Software evolution visualization techniques and methods – a systematic review

Conference Paper · July 2016

DOI: 10.1109/CSIT.2016.7549475

3 authors, including:

Hani Bani-Salameh
Hashemite University
31 PUBLICATIONS **56** CITATIONS

SEE PROFILE

Ashraf H Aljammal
Hashemite University
10 PUBLICATIONS **3** CITATIONS

SEE PROFILE

Software Evolution Visualization Techniques and Methods – a Systematic Review

Hani Bani Salameh
Dept. of Software Engineering
The Hashemite University
Zarqa 13115, Jordan
hani@hu.edu.jo

Ayat Ahmad
Dept. of Software Engineering
The Hashemite University
Zarqa 13115, Jordan
ayat_ahmd1991@yahoo.com

Ashraf Aljammal
Dept. of Computer Science and Applications
The Hashemite University
Zarqa 13115, Jordan
ashrafj@hu.edu.jo

***Abstract*—Background: Software is an important asset for organizations and development teams. It must evolve over time in order to meet different changes in its environment, satisfy the developers' needs, and adapt to new requirements.**

Software developers and team members face difficulties tracking the changes others made to their software. Software visualization is one of the effective techniques that help stakeholders to better understand how software evolves, and which parts of the software are most affected by the change. Many visualization tools and techniques have been introduced by researchers and organizations to facilitate such understanding.

Method: This article presents a systematic literature review (SLR) on software evolution visualization (SEV) tools. The SLR's main focus is to: (1) explore the main target of SEV, (2) analyze the classifications and taxonomies that are used to represent SEV tools, and (3) find out what are the main sources of information used to visualize software's evolution.

Result: 29 papers were analyzed out of 55 papers. The result showed that SEV tools can be classified into five different groups: graph-based, notation-based, matrix-based, and metaphor-based and others. Graph-based are most popular while Notation-based are the least. SEVs focus can be either Artifact-centric visualization, Metric-centric visualization, Feature-centric visualization, or Architecture-centric visualization. The main source of information used to ex-tract information are the software repositories.

Conclusion: This work can help developer, maintainer, researcher to get good knowledge about the state of software evolution and visualization as a whole.

Keywords—Evolution; History; CVS; Repository; Systematic Literature Review (SLR); Software; Tools; Visualization; SEV.

I. INTRODUCTION

In practice, software systems evolve over time in order to respond to any change in their environment. Change is inevitable in any software system, but it is very risky and somehow expensive. It is a major challenge for developers and maintainers to keep the software operational with high quality; software developers face other challenges. For example, over time and with excessive changes, software becomes complex and difficult to understand; tracking the changes is not an easy task, and that is why many research have adopted software systems visualization as a way to track how software evolves during its lifetime cycle, and who is responsible for the change.

As mentioned earlier, visualizing software evolution became an important factor that supports and leads to the software's quality assurance. The increasing demand for quality products showed the huge need for SEV and the tools that support it.

This article presents a systematic literature review (SLR) on SEV tools that help developers, maintainers, and researchers better understand and gain more knowledge about *(1) what are the main targets of SEV tools?, (2) what visualization techniques are mostly used by developers? ,* and *(3) what are the main sources of information used to visualize software evolution?*

Researchers spent much effort studying and analyzing such tools. Breivold et al. [1] presents a systematic review of architecting for software evolvability. The objective is to obtain an overview of the existing approaches to analyze and improve software evolvability at architectural level, and to investigate the impact on research and practice. Shahine et al. [2] present a systematic review of software architecture visualization techniques by analyzing 53 papers and identifying 10 purposes for using visualization techniques in software architecture.

The rest of this paper is organized as follows. Section 2 describes the used SLR procedure. Section 3 analyzes the collected data and presents the results. Finally, Section 4 discusses the results and concludes the paper.

II. SYSTEMATIC REVIEW PROCEDURE

The SLR is carried out by following the guidelines presented by Kitchenham and Charters [3]. This study aims to define a process that identifies, interprets, and evaluates the available and related studies to three research questions related to the area of SEV and the used tools.

It aims to summarize and provide an overview of the current research on *"what software evolution visualization methods and techniques have been reported in the literature?"* To achieve this goal, a set of research questions are suggested in order to be investigated and answered through this SLR.

Subsection 2.1 shows the research questions and what motivates them. The rest of this section covers the main elements of the SLR process.

A. Question(s) Formalization

The SLR is guided by the following research questions:

RQ 1. What kinds of visualization techniques are mostly used in software evolution?

To identify different types of SEV techniques/methods used in visualizing software's evolution, and to find out what are the most and least used kinds.

RQ 2. What is the main target of SEV?

To get an overview of the main focus for the SEV tools, and to identify the major issues when applying SEV.

RQ 3. What are the most used sources of information in SEV?

To identify different sources of information used by the different representations of the software evolution, and which are most used and more reliable when extracting the required information

B. Source Selection

This SLR used a search strategy that is composed of three elements as follows:

1. ***Related work search***: to find and retrieve existing surveys and SLRs. This is achieved by using an automatic search through the available digital libraries.
   - ACM Digital Library (http://www.acm.org)
   - IEEE Explore Digital Library (http://ieeexplore.ieee.org/Xplore/home.jsp)
   - ScienceDirect (http://www.sciencedirect.com)
   - SpringerLink (http://link.springer.com)
   - Wiley InterScience (http://onlinelibrary.wiley.com)
2. ***Search terms:*** Search using different combinations of the initial search terms derived from the suggested research questions.

We carried out an initial scoping study to determine the search string and the resources to be searched. We used the following search strings: (Software evolution **OR** software history **OR** software CVS) **AND** (visualize **OR** visualization **OR** diagram **OR** graphic **OR** graphical **OR** model).

3. ***Conference and workshop papers search:*** Searching the existing results that appear in the well-known software evolution and visualization conferences and workshops.

- **Language:** papers published in English are used in the study.
- **Publication period:** papers published between the years 2002 and 2014 (December 2014).

C. Relevant Studies

The inclusion criteria for determining whether a study should be considered relevant or not, and whether these studies answer the suggested research questions is done by analyzing the *title, abstract*, and *keywords* from the studies retrieved by the search.

After the first iteration of the SLR, we applied the exclusion criteria to obtain the primary and relevant studies; excluding those studies that were not defined and did not contribute in any way to answer the proposed research questions.

D. Primary Studies Selection

The inclusion criteria are listed below:

- Publications introduce approaches with aspects of software evolution visualization.
- Publications that clearly describe the main target of the SEV.

Publications were excluded if:

- Publications does not answer the research question.
- Identical publications from the same authors or research groups.

E. Quality Assessment

After the initial selection, the process involved two stages to evaluate and assess studies. First, studies were evaluated based on the inclusion and exclusion criteria, and more detailed information analysis process. Second, is to evaluate our studies, and which of these studies are clearly stated and clearly specified the objectives. Also, we excluded a number of papers from the study to avoid repetition during the data extraction process; during this step a few studies are excluded as shown in Table 1.

Table 1. Studies Excluded for Similarity.

Excluded study	Similar to
[4]	[11]
[5]	[21]
[6]	[27]

The search procedure produced 29 relevant studies out of the 55 that were selected as relevant studies in the first step.

Figure 1 presents the number of selected studies published per year within the review period from 2002 to 2014.

F. Information Extraction

We applied information extraction process by mapping relevant studies with predefined data extraction: *title, keyword, author, year,* and *publication*. The purpose is to identify the needed information that should be extracted from the selected studies, and support the validation of our research questions. We reviewed and studied each of the primary studies in order

to determine if they match with and help to answer the research questions.

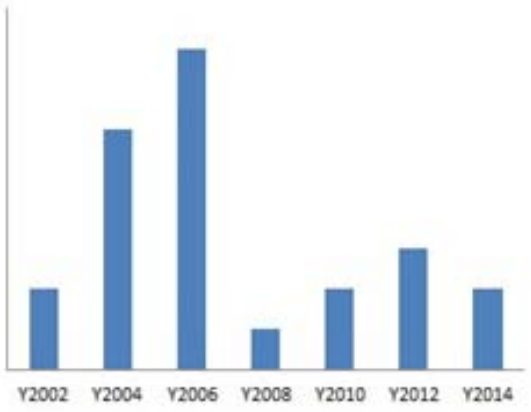

Figure 1. Selected Studies Published per Year.

III. RESULTS

This section reports the obtained results and analysis of the data extracted from the relevant studies, and supports the answers to our research questions.

RQ 1: What kinds of visualization techniques are mostly used in SEV?

In order to answer this question, we followed an existing classification of the visualization techniques presented by Shahin [2]. According to Shahin, visualization techniques are divided into five different categories (See Table 2 and Figure 2).

Table I. Types of Visualization Techniques and Studies they are reported in.

Type	Reported in
Graph-based	[7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17], [18]
Notation-based	[19]
Matrix-based	[7], [13], [15], [20], [21], [22], [23], [24], [25], [26]
Metaphor-based	[27], [28]
Others	[9], [29], [30], [31], [32]

Following is a brief discussion of the above mentioned visualization techniques:

1. Graph-based Visualization

Visualization using nodes and links to show the structural relationships of software's evolution. This type is most popular in SEV, and is reported in 12 studies as shown in *Table 2*. For example, in [7] graph-based representation is used to present the hierarchical interactions between models of the system as tree, where models are the central and arrows between models represent relationship. Different colors are used by different

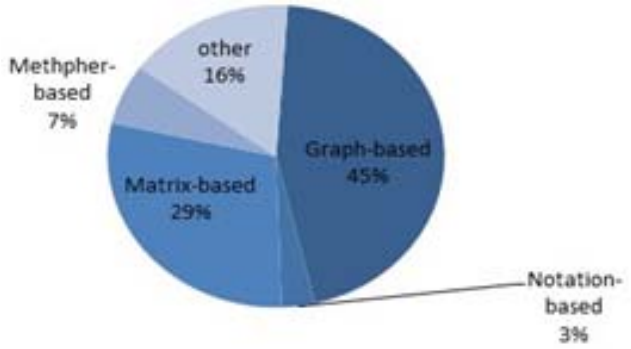

Figure 2. Types of Visualization Techniques as Appeared in the Literature.

types of implicit dependencies (e.g. *red* for inheritance and *black* for invocation). In [12] tree based visualization is used to represent the hierarchal representation of the system; four architecture models are used: *Entity-model, File-Level model, High-Level model,* and *Architectural-Level model.*

2. Notation-based Visualization

Visualization using different modeling techniques such as: *UML* (Unified Modeling Language), *SysML* (System Modeling Language), and other specific notation-based visualizations. Notation-based visualizations are rarely used, and reported in only 1 study ([19]). In this study McNair et al. use UML or E-R diagrams to show the impact of change-set on the system architect using architectural impact view, which uses colors in each entity/relationship (*e.g. green* represent *added*, *yellow* represent *modified*, and so on). This provides a comprehensive view of the system's architectural final state.

3. Matrix-based Visualization

Matrix-based is reported in 10 studies as shown in *Table 2*. Lanza et al. [7] use matrices to display detailed dependencies between two modules. In [21] they use evolution matrix-based on history to show different versions that have just important change, where lines in the matrix represent the class history, and columns represent the system's version.

4. Metaphor-based Visualization

This category used a more familiar visualization technique which is visualization as cities [33]. Metaphor-based is reported in two studies [27] and [28]. Wettel et al. [27] use a 3D city metaphor to represent the evolution of object-oriented software system; using their history to represent two levels of granularity:

Coarse-grained level: classes are represented as monolithic blocks, hidden internal details.

Fine-grained level: methods as cuboids (bricks), arranged on top of each other, based in the order of their creation (i.e. first bottom and recent top). Removed methods are represented as empty spaces. The height of the building increases as the number of methods increase. Provides an immediate access to the class for interaction by using three techniques applied in each granularity level: *Age Map, Time Travel,* and *Timeline*. Xie and Poshyvanyk [28] use metaphor-based visualization to

represent a set of granularity-based views of CVS data (e.g. *system level view, file (class), function (method), and Line (LOC) level view)* using color and height.

5. Others

This category is identified to follow up with the studies that do not use any of the above types, and these studies present different ways of visualization to represent software evolution. For example, McIntosh et al. [32] use musical interpretation to support exploration of software evolution data. They generate music using parameter-based sonification of historical changes, which are recorded in a version control system. Hanjali [30] performs visualization using a (mirrored) radial tree to show the file and scope structures, complemented with hierarchically bundled edges that show clone relations.

RQ 2: What is the main target of SEV?

McNair et al. [19] present a classification that divided the SEVs into four different categories based on their target *(See Table 3 and Figure 3).*

Table II. Major SEV Categories and Studies they are reported in.

Target	Reported in
Artifact-based	[9], [11], [14], [15], [18], [20], [23], [24], [25], [26], [27], [29], [34], [30], [35]
Metric-based	[21], [22], [28]
Feature-based	[10], [16], [31], [32], [36]
Architecture-based	[7], [8], [12], [13], [17], [19], [25]

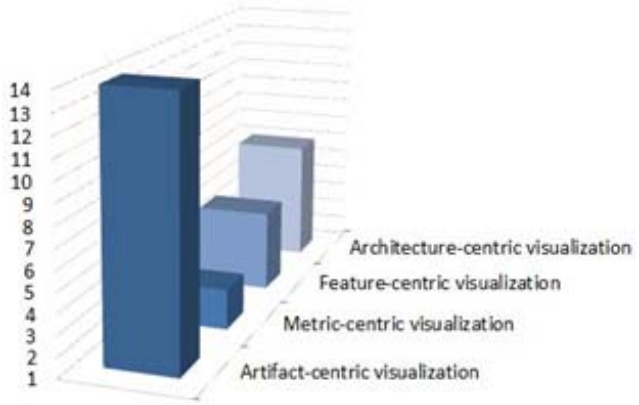

Figure 3. Major SEV Categories Based on their Focus and Target.

Following is a brief discussion of McNair et al. [19] categories:

1. Artifact-centric visualization

Visualizations in this category aim to provide a view of how the artifacts in the software repository change over time; especially the source code files and the line of code within the file. This type is reported in 15 studies as shown in Table III, which shows that *Artifact-centric* visualizations are used in most of the studies. Taylor et al. [29] introduce *Revision Tower* to visualize data extracted from the version control log file; the data include details about each file in the project (e.g. date, version number, and number of lines changed).

2. Metric-centric visualization

This type provides a view about how software metrics have change over time. Three studies reported this category ([21], [22], and [28]). Ducasse and Lanza define history measurement to summarize the change in the evolution of the object-oriented software systems.

3. Feature-centric visualization

In order to analyze how and which features changed over time; features such as comments in the log file, bug reports, or execution traces. This category is reported in five studies ([10], [16], [31], [32], and [36]). Greevy et al. use simple visualizations to show features as a grouping of participating structural entities.

4. Architecture-centric visualization

This category is used to provide a view of how the architecture of the software has been changed over time. Used in seven studies [7], [8], [12], [13], [17], [19], and [25]. YARN [8] provides an approach to display the architecture dependencies graph evolve over time and coloring schema.

RQ 3: What are the mostly used sources of information in SEV?

There are many sources of information used when needed to understand and visualize how systems evolve over time. The most used source of information are the software repositories (e.g. CVS, RCS), where data stored from the change history support the development process and maintenance in an efficient manner [28]. The source code is another source of information, from which needed information can be directly extracted. Some studies use both of them. Table 4 shows the source of information used in each of the relevant studies. Figure 4 explains the results.

Table III. Info-Source.

Info-Source	Studies ID
CVS	All studies
Source code	[10], [26]
Both of them	[8], [12], [19]

IV. CONCLUSIONS

A SLR is presented in this article in order to analyze the SEV tools. A number of studies have been selected systematically, and studied the used visualization techniques.

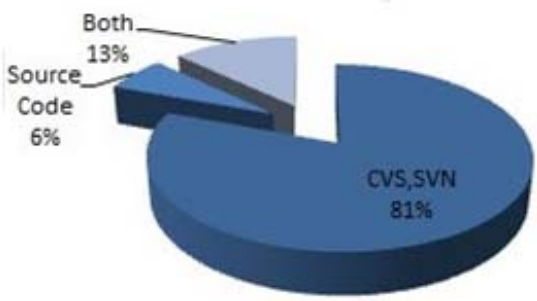

Figure 4. Info-Source.

The study aimed to (1) identify the techniques/methods used in visualizing software's evolution, and to find out what are the most and least used kinds, (2) identify the major targets and issues when applying SEV, and (3) identify the sources of information used, and which are most used and more reliable when extracting the required information.

The SLR process started with 55 papers; 29 out of the 55 are selected based on the inclusion and exclusion criteria for this review. The reported results of the SLR are as follow:

1) SEV techniques can be divided into five categories: *graph-based, notation-based, matrix-based*, *and metaphor-based* and *others*. Graph-based are most popular while Notation-based are the least.

2) The SEV tools and techniques focus can be either *Artifact-centric visualization, Metric-centric visualization, Feature-centric visualization,* or *Architecture-centric visualization*. It appears that both of the Artifact-centric and Architecture-centric are the most popular ones.

3) The sources of information used to extract information in SEV are mainly software repositories because they keep track of data in change history and what, when and how change effect on software evolution. Very few studies extract data directly from source code or both of them.

This work can help developer, maintainer, researcher to get good knowledge about the state of software evolution and visualization as a whole.

REFERENCES

[1] H. P. Breivold, I. Crnkovic, and M. Larsson, (2012). A systematic review of software architecture evolution research. Information and Software Technology,54(1), 16-40.

[2] M. Shahin, P. Liang, and M. A. Babar, (2014). A systematic review of software ar-chitecture visualization techniques. Journal of Systems and Software, 94, 161185.

[3] S. Keele, (2007). Guidelines for performing systematic literature reviews in software engineering. In Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

[4] D. Beyer and A. E. Hassan, (2006, October). Animated visualization of software his-tory using evolution storyboards. In Reverse Engineering, 2006. WCRE'06. 13th Working Conference on (pp. 199-210). IEEE.

[5] T. Grba, M. Lanza, and S. Ducasse, (2005, March). Characterizing the evolution of class hierarchies. In Software Maintenance and Reengineering, 2005. CSMR 2005. Ninth European Conference on (pp. 2-11). IEEE.

[6] R. Wettel, (2009, May). Visual exploration of largescale evolving software. InSoft-ware EngineeringCompanion Volume, 2009. ICSE-Companion 2009. 31st Interna-tional Conference on (pp. 391-394). IEEE.

[7] M. Lungu and M. Lanza. Exploring inter-module relationships in evolving software systems,(2007. In Proceedings of CSMR 2007 (11th European Conference on Software Maintenance and Reengineering), pages 91102, Los Alamitos CA, 2007. IEEE Computer Society Press.

[8] A. Hindle, Z. M. Jiang, W. Koleilat, M. Godfrey, and R. Holt. Yarn: Animating software evolution,(2007. In Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, 2007 (VISSOFT 2007), pages 129-136, 2007.

[9] D. German and A. Hindle. Visualizing the Evolution of Software Using SoftChange, (2006). International Journal of Software Engineering and Knowledge Engineering, vol. 16, no.1, pp. 521.

[10] O. Greevy, S. Ducasse and T. G,rba, (2006). Analyzing Software Evolution through Feature Views. Journal of Software Maintenance and Evolution: Research and Prac-tice, vol. 18, no. 6, pp. 425-456, 2006.

[11] D. Beyer and A. E. Hassan, (2006). Evolution storyboards: Visualization of soft-ware structure dynamics. In Proceedings of the International Conference on Pro-gram Comprehension (ICPC'06), pages 248-251. IEEE Computer Society.

[12] Q. Tu and M. W. Godfrey, (2002). An Integrated Approach for Studying Architec-tural Evolution. In Proceedings of the 10th International Workshop on Program Comprehension, p.127, June 27-29, 2002.

[13] S. Rufiange and G. Melanon, (2014, September). AniMatrix: A Matrix-Based Visu-alization of Software Evolution. In Proceedings of the 2nd IEEE Working Confer-ence on Software Visualization, Sep 29 - 30, 2014, Victoria, CA, pp. 137-146.

[14] H. Gall, M. Jazayeri, and J. Krajewski, (2003, September). CVS Release History Data for Detecting Logical Couplings. In Proceeding of International Workshop Principles of Software Evolution, pp. 13-23.

[15] A. Vanya, R. Premraj, and H. V. Vliet, (2010). Interactive Exploration of Coevolv-ing Software Entities. In Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR 2010), pp. 260-263.

[16] M. Fisher and H. Gall, (2003, September). MDSViews: Visualizing Problem Report Data of Large Scale Software using Multidimensional Scaling. In Proceedings of the International Workshop on Evolution of LargeScale Industrial Software Applica-tions (ELISA 2010), pp. 110-121.

[17] M. Burch, C. Vehlow, F. Beck, S. Diehl, S., and D. Weiskopf, (2011). Parallel edge splatting for scalable dynamic graph visualization. Visualization and Computer Graphics, IEEE Transactions on, 17(12), 2344-2353.

[18] T. Chaikalis, G. Melas, and A. Chatzigeorgiou, (2012, September). SEANets: Soft-ware evolution analysis with networks. In 2012 28th IEEE International Conference on Software Maintenance (ICSM) (pp. 634-637). IEEE.

[19] A. McNair, D.M. German, J. Weber-Jahnke,(2007). Visualizing software architec-ture evolution using change-sets. In Proceedings of 14th Working Conference on Reverse Engineering, 2007, pp. 140149.

[20] C. Mesnage and M. Lanza, (2005). White Coats: Webvisualization of evolving software in 3D. VISSOFT 2005. In Proceedings of the 3rd IEEE International Workshop on Visualizing Software for Understanding and Analysis, 0:40-45, 2005.

[21] T. Girba, S. Ducasse, and M. Lanza, (2004). Yesterday's weather: Guiding early re-verse engineering efforts by summarizing the evolution of changes. In Proceedings of the International Conference on Software Maintenance, pages 40-49. IEEE Computer Society Press.

[22] J. Wu, R. Holt, and A. Hassan, (2004). Exploring software evolution using spectro-graphs. In Proceedings of the 11th Working Conference on Reverse Engineering (WCRE 2004), pages 80-89. IEEE Computer Society Press.

[23] M. Burch, S. Diehl, and P. Weigerber, (2005, September). EPOSee-A Tool For Visualizing Software Evolution. In Proceedings of the 3rd IEEE International Workshop on

Visualizing Software for Understanding and Analysis (VISSOFT 2005), Budapest, Hungary. Sept. 25, 2005, pp. 1-2.

[24] G. Langelier, H. Sahraoui, and P. Poulin, (2008, September). Exploring the evolution of software quality with animated visualization. In Proceedings of the 2008 IEEE Symposium on Visual Languages and HumanCentric Computing (VL/HCC 2008), pp. 13-20.

[25] A. Gonzlez Torres, F. J. Garca Pealvo, and R. Thern Snchez, (2013). How Evolu-tionary Visual Software Analytics Supports Knowledge Discovery. Journal of In-formation Science and Engineering (JISE), 29(1): pp. 17-34.

[26] Z. O. Repository, Multi-Touch Collaboration for Software Exploration. In Proceed-ings of the 24th Aust. Comput. Interact. Conf. Pages 30-33, ACM, no. July, 2010.

[27] R. Wettel and M. Lanza, (2008). Visual Exploration of Large-Scale System Evolu-tion. In Proceedings of the 2008 15th Working Conference on Reverse Engineering, p.219-228, October 15-18, 2008.

[28] X. Xie, D. Poshyvanyk and A. Marcus, (2006). Visualization of CVS Reposi-tory Information. In Proceedings of the 13th Working Conference on Reverse En-gineering, pages 231-242, 2006.

[29] C. Taylor and M. Munro, (2002). Revision towers. In Proceedings of the 1st Inter-national Workshop on Visualizing Software for Understanding and Analysis, p.43-, June 26-26, 2002.

[30] A. Hanjalic,(2013, September). ClonEvol: Visualizing Software Evolution with Code Clones. In Proceedings of the 1st IEEE Working Conference on Software Visualization, Sep 29 - 30, 2014, Eindhoven, NL, pp. 1-4.

[31] J. Zhi and G. Ruhe, (2013, September). DEVis: A Tool for Visualizing Software Document Evolution. In Proceedings of the 1st IEEE Working Conference on Software Visualization, Sep 29 - 30, 2014, Eindhoven, NL, pp. 1-4.

[32] S. McIntosh, K. Legere, and A. E. Hassan, (2014, February). Orchestrating change: An artistic representation of software evolution. In Software Maintenance, Reengi-neering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on (pp. 348-352). IEEE.

[33] R. Wettel and M. Lanza, (2007, June). Visualizing software systems as cities. In-Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on (pp. 92-99). IEEE.

[34] B. Ens, D. Rea, R. Shpaner, H. Hemmati, J. E. Young, and P. Irani, (2014, Septem-ber). ChronoTwigger: A Visual Analytics Tool for Understanding Source and Test Co-evolution. In Proceedings of the 2nd IEEE Working Conference on Software Visualization, Sep 29 30, 2014, Victoria, CA, pp. 117-126.

[35] M. D'Ambros, M. Lanza, and M. Lungu, (2006, May). The evolution radar: Visual-izing integrated logical coupling information. In Proceedings of the 2006 interna-tional workshop on Mining software repositories (pp. 26-32). ACM.

[36] M. D'Ambros and M. Lanza, (2007, March). Bugcrawler: Visualizing Evolving Software Systems. In Proceedings of the 11th European Conference on Software Maintenance and Reengineering (CSMR '07), IEEE Computer Society, March 2007, pp. 333-334.