VR City: Software Analysis in Virtual Reality Environment

Juraj Vincur, Pavol Navrat, Ivan Polasek
Faculty of Informatics and Information Technologies
Slovak University of Technology
Bratislava, Slovakia
{juraj.vincur, pavol.navrat, ivan.polasek}@fiit.stuba.sk

***Abstract*—This paper presents software visualization tool that utilizes the modified city metaphor to represent software system and related analysis data in virtual reality environment. To better address all three kinds of software aspects we propose a new layouting algorithm that provides a higher level of detail and position the buildings according to the coupling between classes that they represent. Resulting layout allows us to visualize software metrics and source code modifications at the granularity of methods, visualize method invocations involved in program execution and to support the remodularization analysis. To further reduce the cognitive load and increase efficiency of 3D visualization we allow users to observe and interact with our city in immersive virtual reality environment that also provides a source code browsing feature. We demonstrate the use of our approach on two open-source systems.**

Keywords—software cities; software visualization; virtual reality; software analysis

I. INTRODUCTION

Software maintenance and development depend on analysis and comprehension of the underlying system structure. Both, analysis and comprehension are time-consuming processes that require a huge amount of resources. These costs could be reduced by using supportive tools that provide more efficient view of the software. Among the many possible approaches, we decided to focus on software visualization, due to its capability to synthesize a large amount of information.

Software visualization is the art and science of generating visual representations of various aspects of software [1]. These aspects could be divided into three distinct kinds [1]: static, dynamic and evolution. The static aspects of software represent information that could be obtained without a program execution, but it is valid for all possible executions (static analysis). On the other hand, the dynamic aspects are the result of the dynamic analysis which requires program execution and resulted information is valid for particular run only. The visualization of the evolution usually shows how the static aspects of software change in time.

There are many approaches that together cover all three kinds of aspects. Most of them are 2D, but the recent trend is to explore 3D software visualizations since they can offer multiple advantages. Some experiments suggest that 3D representations of information structures are easier to identify and recall than 2D equivalents [2]. Ware and Franck [3] indicate that displaying data in three dimensions instead of two can make it easier to understand. 3D representations have also been shown to better support spatial memory tasks [4]. Error rate in identifying routes in 3D graphs is much smaller than in 2D [3], [5].

The biggest disadvantage of 3D approaches is intrinsic 3D navigation problem [6]. It is difficult to navigate in 3D spaces using a 2D input device (e.g. mouse) [7] and therefore, users may get disoriented [8]. In case of disorientation, cognitive load increases rapidly and advantages of a third dimension are abolished. One possible solution to this problem is the use of emerging technologies for displaying virtual reality (VR) that can employ the natural human movement, interaction and perception.

While VR has been successfully applied to scientific visualizations, little research has been done on using VR for software visualization [9], [10]. The main difference between traditional 3D approaches and VR is the aspect of immersion that allows the user to take advantage of their stereoscopic vision. Stereopsis can be a great benefit in disambiguating complex abstract representations and it also helps the viewer to judge relative size of objects and distances between them [11]. There are several other benefits of VR. Spatial memory of the user is utilized since VR environments can directly mimic the affordances offered by physical navigation. Moreover, VR environments can create a sense of presence in virtual space (immersion). Motion in a physical space, through exertions such as walking, has important cognitive consequences [12]. Motion cues combined with stereo viewing can substantially increase the size of the structure that can be perceived [13]. Users' understanding of a 3D structure improves when they are able to manipulate the structure [14]. VR also allows multiple programmers located around the world to join each other in a shared environment.

To further reduce the cognitive load, real-world metaphors could be applied to software visualization. They represent software with a familiar context, by using graphic conventions that the user immediately understands. Visualization based on real-world metaphors relies on the human natural understanding of the physical world, including spatial factors in perception and navigation and so it prevents the problem of the users' disorientation [15]. Several software metaphors have been proposed [16], [17], [18]. The city metaphor [18] is probably the most widely used one. Its potential has been

978-1-5386-2072-4/17 $31.00 © 2017 IEEE
DOI 10.1109/QRS-C.2017.88

shown in both the academic and the industrial world [19], [20]. The majority of proposed approaches address static aspects of software. They usually visualize software metrics combined with organization [21], [22], [23] and in some cases, even relationships between code fragments [20], [24], [25]. Limited research has been done in visualization of software evolution [26], [27] and dynamic aspects [20], [24], [28], [29], and moreover, we are aware of only one approach that covers all three kinds of aspects over time [20], [24]. The layout of the city is another not well covered area. Only few types of layouting algorithms have been developed. The traditional nested one [29], street layout [27] and graph based layout [23]. All of them represent classes by equal shapes (e.g. boxes), provide low-level of detail (method metrics could not be visualized unless buildings represent methods) and ignore relationships between them (e.g. coupling).

Some effort has been made to port software city metaphor into VR environment (without adapting it to VR affordances) [10] and the idea of collaboration has been also covered [11], [31]. However, resulted tools do not provide source code browsing feature, provide only limited detail and interaction options and does not fully employ natural human movement and interaction.

In this paper, we propose new layouting algorithm that considers coupling and allows us to visualize all three kinds of aspects with higher level of detail. Software metrics could be visualized for each method while classes are still represented by buildings. The varying shape of buildings allows users to see the distribution of selected metrics in class across methods what can be a support for them in fast identification of interfaces, abstract classes, abstract methods, getters/setters and code smells related to methods (e.g. god method, long method, complex switch statement) while still providing overall view of class metrics. For effective trace analysis, we create a trace mode, which provides the view, similar to disharmony maps. In this view the most frequently used methods and classes are the most visible. Floor color represents in which trace the method is involved. These features allow user to observe easily which parts of the system are related to which feature or test, what parts are common for multiple traces and to observe the ratio of called methods in the classes. Our animation of evolution shows how metrics of methods and classes change over time, how the number of methods increases or decreases and which developers are responsible for observed changes. The presence of developers is visualized by 3D primitives of distinct colors that float above the city.

To employ affordances of VR we allow users to observe our city in VR environment by utilizing head-mounted display (HMD) HTC Vive. This device is capable of precise room-scale tracking that enables interesting features such as navigation by walking. Interactions are realized by using two wireless controllers that are part of the device and provide intuitive way of grabbing, confirming/using and scrolling.

The rest of the paper is organized as follows. In Section II, related work is discussed. An overview of our visualization is provided in Section III, followed by case studies (Section IV). Conclusions and future work are discussed in Section V.

II. Related Work

This paper addresses two main topics that will be discussed in related work. The first one is the software city metaphor which we adapt and use to visualize software system. The second one is the application of VR affordances in software engineering.

A. The City Metaphor

Steinbrückner and Lewerentz [27] propose stable city layouts for evolving software systems, i.e. systems which are still being developed. Each street represents a certain package. Contained packages form branching streets. The system level is represented by a main road from which the top-level packages branch off. Classes are represented as square building plots which are attached to the street representing their package. They also use landscape elevations to directly represent a software systems development history in the layout. Elevation levels visualize information about the versions of software systems. This approach also provides evolution animation in which one can see how new elements are attached to the end of the street representing their package, removed elements disappear and how the size of a building changes when amount of coupling between class and package changes. Compactness of the city seems to be problematic when systems get often restructured. In this case, very long streets are produced.

Dugerdil and Alam [24], [29] use traditional containment layout that represents package hierarchies by rectangular zones on the ground. The darker the zone, the higher in the containment hierarchy. Relationships between classes can be displayed as solid pipes between buildings. The direction of relationship is represented by animation. Small segment of different color moves on pipe from source to target. Software metrics can be represented by both building height and texture. They also extend the Software City metaphor to represent execution traces. These are visualized in so-called night view in which lightness of scene is significantly reduced and only buildings that are part of a program execution are highlighted. This way they create disharmony map of the city, which allows user to easily identify classes that are involved in trace.

Panas et al. [23] propose the Cities metaphor in which methods are represented by buildings. These are placed on blue plates that symbolize classes. The blue plates are contained in green plates that represent packages (basically one city). The height of a green plate represents the depth of the package in the hierarchy. The cities layouts are computed using two methods: coarse-grained elements (e.g. packages) are positioned by a force-directed algorithm whereas fine-grained elements (buildings) are laid out compactly in a grid based manner. Combinations of metrics could be mapped to building texture and height.

Wettel and Lanza [21] create CodeCity visualization tool that displays classes as buildings, and packages as city districts. The size and the position of districts are defined by Treemap layout. Both building height and width are proportional to chosen software metrics. To address the need of higher level of detail in visualization they propose fine-grained representation. In this mode, methods are represented as equally-sized cuboids (“bricks”) that are laid out on top of each other in layers of 4.

Users can also switch into age map of city in which color of building represents its age. Age is defined as number of sampled versions that the artifact "survived". Using this map, old parts and the recently changed parts of the system can be easily identified. Users can also time-travel through age maps of different versions to see the system's evolution.

Caserta et al. [25] propose a new approach to visualization of relationships in software cities. They name it 3D Hierarchical Edge Bundles. The main idea is to display connections between fragments on top of a hierarchical representation. In this technique, Hierarchical Attraction Points are used to organize edges into clusters, which produce less overlapping and less occlusions. To show that their approach works independently of the layout, they apply it to two cities. One with nested and one with street layout.

Waller et al. [30] create tool named SynchroVis which provides the visualization of static and dynamic aspects of a software system. Moreover, it supports understanding of its concurrent behavior. They also apply nested layout, but in contrast to other approaches, the ground floor of the building represents the actual class object, while the upper floors represent dynamically created class instances. Operation calls contained in the program trace are represented by colored arrows between floors. The authors call them streets, but we do not see any similarity between these lines and actual city streets. In addition to classes, synchronization concepts (e.g. Monitor / Semaphore) are also represented by buildings that are localized in special districts.

Fittkau et al. [28] propose a live trace visualization tool which shows program traces during execution of the applications. It combines a landscape and a system level perspective. The landscape level perspective uses a 2D visualization employing a mix of UML deployment and activity diagram elements. System level perspective provides a visualization utilizing the city metaphor for each software system. In their metaphor, districts represent packages. Each package is visualized as a rectangular layer with a fixed height. Several layers are stacked upon one another to display corresponding package hierarchy. Buildings represent packages or classes. Buildings become districts when they are opened. The maximal count of current instances in each entity maps to the height of the corresponding building. The width of a building is determined by the number of classes inside the represented entity. If the entity is a class, the width is a constant minimal value. Communication is represented by pipes between entities. The thickness of the streets represents the call count between them.

B. *Application of VR Affordances*

Elliott et al. [9] describe both the affordances and challenges of VR tools in software engineering. As a demonstration, they implement two tools: RiftSketch and Immersion. RiftSketch is a live coding environment built for VR which allows users to manipulate a 3D scene. As they type code into the editor, the world around them is updated instantly. The editor is also nested in VR environment and code could be effectively updated using provided input methods without leaving the virtual world. Immersion tool provides an immersive environment for code review. Authors try to utilize spatial reasoning by placing groups of fragments into different sections of floor. Each floor section represents package and its color is related to amount of modifications. They believe that required motion cues (e.g. grabbing, walking) will improve spatial reasoning of reviewers and thus allow them to derive a mental model more easily. To summarize their early research identified, affordances are: spatial cognition, intuitive navigation and interactions, remote collaboration, immersion and live coding. On the other hand, open challenges are: complete separation from real world, efficient 3D mapping of non-3D objects and technology limitations (e.g. low resolution).

Fittkau et al. [10] present their modification of ExplorViz tool that allows users to explore software city in VR environment using head-mounted displays. Another contribution of the paper is the design of gesture-based interactions that are also evaluated on a sample of eleven participants.

Maletic et al. [11], [31] focus on the visualization of very large software systems by combining information visualization with virtual reality and computer supported cooperative work. Their vision is to support the development, maintenance, and reverse engineering by using a highly interactive, collaborative, and immersive environment. They present system Imsovision that uses CAVE as the primary representation medium. It is common to have several people standing in the CAVE at the same time thus this medium natively supports local collaboration. In their visualization, they employ a metaphor similar to Unified Single-View Visualization [23]. Imsovision focuses on static aspects, but is also able to capture the development of the represented software system. Two representations of different system revisions can be overlapped to highlight differences that describe the evolution.

III. Our Approach: VR City for Source Code Comprehension and Analysis

Our approach to software visualization combines affordances of software metaphors and virtual reality to fully employ advantages of third dimension. We utilize and modify the city metaphor to show all three kinds of software aspects with higher level of detail and allow users to observe and interact with our city in VR environment using HMD HTC Vive. Design, structure and features of the city are described in four subsections. First, we describe our city layout - how we position objects that represent methods and classes with respect to coupling and software metrics. Second, we specify the structure of the city, where objects are divided into several layers. In third subsection, we show how selected software aspects are addressed and visualized and in the last subsection, we describe how users can navigate in the city and interact with objects.

A. *Layouting Algorithm*

Our city layout is derived from undirected graph that reflects coupling in a software project. To obtain such structure, dependencies of classes within a Java project are automatically analyzed and recorded. In the graph, each class is

represented by a node and dependencies between them are represented by a single link. The weight of the link equals the total number of dependencies. With this structure, we follow these steps to calculate the layout:

1. *Calculate the spectral ordering of graph nodes:*
   1.1. Calculate Fiedler vector of the graph.
   1.2. Sort nodes by their corresponding elements in the Fiedler vector.
2. *For each class (node) in spectral ordering, calculate sorted array (descending order) that holds values of chosen software metric of all declared methods. Each sorted array represents building floors. The first value defines the size of ground floor and last value the size of top floor.*
3. *Calculate the order of the Hilbert curve* p*:*
   $$p = \lceil \log_4 n \rceil,$$
   where n *denotes the sum of values of ground floors.*
4. *Create ground floors of buildings:*
   4.1. Create n *cuboids and position them according to Hilbert curve of order* p.
   4.2. Merge blocks that belong to one class.
5. *Create other floors in similar way (step 4.). The starting position of next floor in Hilbert curve is always equal to the starting position of ground floor.*
6. *Create space between buildings to form streets.*

This layouting algorithm is derived from space-filling visualization technique for multivariate small-world graphs proposed by Wong et al. [32].

In resulted city, classes are represented by buildings and methods are represented by floors. Each floor consists of number of blocks (cubes) equal to value of chosen software metric (e.g. LOC of the method). In 3D space x, y position of each block is defined by Hilbert curve with its order as parameter. The z position is defined by floor number and floor height. Floor height could represent another software metric or could be set to the constant value. The starting position of each floor in Hilbert curve is equal to the starting position of the ground floor, thus at least one face of building is always straight.

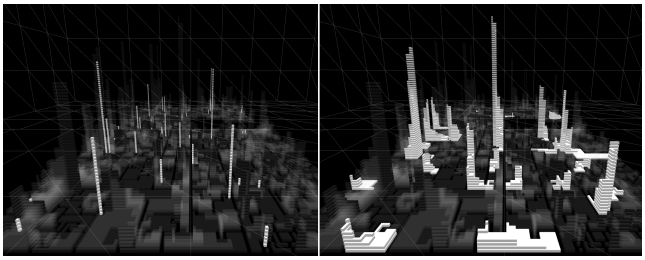

Fig. 1. Highlighted interfaces (left) and abstract classes (right).

Thanks to this layout the shape of building varies. As a consequence, one can easily distinguish between buildings that represent interfaces (skinny and tall), abstract classes (skinny at the top) and ordinary classes without providing any extra information (see Fig. 1). It is also easier to identify code smells related to methods (e.g. long method, god method) or to find out where the main logic of class is implemented.

B. Layers and their Content

Connections layer holds objects that represent relationships between visible entities. Each relationship is visualized by wide semitransparent line.

Authors' layer groups objects that represent authors. Each author is represented by sphere of unique color that levitates above the city. Its movement is defined by waypoints that reflect users' recent activity (e.g. positions of classes affected in last commits). Position of each waypoint is given as centroid of positions of related buildings. It also contains connection layer which is used to specify how authors contribute to source code fragments in time (see Fig. 2).

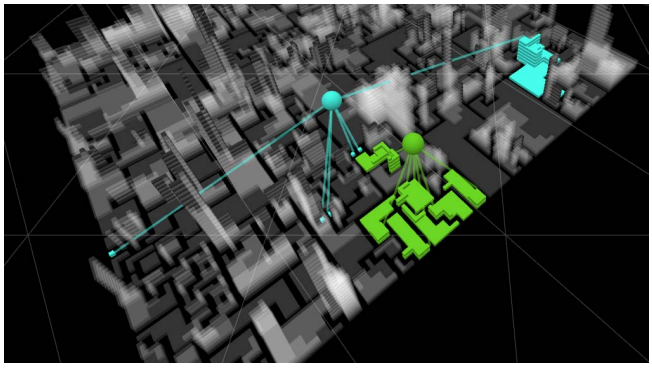

Fig. 2. Recent activity of two authors.

City layer consists of multiple groups. The first group contains all 3D objects (buildings) that represent classes. City layer also contains authors' layer. Connection layer could be added as well to visualize relationships between fragments. Whole layer could be easily translated, rotated or scaled. Scaling allows user to choose any size ranging from tiny representations that provide overall view of the entire system to huge representations that allow them to walk in software city and observe it in detail.

Fig. 3. Example of a expanded code label.

Code space layer allows users to browse source code directly in virtual reality environment. Visual representation of any method could be added to code space by clicking the corresponding floor. This representation consists of label,

which is basically a small plane with text information about parenting class (modifiers and qualified name) and method itself (method header). Code label could be extended by clicking UI buttons. In extended mode method body also becomes visible (see Fig. 3). Code label is always attached by a link to the floor that represents selected method (via its own connections layer) and can be freely moved in space by controllers. This way, user can create his own unlimited 3D workspace, where each fragment of code has its unique position. We believe that these features allow users to utilize their spatial memory more efficiently.

UI space layer provides access to various types of interactions. These can be selected from simple menu that is attached to the left controller. Available actions are: city transformations (e.g. scale, rotate, translate), authors animation, trace visualization, change of color scheme and teleport.

C. Visualized aspects

Our city metaphor is capable of visualizing:

- Static aspects – metrics and coupling that both have been already discussed in layout section.
- Dynamic aspects – trace visualization using disharmony maps.
- Evolution – contribution of authors and the change of static aspects in specified time period.

For effective trace analysis, we create a trace mode in which all buildings are first visualized by equally transparent objects of white color. User can play any recorded trace and choose its color. Each method invocation is then visualized by highlighting respective floor. The highlight event also decreases transparency of building and colorizes the floor per chosen color. In case that one method is involved in multiple traces, special reserved color is used. This way, the user can easily observe which parts of the system are related to which feature or test, what parts are common for multiple traces and observe the ratio of called methods in classes (see Fig. 4). To record a trace, we use Eclipse plugin inTrace that does not require any modification of the source code and allows user to define custom filters.

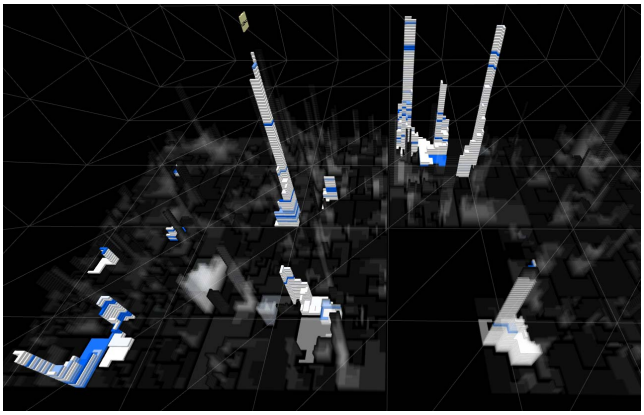

Fig. 4. City in trace mode after playing a trace recorded during initialization of JHotDraw applet (blue color assigned).

Our evolution animation shows how the city has been affected by developers from selected revision to the current state. It does not show all the changes; it only shows how buildings that are available in current revision have evolved. Thus, classes that were created and removed will never appear in animation. We also do not change the position of buildings in current implementation, so our animation shows how metrics have evolved and how methods were added or removed between commits. Since presence of authors is also visualized (authors' layer), it is easy to identify the author responsible for observed modifications. Example of a single commit is shown in Fig. 5.

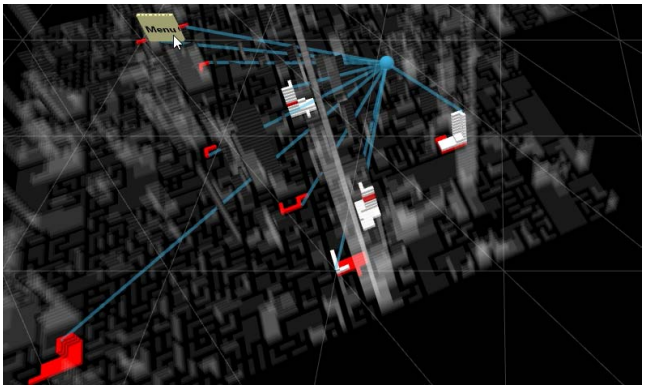

Fig. 5. Visualization of single commit and its author.

In addition, we also provide mode that supports remodularization analysis [33], [34]. In this mode, buildings are colored according to the chosen color scheme. Color could represent either relation of classes to semantic clusters or packages. Package hierarchy is considered by assigning similar colors to subpackages. This allows user to see how semantic topics and packages are distributed across software with respect to coupling (see Fig. 6). Semantic clusters are identified using method proposed by Vincúr and Polášek [35].

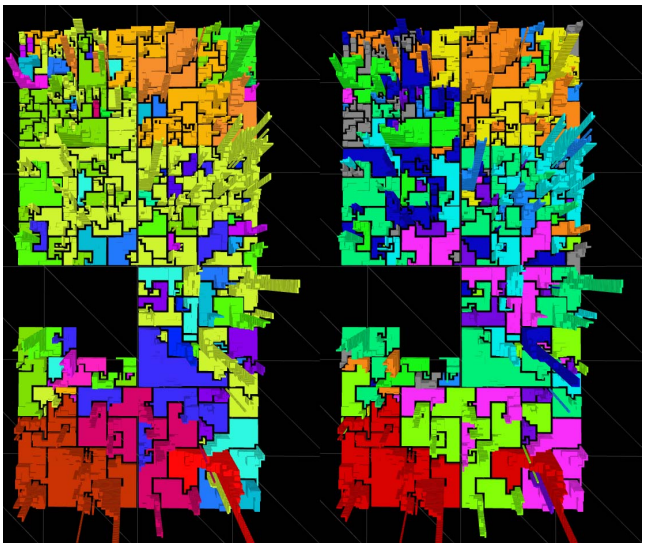

Fig. 6. Comparison of color schemes for JHotDraw. 24 packages (left) vs 11 semantic clusters (right).

D. Navigation and interaction

In our approach, we utilize HMD HTC Vive. This device allows users to interact with scene by controllers (see Fig. 7) that are also visible in virtual world and provide intuitive way of selecting objects, grabbing objects, triggering actions and scrolling. To select an object, user must point at it by controllers' pointer (basically line coming out of controller). Selected objects could be either grabbed or triggered. To grab an object, they simply push grip buttons, which is similar to motion required to grab a real-world object. To trigger an action attached to object, a user has to push the trigger button. This motion is analogous to shooting a gun. Scrolling is provided by trackpad button. To scroll up or down, a user just simply swipes his finger across trackpad area.

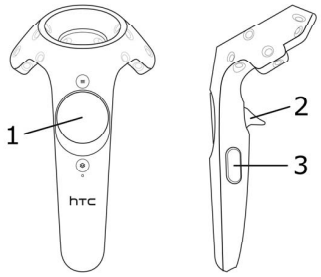

Fig. 7. HTC Vive controllers. 1 – trackpad, 2 – trigger, 3–grips.

Thanks to room-scale tracking of a user, provided by HTC Vive, navigation in scene is realized just as in a real environment. However, the size of the area in which user can walk is limited (approx. 3 by 3 meters). To overcome this limitation and to allow the user to walk in large scaled cities, we give him ability to teleport. This way, he can walk around objects to change his angle of view and position slightly and teleport to other parts of city in case he needs to travel larger distances (see video: https://goo.gl/inrcZs).

IV. Case Studies

Before discussing our approach we briefly introduce the systems that we used for case study (see Table I):

- *JHotDraw* - two-dimensional graphics framework for structured drawing editors.
- *JUnit* - simple framework to write repeatable tests.

TABLE I. Systems under study and number of visualized fragments

System	JHotDraw 7.0.6	JUnit 4
Packages	24	62
Classes	356	1149
Methods	3604	3758
Lines of code	40308	25162
Short SHA	untagged in git	26d6145

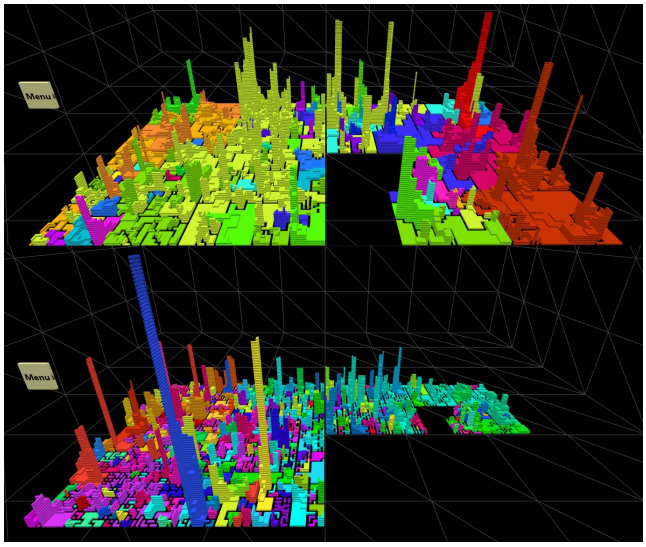

Fig. 8. Top bottom comparsion of JHotDraw (top) and JUnit (bottom).

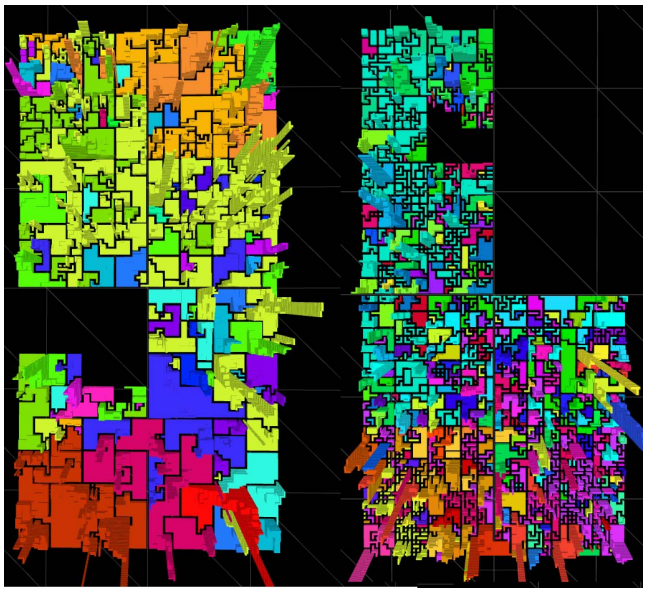

Fig. 9. Side by side comparsion of JHotDraw (left) and JUnit (right).

A. Static aspects

For demonstration purposes we choose lines of code metric (LOC) for both software systems. Resulted cities are shown in Fig. 9 and Fig. 8. It is easy to visually compare characteristics of both cities. JUnit project is obviously smaller than JHotDraw, although it consists of much more classes. This is caused by higher level of decomposition (smaller buildings). With respect to coupling, packages seem to be better organized in JHotDraw (better color segmentation). There is also significant overlap between packages and identified semantic clusters (see Fig. 6). One can see that many buildings in JHotDraw city has oversized base floor. This should be considered as a sign of the long method code smell. It is also possible that these methods contain complex switch statement

which may abuse OOP principles. Average number of methods per class is higher in JHotDraw which is also related to the level of decomposition. The height of the highest class in JUnit seems to be disproportional with respect to the others and it should be considered as a sign of the large class code smell.

B. Dynamic aspects

To demonstrate trace analysis using VR City tool, we first recorded program executions with Eclipse plugin inTrace. Filter was modified to ignore classes that are not part of the city. Exported files are automatically loaded into visualization and could be played from the menu. We select blue color to colorize methods involved in the trace and play recorded initialization of JHotDraw sample applet (see Fig. 4). Note that after the animation, user can easily observe which classes and methods are involved in selected program execution and also how often they are used. Afterwards, we select green color and play recorded trace generated during drawing events in JHotDraw sample applet (see Fig. 10). Now user can also see which methods are involved by second (green color) and both scenarios (purple color). Since aspect of time is preserved (animation) user can also determine the timespans in which code fragment was active. Discussed findings could be used to identify code fragments related to certain feature, to determine test coverage or for profiling.

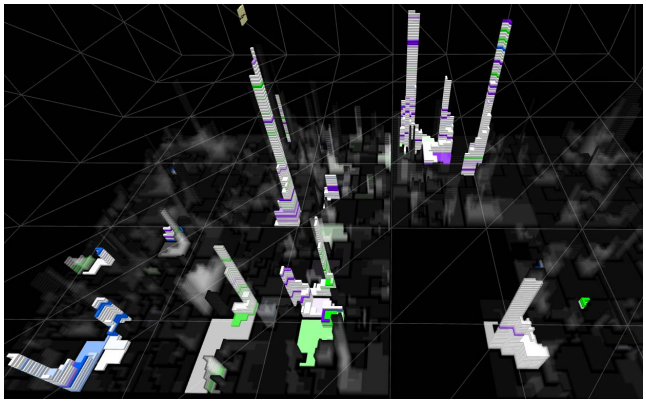

Fig. 10. City in trace mode after playing traces recorded during initialization (blue color assigned) and drawing (green color) in JHotDraw applet. Purple color is assigned to methods involved in both traces.

C. Evolution

User can select any sequence of commits by defining date or SHA of first and last commit. Commits are sampled using Git version control system. For presentation purposes, we sampled commits of JUnit from Jan 2015 to Mar 7 2017. As a result we obtained filtered sequence of 88 commits from 30 authors. Animation starts when a user selects the sequence from menu. After that, author of first commit is visualized and moves to his first waypoint (centroid of positions of affected classes). When author gets near the destination user can see which classes and methods were modified in current commit (see Fig. 5). User can stop the animation and observe modifications in detail (e.g. browse code). He can also apply filters to the sequence to see how only selected authors contribute to the software system or how only selected buildings evolve in time. Analogously, whole sequence of commits is animated (see video: https://goo.gl/inrcZs). Although we do not see practical use of visualization of evolution in software development in general, we think that presented approach might be employed in the process of code review as suggested in [9].

V. Conclusion and Future Work

We proposed an approach to software visualization based on 3D city metaphor, which allows users to explore object-oriented software systems in virtual reality environment. We also proposed and employed new layouting algorithm that allows us to visualize all three kinds of software aspects with higher level of detail. To fully employ discussed advantages of third dimension we allowed users to observe and interact with our city in VR using HMD HTC Vive. In this virtual environment, we provided to the users intuitive way of navigation and interaction with city objects and also the source code browsing feature.

We briefly demonstrated usage of our tool on two open-source projects and discussed its possible applications in maintenance and development tasks.

The main contributions of this paper are:

- A VR approach for exploring object-oriented software systems and related analysis data in virtual reality environment that provides intuitive and direct way of navigation (e.g. by walking) and interaction (e.g. grabbing objects by hands).
- New layouting algorithm that positions buildings with respect to coupling and allows us to visualize all three kinds of software aspects in single unified view at the granularity of methods.

As future work we would like to combine our layouting algorithm with nested layout or graph layout to produce cities that are more stable in time. We would also like to add the visualization of automatically identified code smells and create online tool for public use that will serve as a GitHub extension. We are planning to evaluate our approach with HMD that is capable of eye tracking (e.g. FoveVR) and add a mode for augmented reality glasses (e.g. Meta 2).

Acknowledgment

This work was supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/0646/15 and No. VG 1/0752/14.

This contribution is also a partial result of the Research & Development Operational Program for the project Research of Methods for Acquisition, Analysis and Personalized Conveying of Information and Knowledge, ITMS 26240220039, co-funded by the ERDF.

References

[1] S. Diehl, “Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software," Springer Verlag, Inc., 2007.

[2] P. Irani and C. Ware, “Diagrams based on structural object perception," In Proceedings of the working conference on Advanced visual interfaces (AVI '00). ACM, New York, NY, USA, 61-67.

[3] Ware, C. and Franck, G., (1994), "Viewing a Graph in a Virtual Reality Display is Three Times as Good as a 2D Diagram", in Proceedings of IEEE Visual Languages, pp. 182-183.

[4] M. Tavanti and M. Lind, "2D vs 3D, implications on spatial memory," IEEE Symposium on Information Visualization, 2001. INFOVIS 2001., San Diego, Ca, USA, 2001, pp. 139-145.

[5] Ware, C., Hui, D., and Franck, G., (1993), "Visualizing Object Oriented Software in Three Dimensions", in Proceedings of CASCON'93, Toronto, Ontario, Canada, October, pp. 612-620.

[6] G. Parker, G. Franck, and C. Ware, "Visualization of Large Nested Graphs in 3D: Navigation and Interaction," J. Visual Languages and Computing, vol. 9, no. 3, pp. 299-317, 1998.

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

[8] K. P. Herndon, A. van Dam, and M. Gleicher, "The challenges of 3D interaction: A CHI '94 Workshop," SIGCHI Bull., vol. 26, no. 4, pp. 36–43, Oct. 1994.

[9] A. Elliott, B. Peiris and C. Parnin, "Virtual Reality in Software Engineering: Affordances, Applications, and Challenges," 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015, pp. 547-550.

[10] F. Fittkau, A. Krause and W. Hasselbring, "Exploring software cities in virtual reality," 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), Bremen, 2015, pp. 130-134.

[11] J. I. Maletic, J. Leigh and A. Marcus, "Visualizing software in an immersive virtual reality environment," 2001 ICSE Workshop on Software Visualization, Toronto, 2001, pp. 49-54.

[12] M. Oppezzo and D. L. Schwartz, "Give your ideas some legs: The positive effect of walking on creative thinking," Journal of Experimental Psychology: Learning, Memory, and Cognition 40, 4 (2014), 1142--1152.

[13] Ware, C. and Franck, G., (1996), "Evaluating stereo and motion cues for visualizing information nets in three dimensions", ACM Transaction on Graphics, vol. 15, no. 2, April, pp. 121-140.

[14] Hubona, G. S., Shirah, G. W., and Fout, D. G., "3D Object Recognition with Motion", in Proceedings of CHI'97, 1997, pp. 345-346.

[15] P. Caserta and O. Zendra, "Visualization of the Static Aspects of Software: A Survey," in IEEE Transactions on Visualization and Computer Graphics, vol. 17, no. 7, pp. 913-933, July 2011.

[16] H. Graham, H. Yang, and R. Berrigan, "A Solar System Metaphor for 3D Visualisation of Object-Oriented Software Metrics," Proc. Australasian Symp. Information Visualization, vol. 35, pp. 53-59, 2004.

[17] A. Ghandar, A. S. M. Sajeev, and X. Huang "Pattern puzzle: a metaphor for visualizing software complexity measures, " In Proceedings of the 2006 Asia-Pacific Symposium on Information Visualisation - Volume 60 (APVis '06), Kazuo Misue, Kozo Sugiyama, and Jiro Tanaka (Eds.), Vol. 60. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, 221-224.

[18] C. Knight and M. Munro, "Virtual but visible software," 2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics, London, 2000, pp. 198-205.

[19] R. Wettel, M. Lanza and R. Robbes, "Software systems as cities: a controlled experiment," 2011 33rd International Conference on Software Engineering (ICSE), Honolulu, HI, 2011, pp. 551-560.

[20] M. Lanza, H. Gall and P. Dugerdil, "EvoSpaces: Multi-dimensional Navigation Spaces for Software Evolution," 2009 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, 2009, pp. 293-296.

[21] R. Wettel and M. Lanza, "Visualizing Software Systems as Cities," 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, Banff, Ont., 2007, pp. 92-99.

[22] T. Panas, R. Berrigan and J. Grundy, "A 3D metaphor for software production visualization," Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003., 2003, pp. 314-319.

[23] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen and R. Vuduc, "Communicating Software Architecture using a Unified Single-View Visualization," 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), Auckland, 2007, pp. 217-228.

[24] S. Alam and P. Dugerdil, "EvoSpaces Visualization Tool: Exploring Software Architecture in 3D," 14th Working Conference on Reverse Engineering (WCRE 2007), Vancouver, BC, 2007, pp. 269-270.

[25] P. Caserta, O. Zendra and D. Bodénès, "3D Hierarchical Edge bundles to visualize relations in a software city metaphor," 2011 6th International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), Williamsburg, VA, 2011, pp. 1-8.

[26] R. Wettel and M. Lanza, "Visual Exploration of Large-Scale System Evolution," 2008 15th Working Conference on Reverse Engineering, Antwerp, 2008, pp. 219-228.

[27] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," 2010 5th international symposium on Software visualization (SOFTVIS), New York, USA, 2010, pp. 193-202.

[28] F. Fittkau, J. Waller, C. Wulf and W. Hasselbring, "Live trace visualization for comprehending large software landscapes: The ExplorViz approach," 2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, 2013, pp. 1-4.space filling graph

[29] P. Dugerdil and S. Alam, "Execution Trace Visualization in a 3D Space," Fifth International Conference on Information Technology: New Generations (itng 2008), Las Vegas, NV, 2008, pp. 38-43.

[30] J. Waller, C. Wulf, F. Fittkau, P. Döhring and W. Hasselbring, "Synchrovis: 3D visualization of monitoring traces in the city metaphor for analyzing concurrency," 2013 First IEEE Working Conference on Software Visualization (VISSOFT), Eindhoven, 2013, pp. 1-4.

[31] J. I. Maletic, J. Leigh, A. Marcus and G. Dunlap, "Visualizing object-oriented software in virtual reality," Proceedings 9th International Workshop on Program Comprehension. IWPC 2001, Toronto, Ont., 2001, pp. 26-35.

[32] P. C. Wong, H. Foote, P. Mackey, G. Chin, Z. Huang and J. Thomas, "A Space-Filling Visualization Technique for Multivariate Small-World Graphs," in IEEE Transactions on Visualization and Computer Graphics, vol. 18, no. 5, pp. 797-809, May 2012.

[33] G. Santos, M. T. Valente and N. Anquetil, "Remodularization analysis using semantic clustering," 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), Antwerp, 2014, pp. 224-233.

[34] I. Candela, G. Bavota, B. Russo, and R Oliveto, "Using Cohesion and Coupling for Software Remodularization: Is It Enough", ACM Trans. Softw. Eng. Methodol. 25, 3, Article 24 (June 2016), 28 pages.

[35] J. Vincúr and I. Polášek, "An Incremental Approach to Semantic Clustering Designed for Software Visualization". In: Janech J., Kostolny J., Gratkowski T. (eds) Proceedings of the 2015 Federated Conference on Software Development and Object Technologies. SDOT 2015. Advances in Intelligent Systems and Computing, vol 511. Springer, Cham.