Visual analysis of software systems in virtual and augmented reality

Conference Paper · September 2015

DOI: 10.1109/INES.2015.7329727

4 authors, including:

Peter Kapec
Slovak University of Technology in Bratislava
13 PUBLICATIONS **55** CITATIONS

SEE PROFILE

Visual Analysis of Software Systems in Virtual and Augmented Reality

P. Kapec*, G. Brndiarová*, M. Gloger* and J. Marák*
*Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava, Slovakia
peter.kapec@stuba.sk, gbrndiarova@gmail.com, gloger.michael@gmail.com, xmarak@is.stuba.sk

***Abstract*—Software visualization is an ongoing research area with the goal to make software systems more tangible. In this paper we present our visualization system that allows visual analysis of graph structures representing software systems in both virtual and augmented reality. The provided visual analysis process is demonstrated on two existing software systems. We propose the transition towards visualization in augmented reality and we present developed interaction techniques to explore and examine software visualizations in augmented reality.**

I. Introduction

Today information visualization [1] is often done in various forms ranging from simple 2D visualizations, through desktop 3D virtual reality applications to fully immersive cave automatic virtual environment (CAVE) systems. Augmented reality [2] is often used in e.g. entertainment, in medical and industrial applications, in military systems etc. Information visualization using augmented reality systems is currently rare and combining these two approaches could bring innovative solutions. With the advances in software engineering the software systems developed today are becoming more complex and difficult to understand. Information and software visualization aim to help with software understanding. By providing visual representations of software developers can discuss about visual representations rather than relying solely on vague mental models. The software development industry often uses UML diagrams to model software systems. The various UML diagrams are in fact some form of software visualization. The software visualization research field, a sub-field of information visualization, has developed many visualization systems and techniques in recent years and some solutions slowly become part of daily practice.

In this paper we present our approach for software visualization in virtual and augmented reality. Moving visualization from virtual to augmented reality offers new possibilities for presentation, but at the same time brings new challenges for interaction. The next section provides overview of related work – we introduce the software visualization field and mention related works that deal with information and software visualization in augmented reality. Section III describes in detail the developed visual analysis process, which is demonstrated by analysis of two existing software systems. Section IV describes our setup for visualization in augmented reality and discusses details about provided interaction methods followed by an evaluation with participants. The final section concludes our paper and mentions possible future work.

II. Related Work

A. Software visualization

According to [1] software is a specific type of data that can be visualized. Often software visualization methods and systems utilize visualization techniques developed for other data types – software is often visualized by using trees and graphs, abstract geometrical shapes or e.g. using city and landscape metaphors [3]. Software visualization can focus on three main aspects of software [4]: the visualization of *software structure*, the visualization of *program behavior* and the visualization of *software evolution*. A detailed survey about visualization of static aspects of software can be found in [5] and for recent surveys on program behavior visualization see [6], [7]. The survey [8] summarizes visualization of software evolution during development. Software visualization can be helpful with exchanging knowledge in software companies [9].

The visual analytics framework [10] incorporates interactive information visualization into a global approach that combines visualization with automated reasoning, data mining and data management methods, with knowledge about human perception and cognition and aspects of human-computer interaction. For a recent survey on visual analytics see [11].

B. Augmented reality

The authors of SkyscrapAR [12] used augmented reality to visualize software evolution using a metaphor of an evolving city. The packages are visualized as a rectangular city lots with the buildings on top of it, representing classes, where the area that a building occupies represents the size of a particular class measured in lines of code. Users are allowed to browse over software revisions and see changes in the city appearance, which reflects successive modifications that the classes suffered over time. SkyscrapAR uses marker tracking, which authors also use for interaction with the visualized data. Our work uses graph-based visualization techniques, cloud-point data for augmenting the reality and interaction using hand gestures and partly also markers.

Belcher et al. [13] studied the effects of complex graph visualization in 2D a 3D environments. The authors note that the major difference between visualization on a 2D display and using 3D augmented reality is the opportunity to rotate the model, which leads to better graph understanding. Meiguins et al. developed a prototype to visualize multidimensional information using a 3D scatter plot in augmented reality [14].

Accepted by IEEE for publication. The final IEEE published publication is DOI: 10.1109/INES.2015.7329727

The prototype uses markers for manipulating the 3D scatter plot and provides interaction techniques like filtering and details on demand using a virtual device to select virtual objects. Similar to our approach it focuses on interaction using other means than keyboard and mouse.

III. VISUAL ANALYSIS OF SOFTWARE SYSTEMS

A. Visual analysis process

The visualization process transforms data in one representation to a graphical representation and includes the data preparation and transformation step, the visual mapping step and finally presentation and interaction. These steps have been formalized into various visualization models and frameworks.

The most cited approach is Shneiderman's information seeking mantra: *"Overview first, zoom/filter, details on demand"* [15]. Keim et al. extended this mantra with analysis steps: *"Analyze first, show the important, zoom/filter, analyze further, details on demands"* [16] indicating that only showing data often does not provide significant insight into data. This led to the development of the Visual Analytics framework [10] that integrates visualization with automatic analysis methods.

In our approach we utilize ideas of these models and frameworks to provide visual analysis of software structure as described in following sections. We analyzed two existing software systems implemented in the Lua [17] programming language. From these systems we extracted the module structure and call graph information, which were visualized as graphs. For each function in the call graph we obtained detailed software metrics like lines of code (LOC), McCabe's cyclomatic complexity [18], function points [19], Halstead metrics [20], information flow metrics [21] etc. This software metrics were used to encode visual attributes of the call graph and were used for visual analysis.

B. Exploring graph visualizations

The first software system, what we visually analyzed, was the LuaDist[1] project, which contains 370 functions in 10 source code files. From this project we extracted a call graph containing 418 nodes and 1187 edges. The number of extracted nodes is larger due to extraction of: internal functions provided by the standard Lua libraries and external dependencies, and splitting source code into modules.

The full extracted graph is shown in Fig. 1 using visualization in a virtual environment with terrain and sky in the background. The graph visualization is done in 3D space and the graph layout is achieved using Fruchterman-Reingold [22] force-directed graph layout algorithm. The user can freely navigate in this visualization using a virtual camera that allows fly-mode and orbit-mode camera movements.

The second software system we visually analyzed was the luajson[2] library. It contains 109 functions divided into 18 source code files. After function calls extraction we extracted total of 194 nodes and 360 edges. The full extracted graph is shown in Fig. 2 using the same visualization method as in the previous example.

[1]LuaDist – www.luadist.org

[2]luajson – www.github.com/harningt/luajson

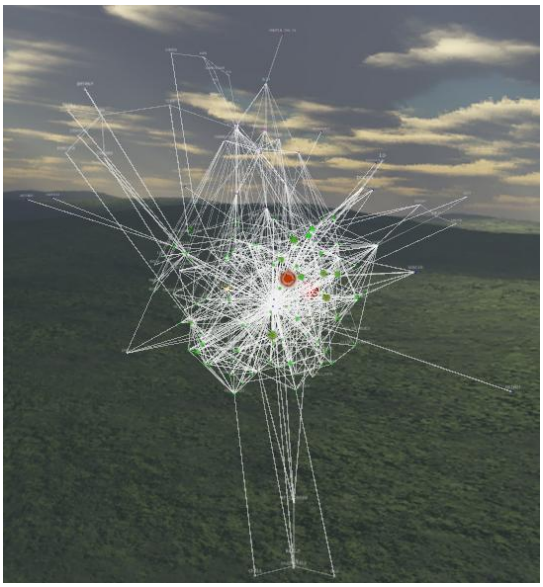

Fig. 1. The full extracted graph for the LuaDist project containing 418 nodes and 1187 edges.

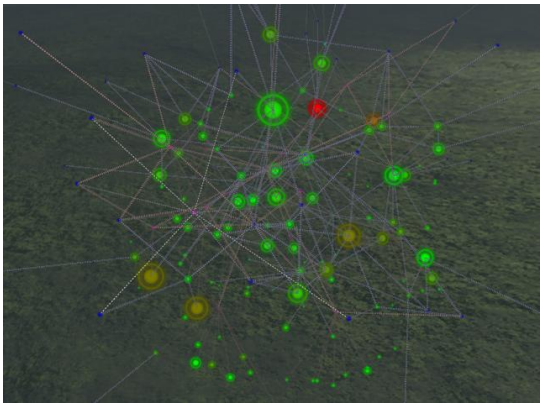

Fig. 2. The full extracted graph for the luajson library containing 194 nodes and 360 edges.

C. Filtering

As can be seen from Fig. 1 and 2, the extracted call graphs are quite complex and difficult to comprehend. Therefore we implemented a simple language to filter graph nodes and edges based on their type and stored information and software metrics. The Fig. 3 shows the call graph from Fig. 1 containing only the functions implementing LuaDist's functionality. This graph was obtained using the following node filter: **params.type like 'function'**, which filtered out all nodes whose type was not *function*.

In Fig. 3 we can see two nodes that are bigger and have red color. We used the software metrics LOC and McCabe's cyclomatic complexity from call graph functions to visually map to node size and color. Using this visual mapping we can easily identify in Fig. 3 the most complex functions of the

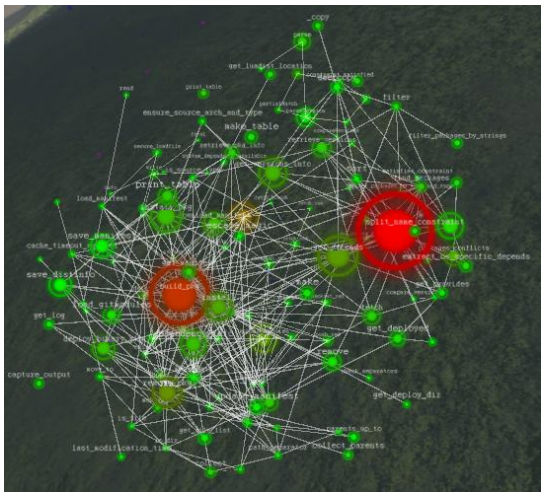

Fig. 3. The extracted call graph for the LuaDist project showing two nodes representing functions with higher complexity.

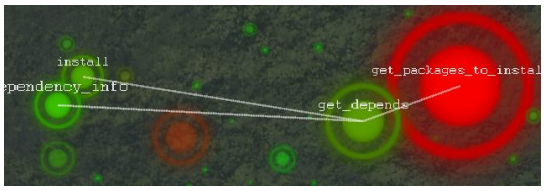

Fig. 4. The three functions that use the most complex function get_packages_to_install in LuaDist.

LuaDist project, which are the *get_packages_to_install* and *build_pkg* functions. Using the filter:

id == get_packages_to_install or
>id == get_packages_to_install or
>>id == get_packages_to_install

we are able to obtain the functions that call the function *get_packages_to_install*. In this filter the > and >> symbols denote nodes with **id**'s connected directly and indirectly to the *get_packages_to_install* node. The Fig. 4 shows that the *get_packages_to_install* function is directly used by one function and is indirectly used by 2 functions.

We used a similar approach to filter out information in our second analyzed software luajson. We decided to show only functions with high number of LOC and high cyclomatic complexity using the filter:

params.type like 'function' and
(params.metrics.LOC.lines >25 or
params.metrics.cyclomatic.upperBound >5)

As can be seen in results shown in Fig. 5 there are multiple functions needing our attention. For instance we are able to determine, that **isArray** is the most complex function in this software and the **generateDecoder** has most lines of code.

D. Visual clutter reduction using edge bundling

Visual clutter is often a problem in graph visualizations due to the big amount of nodes and edges. Edge bundling [23] is a known method for edge clutter reduction that improves readability of 2D node-link graph visualizations. Edges are bended and bundled together according to specified rules using various approaches for edge bundling, e.g. hierarchical, geometry-based, force-directed, etc. Edge bundling is often used in maps with integrated graphs with predefined layout.

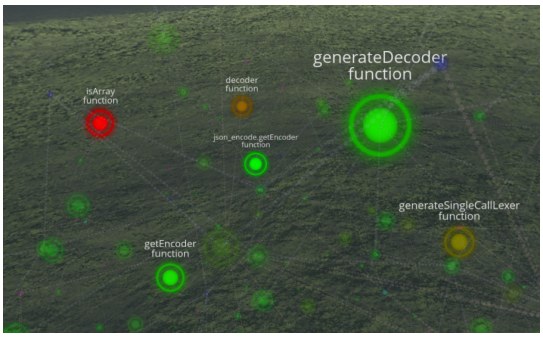

Fig. 5. Filtered graph of the luajson library containing functions with the most lines of code and the higher complexity.

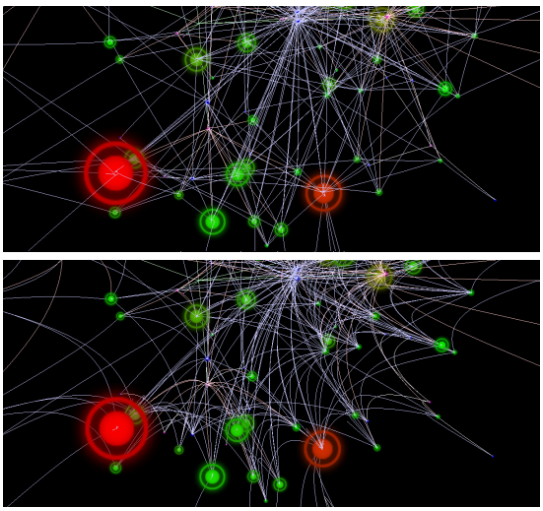

Fig. 6. Part of LuaDist graph. On the top is the original graph and on the bottom the graph after edge bundling.

We tackled this problem by using force-directed edge bundling. We applied the edge bundling to graphs displayed in 3D space, instead of previous solutions for graphs displayed in 2D space. The initial positions of nodes are calculated by the Fruchterman-Reingold [22] force directed algorithm. Then, the node positions are fixed and edge bundling is applied. During edge bundling, every edge is split by helper-nodes and every edge is replaced by a Bézier curve using the helper-nodes as control points for the curve. The helper-nodes are attracted to each other according to their indexes. In addition to attractive forces between helper-nodes, compatibilities between edges are calculated to improve results of edge bundling. We implemented angle, position and scale compatibilities inspired by [23]. The Fig. 6 shows part of the call graph in Fig. 3 before and after edge bundling. Fewer edges are crossing the red and green nodes in the graph with bundled edges. We can

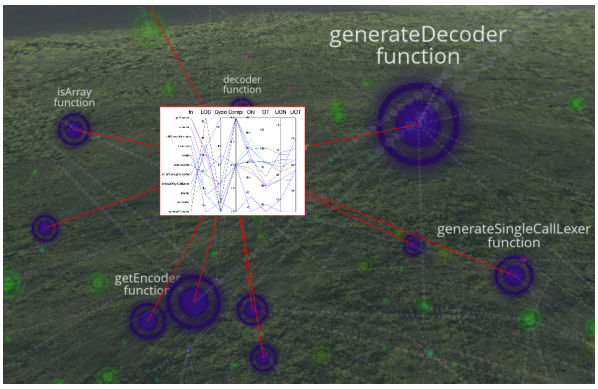

Fig. 7. The additional window with function metric for multiple selected nodes.

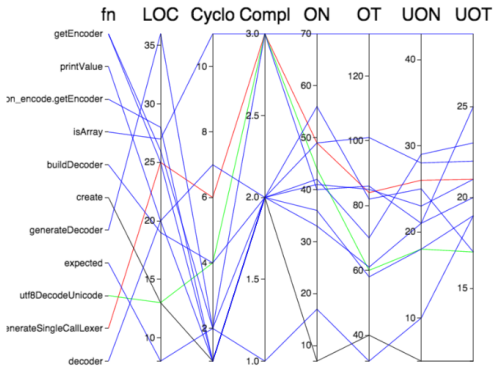

Fig. 8. Detailed view of the parallel coordinates from the window from Fig. 7 showing metrics for 11 functions.

see crossing reduction also in the upper figure parts due to the created edge bundles.

E. Providing details on demand

In previous sections we have shown that our visualization approach enables users to find software artifacts suitable for further inspection. To inspect details about functions in the visualized call graph we need a way to present the stored software metrics. However, a more important task is to compare software metrics for multiple functions. In our visualization system we provide details about nodes via additional windows floating near the selected call graph nodes directly in the 3D space, as show in Fig. 7. Placing the windows with detailed information directly into the graph visualization (and not as a standard GUI element of a desktop application) allows to disable all standard desktop GUI elements for augmented reality, as discussed in section IV.

After selecting one or multiple call graph nodes our system supports two floating window modes:

1) Multiple windows – for each selected node display one corresponding window
2) Single window – show one window with aggregated information for all selected nodes

Displaying multiple windows is usable mostly for a small number of inspected functions as the additional windows can occlude the visualized graph and each other. To compare software metrics for multiple functions the single window with aggregated information is more suitable. The single window uses the parallel coordinate visualization technique to visualize software metrics, as show in Fig. 7. Each extracted software metric type is mapped to a parallel coordinate and the selected functions are displayed as polygonal lines.

F. Analyzing software metrics

Fig. 8 shows a detailed view of parallel coordinates visualization from the Fig. 7, which allows to compare multiple software metric values to help us to better evaluate potentially problematic functions. Using the parallel coordinates visualization it is possible to inspect dozens of software metric types, which is usually not possible to map on visual attributes of graph nodes.

The Fig. 8 show the 11 functions selected in Fig. 7 and 7 different software metrics for these functions: LOC (lines of code), Cyclo (cyclomatic complexity), ON (number of operands), OT (number of operators), UON (number of unique operands), UOT (number of unique operators). We can see that there is a correlation between different values of software metrics. For example the functions *generateDecoder* and *getEncoder* have high lines of code but small cyclomatic complexity. On the other hand function *isArray* is the most complex function in this example. It has not only the highest cyclomatic complexity, but also all operands and operators counts are the highest in this visualization. From this we can assume that the *isArray* function is potentially critical for this software system and should be analyzed in detail.

Further steps required to prevent possible problems could be analyzing the callers of this function (using steps we described in section *C. Filtering*) and determining in which scenarios is this function used and how it affects overall system performance.

IV. SOFTWARE VISUALIZATION IN AUGMENTED REALITY

A. Augmented reality setup

In the previous section we described our software visualization system as it is used as a desktop VR system. The provided interaction methods for navigation, graph exploration, filtering and examining detailed information are based on the standard ”window, icon, menu, pointer” (WIMP) user interface paradigm and utilize keyboard and mouse devices.

Visualization in augmented reality needs additional enabling technologies [2], at least special displays and tracking devices. Our augmented reality setup consists of a spatial optical see-through display [24], a camera, a microphone, a MS Kinect v1 sensor and a Leap Motion sensor. The camera and the Kinect sensor are placed near the see-through display and are oriented towards the user that stands in front of the see-through display, as shown in Fig. 9. Tracking user’s face via camera is used to support depth effect: the graph is rotated and moves

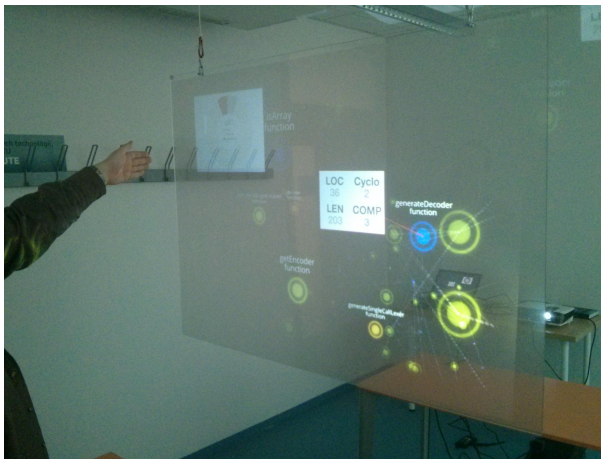

Fig. 9. Our augmented reality setup showing several nodes of the call graph with two windows containing software metrics.

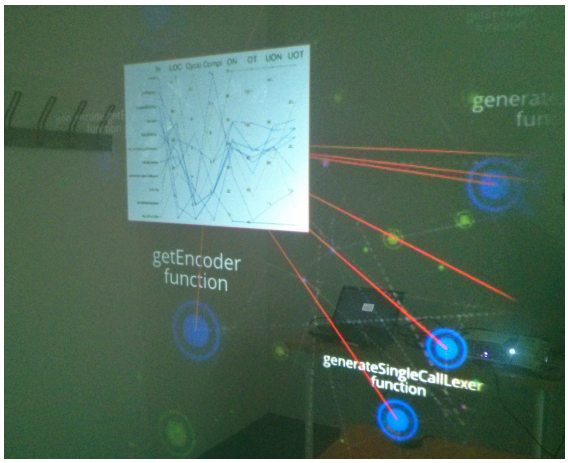

Fig. 10. A closer detail showing the combination of a window with parallel coordinates with call graph in augmented reality.

in opposite direction as the user's face moves in front of the see-through display.

The software visualization based on graph visualizations was transferred to augmented reality by removing all interfering GUI elements from the desktop VR version, so the optical see-through display shows only the visualized graph. Augmented reality solutions strive away from using keyboard and mouse as input devices. We therefore experimented with alternative interaction methods for graph exploration in augmented reality and focused on hand gestures, face tracking and voice commands to provide similar means for visual analysis as provided by the desktop VR version.

B. Interaction methods

Several types of interaction in information visualization have been identified [25]. Our visualization system deals with 3D graph visualizations, so we focused mainly on graph exploration and selection methods. For graph exploration tasks we use hand gestures that are tracked by the MS Kinect sensor and/or the Leap Motion sensor. Using the MS Kinect sensor, and OpenNI and NITE libraries, we track the user's hands which allow to rotate the visualized graph around all three axis. To get detailed information about nodes or edges it is often necessary to zoom the graph or moving the view into the graph. Zooming the graph is done by grabbing the graph and by moving the hand further/closer to the device. We used the OpenCV library to detect if user's hand is opened or closed, because the sensitivity of the MS Kinect v1 sensor does not provide enough detail. To freely navigate through the 3D graph we provide a virtual camera that allows flying through the visualization. This mode is activated by detecting and tracking two hands. One hand manipulates the forward camera speed, depending on the hand distance from the sensor, and gestures of the second hand affect the camera direction.

Similar gestures can be used to explore the graph using the Leap Motion sensor, which provides highly accurate information about hand positions, their orientation and can track individual fingers. To rotate the graph we directly track user's hand – closing hand initiates rotation, opening hand stops rotation. Similarly we use two hands to fly through the visualization: orientation of the right hand changes direction and open/closed left hand manipulates camera forward speed. Zooming is provided by a circle movement of index finger.

As our intent was to minimize the amount of used gestures, we decided to handle multiple actions with the same gesture. By that approach we divided possible actions into two main groups, *manipulation* and *selection*, and we provide methods of voice detection using the MS SpeechSDK for the MS Kinect sensor, which is used to switch between those two modes by voice command (the "manipulate" voice command for manipulation mode and the "select" voice command for selection mode). Voice commands are also used to activate several additional functions e.g. controlling the creation of a path in the visualized graph.

Selection of nodes is an important action for further actions e.g. getting detailed information about functions represented as nodes or navigating along a path in the call graph to identify called or calling functions. We provide methods for selecting nodes. To select a particular element, we use the concept of a 3D cursor, that tracks user's hand and a grab gesture (closing and reopening hand in short time interval). Unselecting can be done by the same gesture preformed in longer time interval. We also provide methods for multiple element selection. This option is activated by the voice command "select multiple" and after that, selection is also done by grab gesture. For every selected node (or group of nodes) we provide an option to show detailed information, using the details window mentioned in section E, by voice command "details". For example, user is allowed to explore details of single function in visualized graph (Fig. 9), or compare group of function against each other (Fig. 10). When the graph grows larger, users may get lost in the amount of visualized data, so we provide methods for navigating the graph structure via paths in graph by selecting nodes. Selection is preformed by previously mentioned grab gesture. Navigation using paths is

activated by voice command "create path". To provide better understanding of graph topology, we added functions that highlight the neighbor nodes of the currently selected node. We calculate the nearest neighbor node to the position of user's hand. Depending on actual node placement sometimes it is easier to select nearby nodes directly, but in other situations it may be easier to select edges, as the nodes are displayed further away. Using a circling hand gesture the user highlights individual edges and nodes connected to the starting selected node. After deciding on the target node, the user can perform a selection gesture and the view is centered on the target node. Repeating this process the user navigates the graph and creates a highlighted path that can be used e.g. to investigate the sequence of function calls in the visualized call graph.

V. EVALUATION

Our prototype for software visualization in augmented reality is in early stage, so a more formal evaluation is yet to be done. However, we performed an informal evaluation with a small group of 5 participants, that have basic knowledge of information visualization and were instructed how to use our prototype. Most participants stated, that identifying the most complex functions of the analyzed software using our visualization system was straightforward and helpful (using the desktop VR version with mouse and keyboard). Most participants appreciated the possibility to inspect software metrics of multiple functions using parallel coordinates. Graphs with bundled edges seemed more useful to 3 participants in task related to graph nodes. On the other hand, all participants noted, that interaction via hand gestures is not as straightforward as using mouse and keyboard, and needs more time to learn and use. Four participants confirmed that the hand-based interaction with visualized data is interesting and should be further investigated.

VI. CONCLUSIONS

In this paper we presented our graph-based software visualizations system. We presented a possible visual analysis process on two studied software systems. Our approach showed that using the provided visualization and interactions methods we are able to identify complex parts of the studied systems and to reason about the software metrics causing their complexity. Our second contribution is the experimental transition of the visualization to augmented reality. We proposed several interaction methods based on hand gestures. However, they only partly cover the necessary interactions for successful visual analysis and are currently not as effective as using standard mouse and keyboard interaction. Future work will be oriented to utilize head-mounted displays that could bring the visualized data "closer" to the user, so the data could rest on user's hand and be manipulated and explored with fingers.

ACKNOWLEDGMENT

This contribution is the partial result of the Research & Development Operational Programme for the project ITMS 26240220039, co-funded by the ERDF. This work was supported by the grant VEGA 1/0625/14.

REFERENCES

[1] D. Keim *et al.*, "Information visualization and visual data mining," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 8, no. 1, pp. 1–8, 2002.

[2] D. Van Krevelen and R. Poelman, "A survey of augmented reality technologies, applications and limitations," *International Journal of Virtual Reality*, vol. 9, no. 2, p. 1, 2010.

[3] A. R. Teyseyre and M. R. Campo, "An overview of 3d software visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 1, pp. 87–105, 2009.

[4] S. Diehl, *Software visualization: visualizing the structure, behaviour, and evolution of software*. Springer Science & Business Media, 2007.

[5] P. Caserta and O. Zendra, "Visualization of the static aspects of software: a survey," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 7, pp. 913–933, 2011.

[6] C. A. Shaffer *et al.*, "Algorithm visualization: The state of the field," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 3, p. 9, 2010.

[7] C. L. Jeffery, *Program monitoring and visualization: an exploratory approach*. Springer Science & Business Media, 2012.

[8] R. L. Novais, A. Torres, T. S. Mendes, M. Mendonça, and N. Zazworka, "Software evolution visualization: A systematic mapping study," *Information and Software Technology*, vol. 55, no. 11, pp. 1860–1883, 2013.

[9] I. Polášek *et al.*, "Information and knowledge retrieval within software projects and their graphical representation for collaborative programming," *Acta Polytechnica Hungarica*, vol. 10, no. 2, pp. 173–192, 2013.

[10] D. A. Keim, J. Kohlhammer, G. Ellis, and F. Mansmann, *Mastering the information age-solving problems with visual analytics*. Florian Mansmann, 2010.

[11] G.-D. Sun *et al.*, "A survey of visual analytics techniques and applications: State-of-the-art research and future challenges," *Journal of Computer Science and Technology*, vol. 28, no. 5, pp. 852–867, 2013.

[12] R. Souza, B. Silva, T. Mendes, and M. Manoel, "Skyscrapar: an augmented reality visualization for software evolution," in *II Brazilian Workshop on Software Visualization, Natal, Brazil*, 2012, pp. 17–24.

[13] D. Belcher, M. Billinghurst, S. Hayes, and R. Stiles, "Using augmented reality for visualizing complex graphs in three dimensions," in *Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2003, p. 84.

[14] B. S. Meiguins *et al.*, "Multidimensional information visualization using augmented reality," in *Proceedings of the 2006 ACM international conference on Virtual reality continuum and its applications*. ACM, 2006, pp. 391–394.

[15] B. Shneiderman, "The eyes have it: A task by data type taxonomy for information visualizations," in *Visual Languages, 1996. Proceedings., IEEE Symposium on*. IEEE, 1996, pp. 336–343.

[16] D. Keim, F. Mansmann, J. Schneidewind, H. Ziegler *et al.*, "Challenges in visual data analysis," in *Information Visualization, 2006. IV 2006. Tenth International Conference on*. IEEE, 2006, pp. 9–16.

[17] R. Ierusalimschy, L. H. De Figueiredo, and W. Celes Filho, "Lua-an extensible extension language," *Softw., Pract. Exper.*, vol. 26, no. 6, pp. 635–652, 1996.

[18] T. J. McCabe, "A complexity measure," *Software Engineering, IEEE Transactions on*, no. 4, pp. 308–320, 1976.

[19] C. Jones, "Software metrics: Good, bad and missing," *Computer*, vol. 27, no. 9, pp. 98–100, 1994.

[20] C. Bailey and W. Dingee, "A software study using halstead metrics," *ACM SIGMETRICS Performance Evaluation Review*, vol. 10, no. 1, pp. 189–197, 1981.

[21] S. Henry and D. Kafura, "Software structure metrics based on information flow," *Software Engineering, IEEE Transactions on*, no. 5, pp. 510–518, 1981.

[22] T. M. Fruchterman and E. M. Reingold, "Graph drawing by force-directed placement," *Softw., Pract. Exper.*, vol. 21, no. 11, pp. 1129–1164, 1991.

[23] D. Holten and J. J. Van Wijk, "Force-directed edge bundling for graph visualization," in *Computer Graphics Forum*, vol. 28, no. 3. Wiley Online Library, 2009, pp. 983–990.

[24] O. Bimber and R. Raskar, "Modern approaches to augmented reality," in *ACM SIGGRAPH 2006 Courses*. ACM, 2006, p. 1.

[25] J. S. Yi *et al.*, "Toward a deeper understanding of the role of interaction in information visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1224–1231, 2007.