Immersive Coding: A Virtual and Mixed Reality Environment for Programmers

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
email: roy.oberhauser@hs-aalen.de

***Abstract*—While virtual reality (VR) has been applied to various domains to provide new visualization capabilities, the leveraging of VR capabilities for programmers doing software development or maintenance has not been sufficiently explored. In this paper, we present a VR environment for programmers called MR-FTC (mixed-reality FlyThruCode) providing software code structure visualization in multiple metaphors with integrated real keyboard/mouse viewing and interaction in mixed reality (MR) to enable basic programming task support without leaving the VR environment. A prototype implementation is described, with a case study demonstrating its feasibility and initial empirical evaluation results showing its potential. This MR solution concept enables programmers to benefit from VR visualization while supporting their more natural keyboard interaction for basic code-centric tasks. This can support programmers in exploring, understanding, and directly interacting with and maintaining program code while leveraging new VR paradigms and capabilities.**

Keywords—mixed reality; virtual reality; programming; software engineering; software visualization.

I. INTRODUCTION

As digitalization progresses, the volume of program source code created and maintained worldwide is increasing steadily. For instance, Google is said to have at least 2bn lines of code (LOC) accessed by over 25K developers [1], and GitHub has over 61m repositories and 22m developers [2]. Worldwide it has been estimated that well over a trillion LOC exist with 33bn added annually [3]. This is exacerbated by a limited supply of programmers and high employee turnover rates for software companies, e.g., 1.1 years at Google [4].

A challenge faced by programmers who are now more frequently facing unfamiliar preexisting codebases is how to familiarize and understand its structure in a short time. Code structures and their dependencies can be difficult to visualize. According to F. P. Brooks Jr., the invisibility of software is an essential difficulty of software construction because the reality of software is not embedded in space [5]. While common display forms used in the comprehension of source code include text and the two-dimensional Unified Modeling Language (UML), there may be a place for leveraging the opportunities afforded by a virtual reality (VR) for software visualization to make it seem less abstract and a more immersive experience for programmers. For instance, it could enhance the ability to comprehend and navigate software structures in a different reality without relying on the typical abstractions they are given, such as hierarchical file structures.

In our prior work, we developed VR-FlyThruCode (VR-FTC) [6][7], an immersive VR software visualization environment supporting flythrough navigation and virtual tablet and virtual keyboard interaction within dynamically-switchable metaphors but which lacked any support for keyboard usage. Programmers typically are keyboard and mouse/trackpad centric when interacting with code, which would typically require removing the VR headset and leaving the VR environment. Virtual keyboards which require selecting one key at a time using the VR controllers are an inefficient means for code input for experienced programmers.

This paper contributes a solution concept enabling mixed reality FTC (MR-FTC) keyboard and mouse viewing and interaction and provides an initial empirical evaluation. Our mixed reality (MR) solution concept enables real keyboard and mouse/trackpad integration while remaining in an immersive visualization and navigation experience through the code structures, allowing the user to view logical modularization and dependencies while enabling code object creation/deletion and source code modification with real keyboard access in the VR environment.

The paper is organized as follows: the next section discusses related work; Section III then describes the solution concept. Section IV provides details about our prototype implementation of the solution concept. In Section V, the evaluation, based on a case study, is described, which is followed by a conclusion.

II. RELATED WORK

As to non-VR software visualization, Teyseyre and Campo [8] give an overview and survey of 3D software visualization tools across the various software engineering areas. Software Galaxies [9] provides a web-based visualization of dependencies among popular package managers and supports flying, whereby every star represents dependency-clustered packages. CodeCity [10] is a 3D software visualization approach based on a city metaphor and implemented in SmallTalk on the Moose reengineering framework [11]. Rilling and Mudur [12] use a metaball metaphor (organic-like n-dimensional objects) combined with dynamic analysis of program execution. X3D-UML [13] provides 3D support with UML in planes such that classes are grouped in planes based on the package or hierarchical state machine diagrams.

Work on VR-based software visualization includes Imsovision [14] which visualizes object-oriented software in

 ISBN: 978-1-61208-590-6

VR using electromagnetic sensors attached to shutter glasses and a wand for interaction. ExplorViz [15] is a Javascript-based web application that uses WebVR to support VR exploration of 3D software cities using Oculus Rift together with Microsoft Kinect for gesture recognition.

With regard to MR and augmented reality support for programming tasks, [16] describe an approach for authoring tangible augmented reality applications with regard to scenes and object behaviors within the AR application being built, so that the development and testing of the application can be done concurrently and intuitively throughout the development process. Billinghurst and Kato [17] show possible concepts for collaboration in VR, but do not depict a keyboard or show how programming task support would work.

In contrast to the above work, the MR-FTC approach combines VR and game engines for software visualization and direct code access, support multiple metaphors, supporting basic programming tasks by integrating real keyboard and mouse views and interaction in the VR environment. To the best of our knowledge, no other software visualization in VR includes real hardware and mouse support.

III. SOLUTION

Our MR-FTC concept uses VR flythrough for visualizing program code structure or architecture, such as software architecture recovery, which can be useful in software maintenance tasks. Our inherent 3D application domain view visualization [8] arranges customizable symbols in 3D space to enable users to navigate through an alternative perspective on these often hidden structures. Certain metadata not readily accessible is visualized, such as the relative size of classes (not typically visible until multiple files are opened or a UML class diagram is created), the relative size of packages to one another, and the number of dependencies between classes and packages. To support programming within the VR environment, our solution concept utilizes MR for integrating keyboard and mouse access to permit coding.

A. Principles

Our previous work [7] details the solution principles which we summarize here: multiple customizable 3D visual metaphors (dynamically switchable between universe and terrestrial), flythrough navigation, a grouping metaphor (solar systems and glass bubbles), a connection metaphor (colored light beams), and a virtual tablet to provide information and interaction capabilities on tagging, metrics, UML, filtering, source code, and project configuration. Figure 1 shows VR system interaction (without the keyboard).

A further solution principle specific to this paper is *MR keyboard/trackpad/mouse integration*. A live camera view is integrated into the VR landscape. This allows the user to determine where their hands and fingers are relative to the actual hardware. Assuming the subject is seated, for instance, tilting their head down can be interpreted as a gesture to activate a live webcam on the VR headset, activating MR, similar to the natural head movement made at a desktop PC to look at the keyboard.

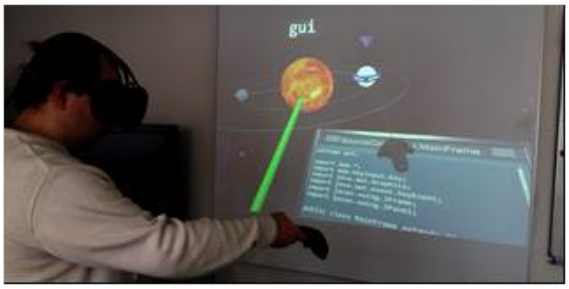

Figure 1. System setup with a subject wearing a Vive HTC headset, the controller visible in a universe metaphor scene selecting a planet (a class)).

B. Process

The process used by the solution approach consists of interaction and round-trip support for change propagation, whereby any changes to code outside of VR is reflected in VR, and any changes made to code in VR is reflected both with the VR visualization and to the actual code.

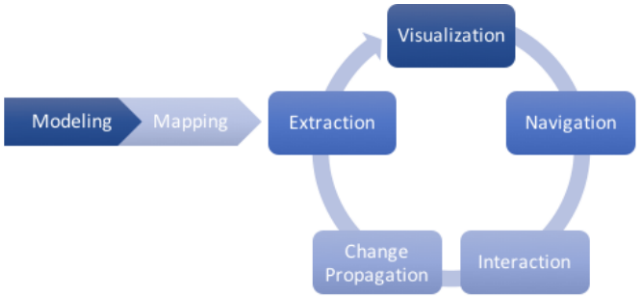

Figure 2. MR-FlyThruCode process steps.

The steps shown in Figure 2 are as follows:

1) Modeling: the modeling of generic program code structures, metrics, and artifacts as well as visual objects.

2) Mapping: the mapping of a program code artifact model to a visual object metaphor, such as universe or terrestrial objects.

3) Extraction: extracting a given project's metadata, structure (via source code import and parsing), and metrics.

4) Visualization: visualizing a given model instance within a metaphor.

5) Navigation: supporting navigation through the model instance (via camera movement based on user interaction), to provide a flythrough experience.

6) Interaction: providing interaction via hardware, such as the VR controller and keyboard or trackpad/mouse, or virtual objects, such as a virtual tablet and the visual equivalent of code artifacts that can be created, deleted, displayed (including metrics or metadata), or modified using a virtual tablet, virtual keyboard, and virtual controllers.

7) Change Propagation: for any interactions that incur changes (create/delete/modify), a background process is triggered that restarts step 3 and step 4 updates the visualization. For instance, if methods were added or

 ISBN: 978-1-61208-590-6

removed, then the height of the buildings or the size of the planet would be affected.

IV. IMPLEMENTATION

We utilized the Unity game engine for 3D visualization due to its multi-platform support, VR integration, and popularity, and for VR hardware both HTC Vive, a room scale VR set with a head-mounted display with an integrated camera and two wireless handheld controllers tracked using two 'Lighthouse' base stations.

We integrated a live camera view into the VR landscape via a virtual plane object. This allows the user to determine where their hands and fingers are relative to the actual keyboard. When the VR user tilts their head down, it is interpreted as a gesture to activate the live webcam on the VR headset and blend this into the virtual plane object. When the head is tilted starkly up, MR is deactivated.

Our MR-FTC architecture is shown in Figure 3.

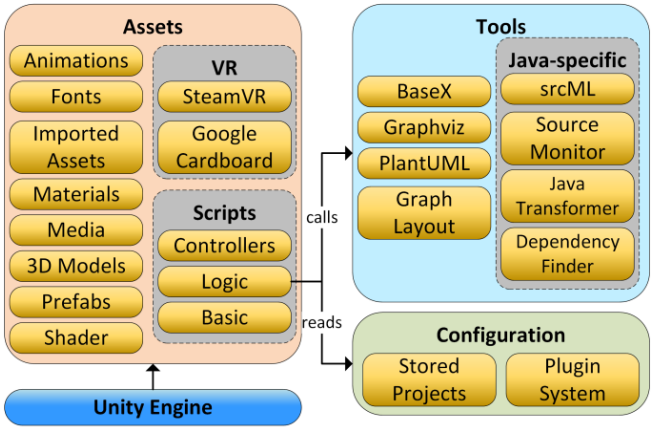

Figure 3. MR-FlyThruCode software architecture.

Assets are used by the Unity engine and consist of Animations, Fonts, Imported Assets (like a ComboBox), Materials (like colors and reflective textures), Media (like textures), 3D Models, Prefabs (prefabricated), Shaders (for shading of text in 3D), VR SDKs, and Scripts. Scripts consist of Basic Scripts like user interface (UI) helpers, Logic Scripts that import, parse, and load project data structures, and Controllers that react to user interaction. Logic Scripts read Configuration data about Stored Projects and the Plugin System (input in XML about how to parse source code and invocation commands). Logic Scripts can then call Tools consisting of General and Language-specific Tools. General Tools currently consist of BaseX, Graphviz, PlantUML, and Graph Layout - our own version of the KK layout algorithm [18] for positioning objects. Java-specific tools are srcML, Campwood SourceMonitor, Java Transformer (invokes Groovy scripts), and Dependency Finder. Our Plugin system enables additional tools and applications to be easily integrated.

A. Code Information Extraction

srcML [19] is used to extract existing code structure information into our model. The source code is converted into XML and stored in the BaseX XML database. Campwood SourceMonitor and DependencyFinder are used to extract code metrics and dependency data, and plugins with Groovy scripts and a configuration are used to integrate the various tools.

B. Project Structure Model

A software project contains the following files, which are used to access the data mapped to the VR objects:

- *metrics_{date}.xml*: metrics obtained from SourceMonitor and DependencyFinder are grouped by project, packages, and classes.
- *source_{date}.xml*: holds all classes in XML
- *structure_{date}.xml*: contains the project structure and dependencies utilizing the DependencyFinder.
- *swexplorer-annotations.xml*: contains user-based annotations (tags) with color, flag, and text including both manual and automatic (pattern matching) tags.
- *swexplorer-metrics-config.xml*: contains thresholds for metrics.
- *swexplorer-records.xml*: contains a record of each import of the same project done at different times with a reference to the various XML files, such as source and structure for that import. This permits changing the model to different timepoints as a project evolves.

V. EVALUATION

The evaluation consists of a case study with usage by master computer science students currently involved in the prototype implementation and familiar with VR.

A. Visualization and Navigation

Two metaphors were used for visualization: Figure 4 shows the universe or space metaphor, where planets represent classes which are grouped in solar systems. Figure 5 shows the terrestrial or city metaphor, where buildings represent classes with a label at the top and the number of stories represents the number of methods in that class, classes being grouped by glass bubbles. For the connection metaphor, in both metaphors colored light beams were are to show directional dependencies between classes or packages (see Figure 6 with extra cones placed around the light beams). To assist the user in remembering objects or characteristics, tagging is supported, which allows any label to be placed on an object (e.g., the Important Tag in Figure 4).

To realize flythrough navigation, the camera position is moved with the touchpad on the left controller of the HTC Vive controlling altitude (up, down) and the one on the right hand controlling direction (left, right, forward, backward). The controllers are shown in the scenery when they are within the view field. A virtual laser pointer was created for selecting objects, as was a virtual keyboard (in case no real keyboard is accessible or desired) to support text input for

 ISBN: 978-1-61208-590-6

searching, filtering, and tagging. Menus and screens showing source code, code metrics, UML, tags, filtering, and project data are placed on the screen of the virtual tablet. To highlight a selected object, we utilized a 3D pointer in the form of a rotating upside-down pyramid (see Figure 4 and Figure 5). This was needed because, once an object is selected, after navigating away one may lose track of where the object was, especially if the object was small relative to its surrounding objects.

Figure 4. Universe metaphor: selected planet (class) has inverted pyramid as arrow, tags evident, and tablet is visible and shows the class interface.

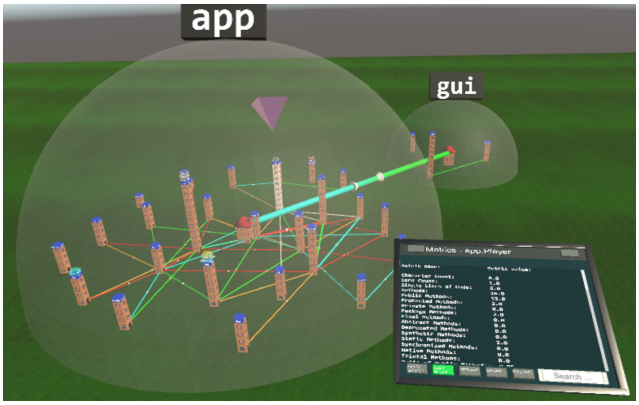

Figure 5. Terrestrial metaphor with bubbled cities (packages) and oracle with metrics for selected object (pyramid pointer).

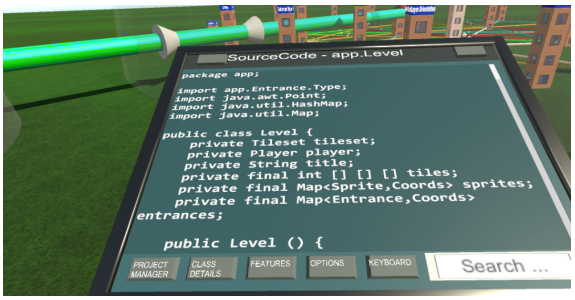

Figure 6. The virtual tablet interface showing source code. A bidirectional (dependency) pipe is visible in the background.

B. *Mixed Reality Interaction*

To achieve MR and access the video stream of the front camera, the standard Unity Plane asset was used and a small Script added to this game object. From the SteamVR TrackedCamera script, the method VideoStreamTexture is invoked, which returns a Texture that is set to the material of the Plane (see Figure 7 and Figure 8).

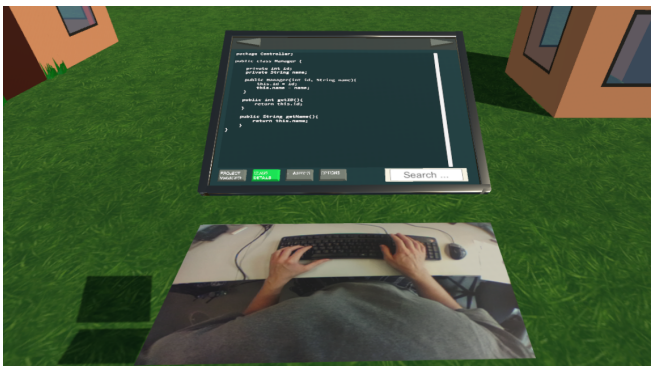

Figure 7. Coding with MR view of keyboard and mouse blended in and scroll bar shown on the virtual tablet.

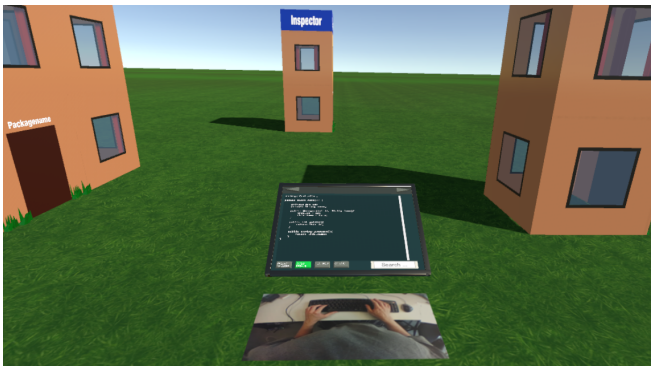

Figure 8. Far view in MR of keyboard with VR object visible.

The Vive camera must be activated (manually) in the SteamVR settings. We chose to automatically activate and show MR when the user's tilts the goggles low enough, as one would if one were to wish to see the keyboard when using it, and turn MR off again if one tilts the head up far enough again. Keyboard and mouse inputs are accessible at any time, not just when MR is activated.

For an initial empirical evaluation of the MR keyboard capability, we utilized a convenience sample of Computer Science students.

For evaluating typing speed in particular for comments which are full words without special characters, five subjects were required to write two unique pangrams consisting of 18 words using a text editor (Notepad++), the MR keyboard, and the VR only keyboard. We varied the starting configuration order among the subjects to minimize training effects. As shown in Table I, the text editor was the most efficient with 50 seconds duration and 22.5 words per minute (wpm) with an average error rate of 3.3%. With MR 75 seconds were required (16.0 wpm) with an error rate of 3.3%. With the VR keyboard 110 seconds were required

 ISBN: 978-1-61208-590-6

(10.1 wpm) with an error rate of 4.4%. Thus the MR keyboard was faster than the VR keyboard and did not exhibit a higher error rate. However, the subjects needed 11 seconds on average between laying down the VR controllers and pressing the first letter on the keyboard.

TABLE I. TEXT EDITOR, MR, AND VR PANGRAM MEASUREMENTS (AVERAGE)

	Text Editor	MR	VR
Duration (seconds)	50	75	110
Words per minute	22.5	16.0	10.1
Error rate	3.3%	3.3%	4.4%

For evaluating programming, four subjects were required to view a certain class and then create a class and were given certain specified modifications thereafter (creating some object and setting some variable to some value) using either a text editor or the MR keyboard. As seen in Table II, using a text editor (Notepad++), they needed on average 50 seconds to analyze a similar class, 30 seconds to create a new class, and 144 seconds to do the programming. Using the MR keyboard, they needed 84 seconds to analyze a similar class, 77 seconds to create a class, and 245 seconds to complete the programming.

TABLE II. TEXT EDITOR AND MR MEASUREMENTS (AVERAGE IN SECONDS)

	Analysis	Class Creation	Programming
Text editor	50	30	144
MR	84	77	245

While a text editor remains more efficient, usage of the MR keyboard was faster than a purely VR keyboard and, once familiar with a certain keyboard, we expect the overhead of MR to be reduced to an acceptable level given sufficient practice. The overhead of switching between VR controllers to keyboard and back again can be seen as analogous to the overhead of keyboard use on a PC and moving the hand to the mouse and back again and may thus be considered acceptable for certain users. We will investigate this further in future work.

We were pleased that none of the subjects reported motion sickness despite the inclusion of MR and the average response to how they felt afterwards was 4.75 (on a scale of 1 to 5 with 5 best).

Although the keyboard was a German layout keyboard, we noted that some subjects already had used that specific keyboard model before (Logitech K280e) while others had not and thus needed more time to search for certain specific keys. In searching they needed to get close with the VR goggles to see the key label, so we will consider providing a zoom or magnification option in the interface in the future.

C. *Coding Support Interaction and Change Propagation*

To provide basic programming support, package and class creation are supported via a VR controller wrist-based selection capability in both metaphors (see Figure 9 and Figure 10). Thereafter a screen appears on the VR tablet for naming the object (Figure 11 and Figure 12), which can be done with a real or virtual keyboard and the change is then propagated. For deletion, an object must first be selected, then a deletion option appears as an "X" (not shown) on the left VR controller, and if this "X" is targeted by the right controller, the object and underlying source code file (or directory in case of a package) is then removed from the file system via change propagation. File editing can be performed by pointing the VR controller or mouse at a character on the virtual tablet when it is showing source code and then typing with the physical (or virtual) keyboard. A blinking cursor is then shown and editing via insertion/deletion can be performed (selection not yet supported). The code state is transmitted and change propagation triggered when the virtual tablet is switched to a different view, updating the visualization.

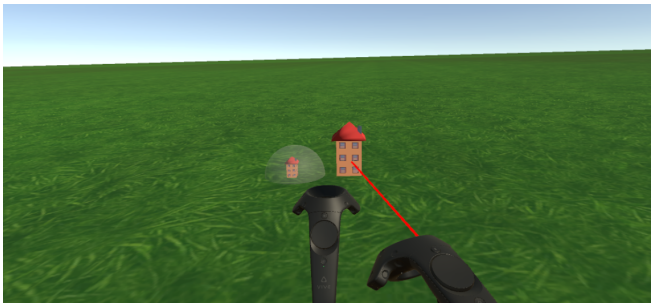

Figure 9.VR controller wrist-based palette selection for package (bubble) or class (building) creation in the terrestrial metaphor.

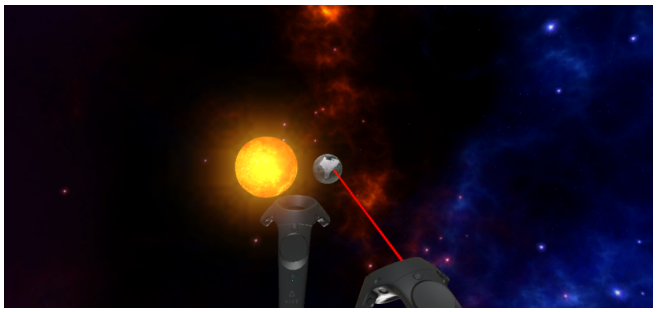

Figure 10. VR controller wrist-based palette selection for package or class creation in the universe metaphor.

Figure 11. Package creation (glass bubble) input screen.

Figure 12. Class creation (building) input screen.

Our case study subjects found the ability to interact and make changes to the software structure to be an improvement over pure visualization, and the use of VR controller wheel and pointer selection to be a natural in VR mode for creating and removing object. The accessibility of a real keyboard instead of a virtual keyboard via MR integration was also found to be more natural and efficient versus virtual keyboard usage for code editing.

Our prototype demonstrated that the MR solution concept is feasible and can be a viable alternative to a virtual keyboard. This enables touch typing and the use of the mouse for screen interaction where appropriate, enabling programmers to interact more naturally for their code-centric programming tasks in the VR environment, without having to interrupt their VR experience to take of the goggles and do programming changes, and then put the VR gear on again. In future work we will investigate empirical usage of our MR-FTC concept prototype including programmers who are not familiar with VR.

VI. CONCLUSION

As VR devices become ubiquitous, it is only a matter of time before programmers wish to utilize VR capabilities as well. Visualization of code structures in various metaphors permits VR users to view code structures in new ways. However, current interaction mechanisms are hampered due to a lack of keyboard and mouse access for programmers working directly with code, which can frustrate programmers or require them to remove the VR headset and work with a PC directly, and then return to VR mode.

This paper described our solution concept MR-FTC, which provides a direct integration via mixed reality of the keyboard and mouse into the VR landscape when the user looks down, allowing them to orient their hands and fingers after laying down the VR controllers. The prototype demonstrated the feasibility of our solution concept, and the case study with its empirical evaluation showed the potential of MR using keyboards for supporting programming in VR environments.

Future work includes a comprehensive empirical study and the inclusion of additional features specific to the programming interface such as syntax highlighting.

ACKNOWLEDGMENT

The author thanks Carsten Lecon, Dominik Bergen, Alexandre Matic, Lisa Philipp, and Camil Pogolski for their assistance with various aspects of the design, implementation, and evaluation.

REFERENCES

[1] C. Metz, *Google Is 2 Billion Lines of Code—And It's All in One Place*. http://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/ [retrieved 2017.08.31]

[2] GitHub. https://github.com [retrieved 2017.08.31]

[3] G. Booch, "The complexity of programming models," Keynote talk at AOSD 2005, Chicago, IL, Mar. 14-18, 2005.

[4] PayScale, Full List of Most and Least Loyal Employees. http://www.payscale.com/data-packages/employee-loyalty/full-list [retrieved 2017.08.31]

[5] F. P. Brooks, Jr., The Mythical Man-Month. Boston, MA: Addison-Wesley Longman Publ. Co., Inc., 1995.

[6] R. Oberhauser and C. Lecon, "Virtual Reality Flythrough of Program Code Structures," Proc. of the 19th ACM Virtual Reality International Conference (VRIC 2017). ACM, 2017.

[7] R. Oberhauser and C. Lecon, "Immersed in Software Structures: A Virtual Reality Approach," Proc. of the Tenth International Conference on Advances in Computer-Human Interactions (ACHI 2017). IARIA, 2017, pp. 181-186, ISBN 978-1-61208-538-8.

[8] A. R. Teyseyre and M. R. Campo, "An overview of 3D software visualization," Visualization and Computer Graphics, IEEE Trans. on, vol. 15, no. 1, 2009, pp. 87-105.

[9] A. Kashcha. *Software Galaxies*. http://github.com/anvaka/pm/ [retrieved 2017.08.31]

[10] R. Wettel and M. Lanza, "Program comprehension through software habitability," in Proc. 15th IEEE Int'l Conf. on Program Comprehension, IEEE CS, 2007, pp. 231–240.

[11] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in Proc. of the 33rd Int'l Conf. on Software Engineering, ACM, 2011, pp. 551-560.

[12] J. Rilling and S. P. Mudur, "On the use of metaballs to visually map source code structures and analysis results onto 3d space," in Proc.. 9th Work. Conf. on Reverse Engineering, IEEE, 2002, pp. 299-308.

[13] P. M. McIntosh, "X3D-UML: user-centred design, implementation and evaluation of 3D UML using X3D," Ph.D. dissertation, RMIT University, 2009.

[14] J. I. Maletic, J. Leigh, and A. Marcus, "Visualizing software in an immersive virtual reality environment," 23rd Intl. Conf. on Softw. Eng. (ICSE 2001) Vol. 1, IEEE, 2001, pp. 12-13.

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

[16] G.A. Lee, D. Nelles, M. Billinghurst, and G.J.Kim, "Immersive authoring of tangible augmented reality applications," Proc. of the 3rd IEEE/ACM international Symposium on Mixed and Augmented Reality, IEEE Computer Society, 2004, pp. 172-181.

[17] M. Billinghurst and H. Kato, "Collaborative mixed reality," Proc. First International Symposium on Mixed Reality (ISMR '99), Springer Verlag, 1999, pp. 261-284.

[18] T. Kamada and S. Kawai, "An algorithm for drawing general undirected graphs," Information processing letters, 31(1), 1989, pp. 7-15.

[19] J. Maletic, M. Collard, and A. Marcus, "Source code files as structured documents," in Proc. 10th Int. Workshop on Program Comprehension, IEEE, 2002, pp. 289-292.

 ISBN: 978-1-61208-590-6