Software Evolution Visualization Tools Functional Requirements: a Comprehensive Understanding

Hani Bani-Salameh
Department of Software Engineering
The Hashemite University, Jordan
Email: hani@hu.edu.jo

Ayat Ahmad
Department of Software Engineering
The Hashemite University, Jordan
Email: ayat_ahmad1991@yahoo.com

Dua'a Bani-Salameh
Department of Computer Science
JUST, Jordan
Email: duaabs@gmail.com

***Abstract*—Software is usually going under many changes during its life time cycle. Following up software changes and enhancements is an essential process for many reasons: it increases the complexity of software projects, and affects the software structure and quality. Software visualization is considered as one of the comprehensive techniques that developers make use of daily in order to analyze and understand how the software evolves and changes over time. To achieve this, developers use software evolution visualization (SEV) tools, and face difficulties finding/identifying the most suitable tool. The goal of this study is to identify generic functional requirements for software visualization tools, in order to help developers choose their tools based on the supported features/requirements. The main focus is on tools that target softwares' evolution. The research methodology is based on a systematic review that aims to summarize the current research on software SEVs and to answer a question on "what are the main functional requirements for software evolution visualization tools that have been identified in the literature?" The most common functional requirements and activities that have been identified in this study are views, detailed-on-demand, filter, select, re-arrange, and comparison.**

Keywords–Evolution; Software Evolution Visualization; Tools.

I. INTRODUCTION

Software evolution is an essential topic in software engineering and maintenance [1]. Software continually changes in response to any changes in (1) the requirement, (2) environment to adapt new technology, and (3) to repair errors. With increasing the size and complexity of the software systems, following up the continual software changes is not an easy task. Software engineers need to understand how software evolves over time to keep the software operational with a high quality. Also, this is necessary to present its state and to predict its future development.

Software visualization is a powerful technique for software comprehension by mapping different software aspects with visual properties such as position, size, shape, and color [2][3]. It is shown that it can be effective to understand and analyze software evolution. There are many SEV tools. Such tools aim to visualize the version history of software systems by visualizing history of the software artifact such as - source code changes, test files, and documentation. No matter what is the source of visualization or in what level (source code, documentation, or other) it is implemented . These tools have common requirements and functionalities, and have been studied to extract a set of common functional requirements for SEV tools.

Requirements identification is the base and the start point for building any software project [4]. Also, it is important for the construction of SEV tools. Each tool uses a different technique to represent different types of changes, have particular advantages, and is facing different problems and challenges.

SEV tools usually deal with and process a huge amount of data that is needed to visualize in order to follow software changes and the impact of these changes [5]. A good user interface design for SEV tools involves multiple user interaction techniques which give users the ability to interact with and represent data such as multiple views, select, filter and other features, that can help to overcome challenges and increase user's satisfaction. Also, interaction techniques can affect the quality of SEV tools by allowing higher flexibility and extensibility [6]. Studying such tools allows us to specify more important features and functional requirements that increase the users' satisfaction, and humans' understanding for software evolution in specific.

The main goal of this study is to identify a list of generic functional requirements for SEV tools. Defining requirements for such tools can be used as an indicator to ensure if these tools meet the quality attributes, enhance human perception, and meet user's expectations. Also, this study presents the current state of such tools which represents the first step when building a new SEV tool, and in defining the requirements for the future.

The remaining part of this paper is organized as follow. Section II describes the research approach that have been followed when conducting the review. Section III discusses the functional requirements uncovered by this study. Section IV provides a brief discussion. Finally, Section V presents the future work and concludes the paper.

II. METHOD

To identify the functional requirements for the SEV tools, we started our study by identifying a research question as follows.

"What are the main functional requirements for the software evolution visualization tools that have been identified in the literature?"

 ISBN: 978-1-61208-498-5

Then, we started to collect papers that have clear demonstration for the used features and interaction techniques. The total of all surveyed papers is 39.

A list of all interaction techniques is introduced, then we started to analyze the papers and summarize most common set of techniques used.

Next, similar features are grouped (categorized) based on what tasks they perform, and what are the goals from those operations. Different interactions/features techniques are used to perform similar tasks and similar goals.

During the analysis of the SEV tools, the focus was on *what important features that help developers to better understand the software evolution process*. For this reason, we focused on what users achieve by using such operations rather than how.

III. FUNCTIONAL REQUIREMENTS

Following is a list of six functional requirements that have been found common in most SEV tools (see Table I):

- **Views:** show different representation.
- **Details-on demand:** show more information/details.
- **Filter and Search:** show something conditionally.
- **Select:** marks something as point of interest.
- **Re-arrangement:** shows different arrangements of the data sets (e.g. sort, cluster, and aggregate).
- **Comparison:** shows the differences between different data sets.

TABLE I. MOST COMMON FUNCTIONAL REQUIREMENTS.

Method/Functionality	Ways to Visualize
Views	• Different Granularity Levels • Increase Abstraction Level • Dependencies & Relationships • Others
Details-on-Demand	• Zoom • Expand • Mouse Cursor Hovers
Filter & Search	• Uses checkbox • Change color, and change representation
Select	Select and highlight data sets.
Re-arrange	Analyze changes in different ways such as aggregate, cluster and sort.
Comparison	Comparing changes that occur from time to time, and between different software versions.

A. Views: to show different representation

SEV tools provide different visual representations of the target software history (e.g. different colors, size, and shapes with animation). Multiple views is an effective technique that helps software engineers to understand how different parts of the software evolved and changed in different perspectives. It provides flexibility to analyst to directly find views that fit their goals.

Multiple views are used in SEV tools to show how changes occur in different ways as the following:

- **In different granularity levels of the system:** an example of multiple views is provided by Xie et al. who define multiple views of the data history changes at different granularity levels (e.g. system level, file(class) level, method level, and line level) by supporting navigation within views [7].
- **To increase abstraction level:** the DEVis [8] tool uses multiple windows/views to increase the abstraction level. When a part or icon is selected a more detailed window appears.
- **Clear insight about dependencies and relationships:** using different representations which provide the user with a clear insight about dependencies and relationships between software artifacts, and to identify which parts of the project are changed together (see ChronoTwigger [9]).
- **Different types of visualization (chart with histogram):** multiple views can be used to represent how data changed (e.g. charts with histogram), in order to give insight into evolutionary trends and phenomena governing software growth (see SEANet tools [10], AniMatrix [11]). They help to give users with the ability to navigate/travel/fly within views by selecting or clicking a specific node.

B. Details-on-Demand: Show More Information and Details.

Most SEV tools provide users with the ability to get more details when needed about the presented data such as: number of revisions in a specific date, type of changes occurred, and number of nodes in a specific version. User interaction techniques used to get more details are (1)*Zooming* (Zoom-in, Zoom-out), (2)*Expand*, and (3)*Mouse Cursor Hovers* over a data item. Using zoom-out users can change scale of data overview for large data sets, or decrease it using zoom-in to get detailed view for small data sets [12]. Examples such as (AniMatri [11], SEANet [10], and DEVis [8]).

C. Filter and Search: Show Something Conditionally.

Filter used in SEV tools which allows users to control the displayed data based on a specific condition (query), where users specify conditions or ranges (e.g. based on type, range of specific duration time, or another criteria). Only the data that satisfy the condition is presented and the other data is hidden from the view or shown in a different way or different color. ClonEvol [13] uses checkbox to filter by node's type (attribute, class, and file), and by type of operation (add, delete, modified). ChronoTwigger [9] allows to show only the node(file) that have co-change/changed within selected timespan, and hides nodes that do not have any change event. Filter is used in AniMatrix [11], that allows users to control data presented in history navigator, is using filter by interface name or by type of coupling in order to show classes that have been changed over time, and to show different types of coupling. From the above, it appears that user interaction techniques implemented in most of the SEV tools are: *checkbox, change color, and change representation.*

D. Select: Mark Something as Point of Interest.

SEV tools which provide users with the ability to select data sets and highlight them, to facilitate, follow up, and track how large data changed and evolved over time, and to identify

locations of more important data that have co-change or most impact change. ChronoTwigger [9] allows users to select a node or set of nodes on the 2D visualization to highlight the set of corresponding/related nodes in 3D view.

E. Re-arrangement: Show Different Arrangement of Data Sets.

SEV tools which provide users with the ability to change the way of representation of software entities in new arrangements, and group it in different perspectives. Re-arrangement allow developers to analyze changes in different ways such as *aggregate, cluster, and sort*. Storyline tool allows users to aggregate developer in one large clusters, and connects everyone to everyone else [14]. iVIS [15] allows to re-arrange the entity layout of the visualizations by selecting software entities and dragging them around. This allows the developers to analyze co-changes between selected entities in different groups, and can give more attentions for a specific entity by the developer.

F. Comparison: Show the Differences Between Different Data Sets.

Comparing changes that occur from time to time, and between different software versions is an essential requirement in the SEV tools, that helps developers to analyze the differences between software versions and to identify the state of the system. Some tools provide comprehensive techniques in the single view using color and animation.

Comparison mode used in eCITY tool [16] which allows to compare between two different dates by using animation, and sequentially insert the new components. This allows users to compare the states of specific dates side by side. Salamanca et al. [17] present a tools that provides a software structures view, and allows a side-by-side comparison of the evolving package or class hierarchy structures from the selected project revisions in circular timeline. Other examples appear in [18] and [19].

G. Other Requirements

Following is a list of requirements mentioned by researchers but not commonly used.

- Source code presentation: which gives users the ability to access the underling source code [11][17][21].
- Panning [11]22], Brushing, Collapse [22][23], manipulate [9][22], move up and down [8][18][24].

The SEV tools can be divided into two categories. The categorization is based on the (1)**type** of visualization that is used (at which level such as Artifacts and Structure Evolution), and (2)**method** to represent the techniques used to visualize the software evolution (See Figure 1). Figure 1 shows that the visualization tools are categorized either based on the type of visualization they support (eg. visualizing software artifacts and structure) or the methods of visualization (eg. views, filters, comparison, etc.).

This study does not pay attention to what are the used types of SEV tools. The main focus is on getting generic functional requirement to guide all users either researchers, developers, or maintainers to better understand software's evolution process.

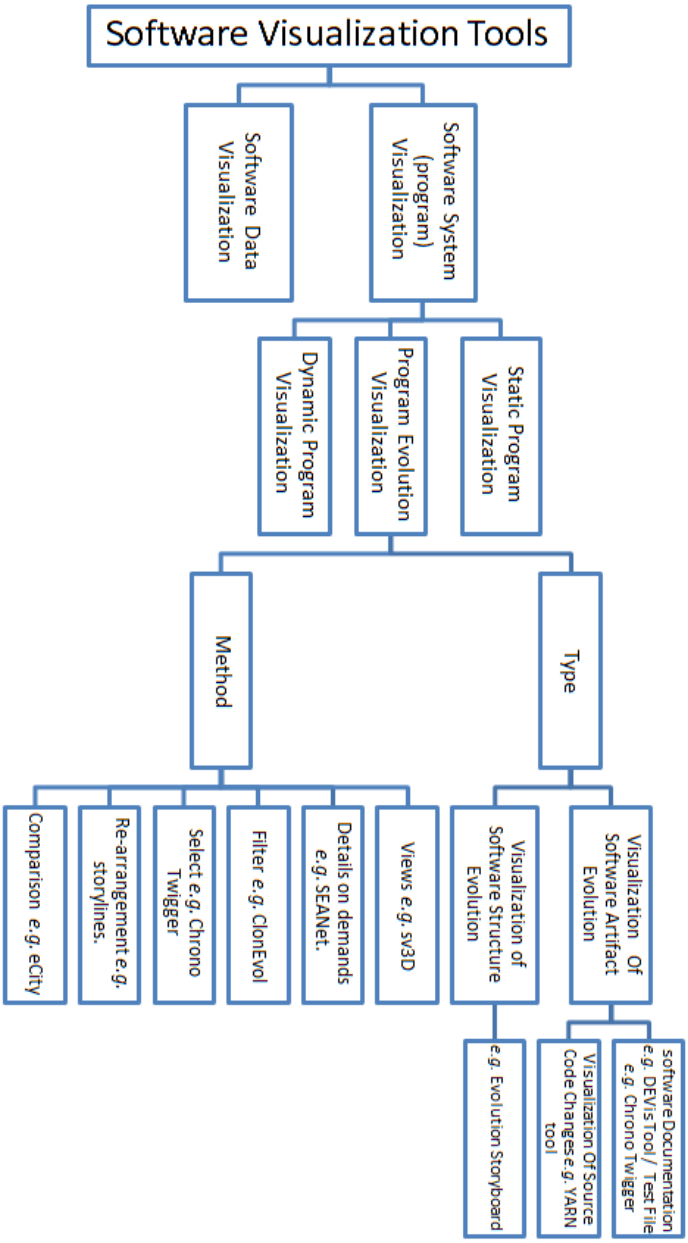

Figure 1. Classification of SEV Tools.

IV. DISCUSSION

It is difficult to find all interaction technique used in the SEV tools. The proposed functional requirements for the SEV tools are defined based on features supported in most of the existing tools. The supported features can help users to better understand software changes and evolution. This section discusses more issues about the proposed functional requirements.

Interactive facilities provided to the user are essential to the quality of the SEV tools. They support higher flexibility and extensibility [6], where users not only can see information but also can interact with it to reduce the effort needed for better understanding, and supports the ability to faster change context of information as needed in powerful ways.

Different visual attributes such as color, animation and

different shape "way of representation" are used by SEV tools requirements to enrich user interface and increase user satisfaction. Also, animation is used to enable to analyze large information in a usable way. It allows to compare instances sequentially, show dependencies between software artifacts. Also, it illustrates dynamic behaviors for software evolution.

Some interaction techniques are used in the SEV tools to fulfill the user's multiple intents, and it is possible to fit more than one requirement/feature in the tool. For example, comparison features are intended to show differences between specific data sets; filters feature helps users to compare their data sets, the re-arrange feature allows to compare two groups, and the multiple views feature allows users to compare between different views of the software. Also, multiple views are used to increase the abstraction level in order to give users the ability to get more details as needed.

We believe that the proposed generic functional requirements for SEV tools is useful and has several advantages. First, they can be used as a first step when building new SEV tools, and in order to examine whether they meet users' needs. Second, they can be used to understand what are the user's needs, and help to find new techniques that might help to better understand the software evolution process.

V. Conclusion and Future Work

This research project identifies generic functional requirements for the software evolution visualization tool by studying the existing literature. The main goal for identified requirement for software evolution visualization tool is to find more essential requirements that help all stockholder to better understand the software evolution process. The six reported functional requirements are: *views*, which provide multiple representations for changes in different granularity level, different version, for different type of artifact; *detailed-on demand* requirements provide the user with more information than needed such as number of revisions, number of nodes in a specific version, and so on, by mouse hover and zoom-in, zoom-out operation; *filter and search* which has the ability to change data presented based on specific condition(e.g. a developer that wants to analyze changes that occur in a specific time stamp and hide the other changes). *Select* requirements give the user the ability to mark specific things as needed. *Re-arrange* feature is considered as a good way to group presented data in different perspectives by sorting or clustering. Finally, *Comparison* is the last suggested requirement in this study. It gives the user the ability to make comparisons, and allows them to notice differences between software versions for different types of entities in different level.

Further study can be performed in the future in order to expand our list of functional requirements. Also, the study can be extended to gather feedback from developers, and domain experts which helps to identify a list of non-functional requirements for the SEV tools. This can help developers and designers to build a generic software quality model for such tools.

References

[1] R. Lima, A. Torres, T. Souto, M. Mendona, and N. Zazworka, "Software evolution visualization: A systematic mapping study", Information Software Technology, vol. 55, no. 11, pp. 1860–1883, 2013.

[2] D. Gracanin, K. Matkovic, and M. Eltoweissy, "Software visualization," Innovations in Systems and Software Engineering, vol. 1, no. 2, pp. 221-230, 2005.

[3] J. Heer, B. Shneiderman, and C. Park, "Interactive Dynamics for Visual Analysis: A Taxonomy of Tools that Support the Fluent and Flexible Use of Visualizations," Queue - Micoprocessors, vol. 10, no. 2, pp. 30-55, 2012.

[4] H. M. Kienle and H. A. Muiller, "Requirements of Software Visualization Tools: A Literature Survey," In Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), Canada, 2007, pp. 2–9.

[5] S.-L. Voinea. "Software Evolution Visualization," PhD thesis, Technische Universiteit Eindhoven, 2007.

[6] M. Lanza, "CodeCrawler-Lessons Learned in Building a Software Visualization Tool," In Proceedings of the European Conference on Software Maintenance and Reengineering, pp. 409–418, 2003, IEEE Press.

[7] X. Xie, D. Poshyvanyk, and A. Marcus, "Visualization of CVS Repository Information," In Proceedings of the 13th Working Conference Reverse Engineering., 2006, pp. 231-242.

[8] J. Zhi and G. Ruhe, "DEVis: A Tool for Visualizing Software Document Evolution," In Proceedings of the First IEEE Working Conference on Software Visualization, Eindhoven, Netherlands, 2013, pp. 1-4.

[9] B. Ens, D. Rea, R. Shpaner, H. Hemmati, J. E. Young, and P. Irani, "ChronoTwigger: A Visual Analytics Tool for Understanding Source and Test Co-Evolution," In Proceedings of the 2nd IEEE Working Conference on Software Visualization (VISSOFT). Victoria, Canada. 2014, pp. 117–126.

[10] T. Chaikalis, G. Melas, and A. Chatzigeorgiou, "SEANets: Software Evolution Analysis with Networks," In Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 2012, pp. 634-637.

[11] G. Melanc, "AniMatrix: A Matrix-Based Visualization of Software Evolution," In Proceedings of the Second IEEE Working Conference on Software Visualization (VISSOFT), 2014, pp. 1–10.

[12] J. S. Yi, Y. Kang, J. T. Stasko, and J. A. Jacko, "Toward a Deeper Understanding of the Role of Interaction in Information Visualization," IEEE Transactions on Visualization and Computer Graphics (TVCG), vol. 13, no. 6. Presented in InfoVis 2007, Sacramento, California, October 28 - November 1, pp. 1224–1231.

[13] A. Hanjali, "ClonEvol: Visualizing Software Evolution with Code Clones," In Proceedings of the First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pp. 1-4.

[14] M. Ogawa, "Software Evolution Storylines," 2010, URL:http://www.michaelogawa.com/research/storylines/, [accessed: 13–07–2016].

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

[16] T. Khan, H. Barthel, A. Ebert, and P. Liggesmeyer, "eCITY: A Tool to Track Software Structural Changes using an Evolving City," In Proceedings of the 29th IEEE International Conference on Software Maintenance (ICSM), 2013, pp. 492–495.

[17] A. Gonzalez, R. Theron, A. Telea, and F. J. Garcia, "Combined Visualization of Structural and Metric Information for Software Evolution Analysis," In Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, ACM, 2009, pp. 25-30.

[18] O. Benomar and P. Poulin, "Visualizing Software Dynamicities with Heat Maps," In Proceedings of the First IEEE Working Conference on Software Visualization (VISSOFT), 2013, pp. 1-10.

[19] Q. Tu and M. W. Godfrey, "An integrated approach for studying architectural evolution," In Proceedings of the 10th International Workshop on Program Comprehension, 2002, pp. 127-136

[20] M. Burch, C. Vehlow, F. Beck, S. Diehl, D. Weiskopf, and I. C. Society, "Parallel Edge Splatting for Scalable Dynamic Graph Visualization, IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 12, 2011, pp. 2344–2353.

 ISBN: 978-1-61208-498-5

[21] M. Lungu and M. Lanza, “Exploring Inter-Module Relationships in Evolving Software Systems,” In Proceedings of the 11th European Conference on Software Maintenance and Reengineering, Amsterdam, 2007, pp. 91–102.

[22] S. Boccuzzo and H. C. Gall, “Multi-Touch Collaboration for Software Exploration,” International Conference on Program Comprehension, Portugal, 2010, pp. 30–33.

[23] M. D. Ambros and M. Lanza, “A Flexible Framework to Support Collaborative Software Evolution Analysis,” In Proceedings of the 12th IEEE European Conference on Software Maintenance and Reengineering, 2008, IEEE CS Press, pp. 3-12.

[24] M. D. Ambros and M. Lanza, “BugCrawler: Visualizing Evolving Software Systems,” In Proceedings of the 11th European Conference on Software Maintenance and Reengineeringpp, Amsterdam, 2007, pp. 333–334.

 ISBN: 978-1-61208-498-5