambros and Lanza, 2009; Antoniol et al., 1999; Godfrey and Tu, 2001). Nonetheless, most software engineering tasks require access to detailed information on software modules and subsystems. Furthermore, analyzing all versions at the same time goes against the current state of the practice, which usually focuses on the distinction between two sequential versions.

In previous works (Novais et al., 2013a; Novais and de Mendonça Neto, 2014), we defined the term visual strategies of analysis as the way the visualization is used to portray software evolution. Five strategies were divided into two main types: Temporal and Differential strategies. Temporal Strategies represent the evolution considering several of the available versions for analysis. Differential Strategies take into consideration only two versions to analyze evolution at a given time. They are both important, since each one has benefits and shortcomings in terms of visualizing software evolution. It is, thus, critical to combine them in a systematic way. However, the few works that explore this issue fails to: (i) highlight the importance of the combination; (ii) show the coordination and navigation between views; and most importantly, (iii) validate it experimentally.

In the past six years, we have been incrementally working on an infrastructure, called SourceMiner Evolution, which sup-

* Corresponding author at: Rua Emídio dos Santos, s/n – Barbalho, CEP 40301-015, Salvador-Bahia, Brazil

E-mail addresses: renatonovais@gmail.com, renato@ifba.edu.br (R. Novais), zeamancio@ecomp.uefs.br (J.A. Santos), manoel.mendonca@ufba.br (M. Mendonça).

http://dx.doi.org/10.1016/j.jss.2017.03.006
0164-1212/© 2017 Elsevier Inc. All rights reserved.

ports multiple SEV strategies. It has been used for different purposes, such as the comprehension of software structure evolution (Novais et al., 2011) and feature evolution (Novais et al., 2012a, 2013b), using a Differential Strategy (Novais et al., 2011).

In a controlled experiment for feature evolution analysis, which we will call the *previous experiment*, we compared the differential visualization strategy against an approach based on the Concern Mapper tool (Robillard and Weigand-Warr, 2005). The experiment showed that the visualization approach performed significantly better in terms of correctness, but only slightly better in terms of execution time (Novais et al., 2012a). Based on these observations, we built a new visualization supporting a temporal analysis strategy, and integrated it to the infrastructure (Novais et al., 2012b).

This paper reports an experimental study that we ran to validate the benefits of the differential and temporal strategies combination. Our goal was to build empirical evidence on the use of the combined multiple strategies for software evolution comprehension, and strength the results obtained in the previous experiment. Using the extended version of the tool at three different sites, the participants of the experiment had to perform program comprehension related to feature extraction on the top of five versions of a large-scale software project. The results show that the combined visualization strategies perform better both in terms of correctness and in terms of execution time.

The remainder of this paper is organized as follows. Section 2 presents the background and related work. Section 3 presents the SourceMiner Evolution tool, addressing its views, strategies of analysis and how it can be used to support feature evolution comprehension. Section 4 presents the experimental procedures for the experiment. Section 5 discusses the results of the experiment. Section 6 shows the main findings of the study and analyzes the findings. Section 7 considers the threats to the validity of this research. Finally, Section 8 presents a summary of the work and the directions for future work.

2. Background and related work

This section firstly reviews two topics related to work: Software Evolution Visualization (Section 2.1) and Visual Strategies of Analysis (Section 2.2). Secondly, on Section 2.3, it discusses related work that, like ours, combines visual strategies for software analysis.

2.1. Software evolution visualization

In the context of software development, the use of visualization mechanisms has become more and more important. This is largely due to the large amount of information associated to the evolution of a software system. This type of information can be summarized and represented by different visual paradigms (Keim, 2002; Ferreira de Oliveira and Levkowitz, 2003). Visualization can help to achieve goals such as: identifying critical points (hot-spots) of project erosion (design) and code decay (Ratzinger et al., 2005); finding software elements that are inducing code decay (Eick et al., 2001); and analyzing code anomalies (code smells) in software (Lanza et al., 2005).

The recognition that the use of software visualization can help software evolution analysis is not new. First works in the area appeared about 25 years ago (Eick et al., 1992). However, recently a growing body of relevant work is being developed in the area.

A seminal work in the area was Seesoft (Eick et al., 1992). It mapped lines of code as thin lines on the computer screen. The color of each line indicated an attribute (or statistic, as the authors define) of interest. The main purpose of Seesoft was not visualizing software evolution. Instead, it focused on: (a) static analysis to see where functions are called in the code; and (b) dynamic analysis for profiling through presentation of the memory used and program execution time (Eick et al., 1992). In the context of software evolution visualization, it showed the age of the source code lines (Ball and Eick, 1996). The most recently modified lines were painted red, and the older ones in blue. There was an interpolation between these two colors to represent the rows (lines of code) with intermediate ages.

In 2001, Lanza proposed Evolution Matrix (Lanza, 2001) to visualize software evolution. The Evolution Matrix uses rectangles to represent the software modules and their horizontal position represents different versions of the module. It is possible to see software evolution patterns such as growth of modules or stagnation of their development. The author used an astronomy metaphor to analyze some aspects of software module evolution. In the paper, the evolution of classes was classified according to well-known types of stars (e.g. Pulsar, Supernova, Red Giant, etc.). In 10 years, this paper became the most cited work within the software evolution visualization community (Novais et al., 2013a).

D'Ambros et al. (2009) proposed the Evolution Radar, a view-based approach that integrates logical coupling information both at file and module level. Ripley et al. (2007) proposed a visual approach that allows the gain of a better understanding of the software project evolution. The approach provides an overview of all staff development activities, relating them to the project evolution, using information contained in source code repositories. Similarly, Evolution Storyboards (Beyer and Hassan, 2006) is an animated display of the software history. It enables the developer to identify software modules that are becoming more or less dependent on others. This visualization tries to show code decay symptoms, highlighting candidates for refactoring and also good structures.

Collberg et al. (2003) proposed a software evolution display system based on graphs. This system displays the evolution of the software using a new technique of drawing graphs, which allows one to view large structures using a temporal component. Voinea and Telea (2006b) developed an open framework for consultation, analysis and visualization of CVS repositories of data. This tool uses multiple perspectives to show software evolution in a square matrix. Each column of the matrix shows the evolution of a metric.

Wu et al. (2004a) used Spectographs to explore the evolution of software. Evolution Spectograph combines time spectrum and source code measurement properties in colors to characterize software evolution. The color technique used aims to easily distinguish patterns in evolutionary data.

Considering that data evolution is multidimensional, some authors have proposed the use of animated visualizations. The work of Langelier et al. (2008) is one example. They proposed an approach that uses animated visualization to explore software quality evolution.

In 2012, Kuhn and Stocker proposed the CodeTimeline (Kuhn and Stocker, 2012), an approach that uses visualization to tell the story of software project based on its versioning data. Notes can be used to share events of system life memories as justifications for the design used in the past. The notes can also be used to register more casual project memories, such as photos of any time of interest of the project team being viewed. CodeTimeline uses color and vertical positioning to tell the story of the software. It uses a base layer to show an authorship map, where colors identify the developers, lines represent the history of the file, and bubbles represent the commits for the files.

Several other studies have been proposed over the years. In a systematic mapping study published in 2013 (Novais et al., 2013a), we found 146 works addressing the topic Software Evolution Visualization. MaEny studies found in literature use the information contained in source code repositories, such as the commits and authors. They usually focus on a specific activity of software evolution. They almost always use a single view and a restricted set of