Information and Software Technology xxx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

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

A domain-specific language to visualize software evolution

Alison Fernandez[a], Alexandre Bergel[*,b]

[a] Universidad Mayor de San Simon, Bolivia
[b] Pleiad Lab, DCC, University of Chile, Chile

ARTICLE INFO

Keywords:
Git
History visualization
Domain-specific language

ABSTRACT

Context: Accurately relating code authorship to commit frequency over multiple software revisions is a complex task. Most of the navigation tools found in common source code versioning clients are often too rigid to formulate specific queries and adequately present results of such queries. Questions related to evolution asked by software engineers are therefore challenging at answering using common Git clients.
Objective: This paper explores the use of stacked adjacency matrices and a domain specific language to produce tailored interactive visualizations for software evolution exploration. We are able to support some classical software evolution tasks using short and concise scripts using our language.
Method: We propose a domain-specific language to stack adjacency matrices and produce scalable and interactive visualizations. Our language and visualizations are evaluated using two independent controlled experiments and closely observing participants.
Results: We made the following findings: (i) participants are able to express sophisticated queries using our domain-specific language and visualizations, (ii) participants perform better than GitHub's visualizations to answer a set of questions.
Conclusion: Our visual and scripting environment performs better than GitHub's visualizations at extracting software evolution information.

1. Introduction

Programming activities often require historical information from source code. Consider the following two software evolution tasks [1]: *"Identify the two classes someone changed the most in the past days"* and *"Identify the methods that someone else has also changed"*. Both tasks are likely to be asked by a developer in order to become familiar with someone else's work or to become aware of a team activity. It has been shown that completing these particular two tasks requires dedicated tooling and traditional code versioning systems are suboptimal in that respect [1].

This paper presents and evaluates a visualization framework to explore the evolution of a source code repository. Our approach is based on two main ingredients: (i) a domain-specific language that focuses on the notion of time, Git commit, and metric, and (ii) a visual way to stack adjacency matrices. The language we have designed aims to tailor visualizations in order to address particular questions related to software evolution.

Executing a script in our domain-specific language produces an interactive visualization. As a visual support, we employ MultiPile [2] as a compact way to summarize and navigate through a set of matrices. MultiPile was proposed as a visualization to explore temporal patterns in dynamic graphs. It employs a natural and intuitive analogy of piling adjacency matrices, each matrix representing a temporal snapshot. MultiPile was designed to help neuroscientists. Our article is about assessing MultiPile to solve software evolution problems.

Contributions. This paper makes the following contributions:

- We present GitMultipile, a domain-specific language coupled with stacked adjacency matrices to produce interactive visualizations.
- We evaluate GitMultipile using two experiments: (i) a first controlled experiment focusing on the expressiveness of our domain-specific language, (ii) a second controlled experiment to compare visualizations of GitMultipile against the ones of GitHub. For both experiments, we observed and monitored the participant activities.

Findings. We made the following findings:

- The language offered by GitMultipile is more efficient than Excel at retrieving data from a simple CSV sheet.
- Participants are more efficient at defining and using visualizations with GitMultipile than using GitHub to answer a set of software

* Corresponding author.
E-mail address: abergel@dcc.uchile.cl (A. Bergel).

https://doi.org/10.1016/j.infsof.2018.01.005
Received 23 May 2017; Received in revised form 19 December 2017; Accepted 10 January 2018

evolution questions.

- By observing participants during our experiments, we identified some obvious limitations of the navigation tools offered by GitHub.

Paper outline. Our paper is structured as follows:

Section 2 describes Multipile, the visual foundation used in our work. Section 3 presents GitMultipile, our approach to assess Git-based repositories using a domain-specific language and Multipile. Section 4 illustrates the use of GitMultipile on two large Git repositories. Section 5 discusses the methodology we use to evaluate GitMultipile. Section 6 evaluates the expressiveness of our language by using a controlled experiment. Section 7 compares the visualization produced by some participants against the visualizations of GitHub. Section 8 lists some observations of our participants during their activity. Section 9 lists the threats to validity our work may be subject to. Section 10 presents the work related to this paper. Section 11 concludes and outlines our future work.

2. Background: stacking adjacency matrices

Matrix pile. Adjacency matrices are often used to visualize edges between software related components [3–5]. Each element of a matrix indicates whether two elements are related. A matrix is made of edge weights. Fig. 1 gives three adjacency matrices. On this contrived example, each matrix is squared and has a size of 4. Matrix 1, located on the left, indicates that element B is connected to element D, while D is connected to A and C to A. We assume that the three matrices represent the evolution in time of the graph composed of the nodes A, B, C, and D.

A matrix pile, as proposed by Bach et al. [2], is a structure that stacks adjacency matrices. A matrix pile is the superposition of stacked adjacency matrices. All the matrices that belong to a same pile have the same dimension and same object values for both axes.

Fig. 2 represents a piled matrix obtained from the three matrices given in Fig. 1. A piled matrix is made of three distinct parts. Part 1 is a matrix showing the superposition of the three matrices. This superposition is called the "coverage matrix" and the weight of element C_{ij} is the average weight of the same cell in all the matrices:

$$C_{ij} = \frac{1}{T}\left(\sum_{n=1}^{T} M_{ij}^{n}\right)$$

where M^n is a stacked matrix and T the number of stacked matrices.

Part 2 and Part 3, called top-preview and left-preview, respectively, are a small visual summary of the piled matrices. The previews summarize the content of the pile, each thin bar corresponding to a matrix. The top-preview is made of three thin horizontal bars, each representing a piled matrix. The order of piled matrices goes from bottom to top. Each horizontal bar has n parts, each summarizing a column and n is the number of columns of the coverage matrix. The summary of a part is obtained from the number of the weights greater than 0. Similarly, the left preview summarizes each row of the piled matrices.

A preview is also a navigation widget: locating the mouse above a stacked matrix summary (*i.e.,* thin bar), has the effect to replace the coverage matrix by the actual pointed matrix. Fig. 3 illustrates this point: locating the mouse cursor on the top line in the preview replaces the coverage matrix with Matrix 3, given in Fig. 2.

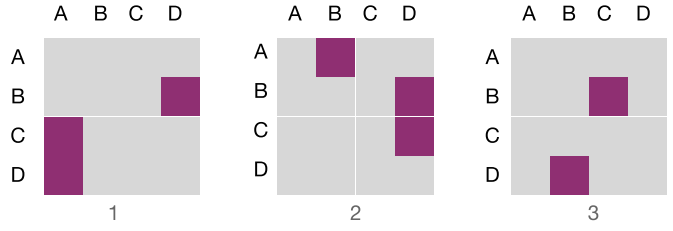

Fig. 1. Three adjacency matrices.

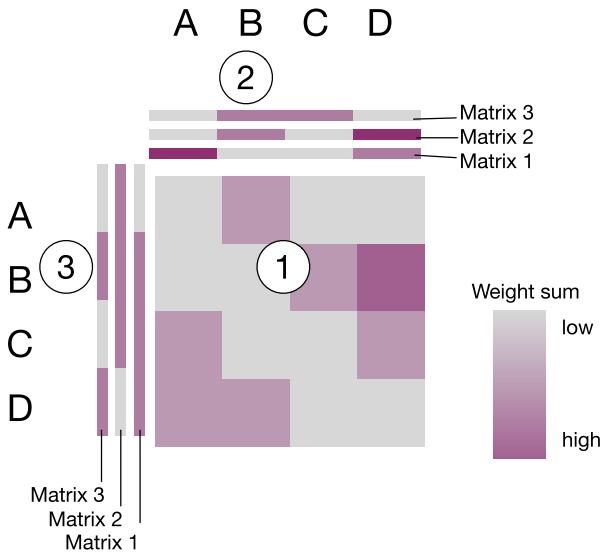

Fig. 2. A piled stack of matrices, obtained from the three matrices given in Fig. 1.

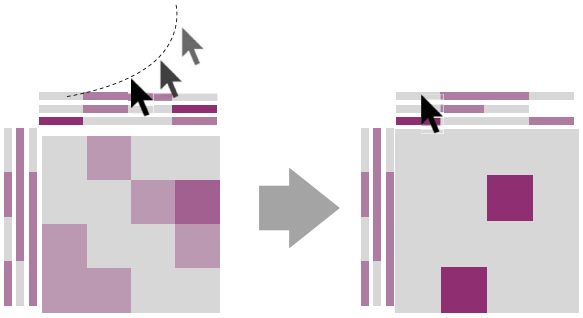

Fig. 3. The previews support navigation in a matrix stack.

Two or more matrices can be piled to produce a pile matrix. Note that the original definition of pile matrix [2] considers the coverage matrix and the top preview (Part 1 and 2 of Fig. 2). We extended this original definition with a left preview (Part 3).

Timeline. A visualization may be composed of several stacks of matrices and some non-stacked matrices. A timeline represents a summary of the whole visualization and is also a way to navigate through the different parts by highlighting parts related to the element in the timeline pointed by the mouse.

Fig. 4 contains a timeline summarizing the three non-stacked matrices. Each stacked and non-stacked matrix has an identifier, and this identifier is used in the timeline to indicate the represented matrix.

Each vertical box of the timeline summarizes a matrix. The first column, with the id 1, represents Matrix 1 given in Fig. 1. In that matrix, the element A has two incoming edges (C and D), and D receives an

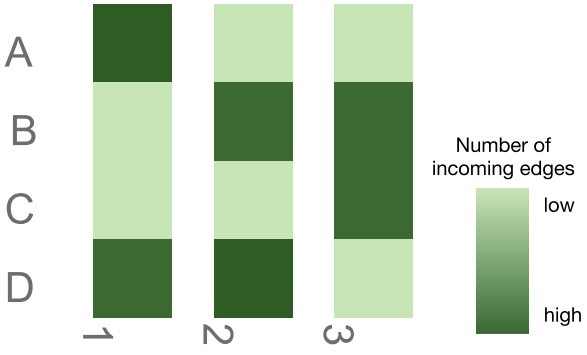

Fig. 4. Timeline for the three non-stacked matrices given in Fig. 1.