Towards Virtual Reality Immersion in Software Structures: Exploring Augmented Virtuality and Speech Recognition Interfaces

Roy Oberhauser and Carsten Lecon
Computer Science Dept.
Aalen University
Aalen, Germany
email: {roy.oberhauser, carsten.lecon}@hs-aalen.de

Abstract - **Due to its abstract nature, program code structures have been inherently challenging to visualize. As virtual reality (VR) hardware products become common, their utilization for providing insights into these software structures become feasible. However, for certain user programmer-centric interaction scenarios, typical VR interfaces (controller held in each hand) can be awkward. This paper describes our VR and mixed reality (MR) fly-through software structure approach for visualizing internal program code structures and investigates additional interfaces to augment the virtuality. MR use of any keyboard and mouse are supported for programming tasks. To interface with a virtual tablet menu that is used as an oracle, a real tablet is used as a touchpad for the VR tablet. Voice control of the tablet menu was also implemented and investigated in comparison to the other interfaces. The paper evaluates the various VR and MR interfaces for their suitability for selected software development and computer science educational tasks. The evaluation results provide insight into which interfaces were more efficient and preferred by subjects.**

Keywords - Virtual reality; mixed reality; augmented virtuality; software visualization; program comprehension; software engineering; speech recognition; voice control.

I. INTRODUCTION

This paper extends [1], where we described a VR approach for visualizing, navigating, and conveying program code information interactively in a VR environment called VR-FlyThruCode (VR-FTC).

The volume of program source code produced and maintained worldwide continues to increase, yet gauging this metric is difficult since large portions are not publicly available. Google alone is estimated to have at least 2bn lines of code (LOC) internally accessible by 25K developers [2]. By some estimates well over a trillion lines of code (LOC) exist worldwide with 33bn added annually [3]. The limitations for humans to comprehend source code are evident in the relatively low code review reading rates of around 200 LOC/hour [4].

Faced with this ever-increasing code base, the question becomes: how can programmers quickly comprehend large amounts of code and understand their underlying and mostly invisible abstract structures? Common display forms used in the comprehension of source code include text, the two-dimensional Unified Modeling Language (UML), and software analysis tools. For large projects, typically multiple UML diagrams exist that are not coherently tied together and no screen can support showing all diagrams simultaneously, thus a mental patchwork of the multiple visual images is used to create a coherent model in the programmer's mind. For code files, typically these are hierarchically hidden among various subdirectories, and determining relations is an arduous task. These forms of extraction leave the programmer creating and stitching explicit or implicit visual images together.

One software pioneer, F. P. Brooks, Jr., asserted that the invisibility of software is an essential difficulty of software construction because the reality of software is not embedded in space [5]. Yet the philosopher Aristotle once stated, "thought is impossible without an image."

In 1998, Feijs and De Jong [6] presented a vision of walking through a 3D visualization of software architecture using VRML. Currently the immersive potential of VR and game engines for improving software engineering (SE) tools has still not been realized, and their practicality with off-the-shelf VR hardware remains insufficiently explored.

This paper describes our visually immersive VR approach for visualizing, navigating, and conveying program code information interactively to support exploratory, analytical, and descriptive cognitive processes [7]. In extending [1], it contributes additional interface capabilities to the VR-FTC solution concept and investigates their suitability. Specifically, the following interfaces: speech recognition for voice-directed control of the oracle – our VR voice FTC (VRVoc-FTC), as well as an MR tablet that was added to the to the MR-FTC variant to control the oracle like a touchpad. Furthermore, the fly-in theaters were replaced with an always-accessible virtual tablet which functions as an oracle – an interactive screen supplying information on request to the user. Also, as a form of augmented virtuality [8], MR support for real keyboard and mouse interfacing were added to support programming on the tablet [9], henceforth known as the MR-FTC variant. A prototype realization demonstrates the viability of these capabilities, and initial empirical experiments investigate effectiveness, efficiency, and user experience (UX) factors of the various interfaces for programming and menu navigation tasks.

The paper is organized as follows: the next section discusses related work; Section III then describes the solution approach. Section IV provides details on our prototype realization. Section V describes the evaluation of prototype and the alternative interfaces from a technical or empirical perspective. It is followed in Section VI by a conclusion.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

II. Related Work

As to voice interfaces in SE, work related to voice control of SE tools includes Delimarschi et al. [10], who applied voice and gesture control to IDE tools. Lahtinen and Peltonen [11] investigated voice control for UML modeling tasks.

As to utilizing tablets in VR, Afonso et al. [12] tracked hand movement to determine if people preferred to see a virtual hand on a virtual tablet while holding a real tablet in VR. In our case, the tablet is not necessarily held (it is placed on the desk like the keyboard and mouse in a known position), we do not track hand movement, and we do not show a hand but do show a small box indicator where the tablet was last touched.

Work on visualization of software structures in VR includes Imsovision [13], which visualizes object-oriented software in VR using electromagnetic sensors attached to shutter glasses and a wand for interaction. ExplorViz [14] 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.

Work regarding software visualization without the use of VR includes Teyseyre and Campo [15], who give an overview and survey of 3D software visualization tools across the various software engineering areas. Software Galaxies [16] provides a web-based visualization of dependencies among popular package managers and supports flying. Every star represents a package that is clustered by dependencies. CodeCity [17] is a 3D software visualization approach based on a city metaphor and implemented in SmallTalk on the Moose reengineering framework. Buildings represent classes, districts represent packages, and visible properties depict selected metrics, improving task correctness but slowing task completion time [18]. Rilling and Mudur [19] use a metaball metaphor (organic-like n-dimensional objects) combined with dynamic analysis of program execution. X3D-UML [20] provides 3D support with UML in planes such that classes are grouped in planes based on the package or hierarchical state machine diagrams. A case study of a 3D UML tool using Google SketchUp showed that a 3D perspective improved model comprehension and was found to be intuitive [21]. Langelier et al. [22] supports the visualization of metrics (e.g., coupling, test coverage).

In contrast to the above work, the VR-FTC approach and its variants (MR-FTC and VRVoc-FTC) leverage game engine capabilities to support an immersive VR software structure visualization environment; provide multiple dynamically-switchable (customizable) metaphors; use one VR system and controller set (without requiring gesture training) for interaction and navigation; uses a virtual tablet to provide an information screen within the VR landscape; leverages MR to support keyboard, mouse, and tablet interfaces in VR; and provides a voice direction capability to control menu options.

III. Solution Approach

As described in [1], our VR-FTC solution approach uses VR flythrough for visualizing program code structure or architecture. This inherent 3D application domain view visualization [15] arranges customizable symbols in 3D space to enable users to navigate through an alternative perspective on these often-hidden structures. For example, certain information typically 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 dependencies between classes and packages.

A. Principles

The principles (P:), (basic ideas or primary methods) involved in the VR-FTC solution approach include:

P:Multiple 3D visual metaphors: Analogous to the concept of skins, it models and supports tailoring and switching between multiple code structure visualization metaphors. While our initial implementation focused on modeling and visualizing object-oriented packages, classes, and their relationships, the approach is extensible for other programming languages. Initially, two metaphors are provided "out-of-the-box" while custom mappings to other object types are supported. In the universe metaphor, each planet represents a class with its size based on the number of methods, and solar systems represent a package. Any metric can be used to map to any visual object property (like color). Multiple packages are shown by layer solar systems over one another. In the terrestrial metaphor, buildings can represent classes, building height can represent the number of methods, and glass bubbles can group classes into packages. Relationships are modeled visually as colored pipes.

P:Group metaphor: elements (classes) are grouped and delineated in a way appropriate for that language (packages for Java) and metaphor. For instance, the terrestrial metaphor uses either a glass bubble over a city or a circle of trees at the city border, and the universe metaphor uses solar systems.

P:Connection metaphor: elements (classes) are connected in a way appropriate for that metaphor. For our two metaphors, we chose colored light beams, which often are used to portray networks on a geological background.

P:Flythrough navigation: 3D navigation (motion) is provided by moving the camera in space based on controller or motion sensor input. The scenery, however, remains anchored in the scene, allowing users to remember places via their geolocation relative to other elements.

P:Oracle: a virtual tablet is provided, and can be viewed as a type of oracle to answer questions a user might have, although these cannot be formulated directly like a chatbot. Instead, it provides menus and displays source-code and SE tool-generated information as selected by the user within their given context (such as a selected object). The screens currently presented include:

- *Tags*: Setting, searching, or filtering automatic (via patterns) or manual persistent annotations/tags.
- *Source Code*: code is shown in scrollable form.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

- *UML*: 2D UML diagrams can be shown on the tablet, including dynamically generated 2D diagrams, allowing users to leverage knowledge of this form if already available or if dynamic generation thereof is desired.
- *Metrics*: code metrics are displayed textually due to the large number of possible metrics that may be of interest to the user; displaying a large amount of metric information visually may be disconcerting. Customization enables metrics of interest to be utilized in a metaphor (e.g., colors, object height can relate to number of class methods, font colors can indicate a threshold is exceeded).
- *Filtering*: shows elements that match selectors.
- *Project*: change metaphors, load, or import a project.

P:MR interfaces: in addition to the VR controllers and a virtual keyboard and a virtual tablet, use of a real keyboard, mouse, and tablet used as a touch pad are supported in the MR-FTC variant.

P:Voice interface: Voice control of the tablet menu is supported in the VRVoc-FTC variant using speech recognition.

B. Process

Five steps are involved in the solution process, consisting of:

1) Modeling: modeling generic program code structures, metrics, and artifacts as well as visual objects. More details on what models and formats are used in our prototype are given in Section IV.

2) Mapping: mapping a model to a visual object metaphor.

3) Extraction: extracting a given project's 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 VR model instance (via camera movement based on user interaction) and navigation of the oracle menu.

IV. Implementation

For the implementation, we utilized the Unity 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 and two wireless handheld controllers tracked using two 'Lighthouse' base stations. First, we reiterate implementation details based on [1] and then in Sections E, F, and G we provide descriptions of the new interface implementations.

A. Architecture

Figure 1 shows the 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 [23] which we use for placing and spacing objects within a metaphor. 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.

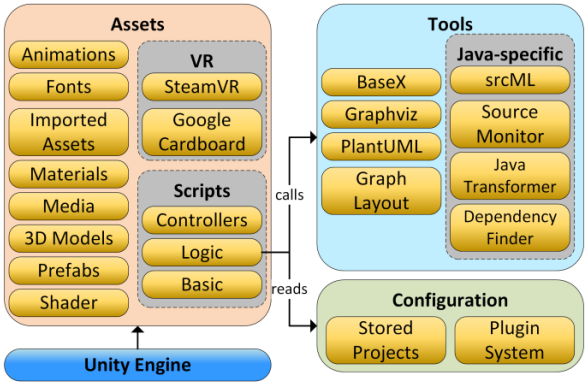

Figure 1. VR-FlyThruCode software architecture.

B. Information Extraction

For extracting existing code structure information into our model, srcML [24] is used to convert source code into XML that is then stored in the XML database BaseX, 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.

C. Project structure

For an imported project the following files are created:

- *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.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

D. Metaphor Realization

To support *P:Multiple 3D visual metaphors*, a universe and a city metaphor were chosen since these are universally known and can be easily related to by users, however other metaphors are easily realizable. A welcome room similar to a cockpit, shown in Figure 2, enables the user to select the desired metaphor and project state. Figure 3 shows the city metaphor, where buildings represent classes with a label at the top and the height can portray a metric of interest such as the number of methods in that class. In the City metaphor, *P:Group* was implemented as a glass bubble over the city as shown in Figure 4. In the universe metaphor, planets represent classes and have a label in the center as shown Figure 5. *P:Group* was implemented as solar systems (see Figure 6 and compare with Figure 7). For *P:Connection*, in both metaphors colored light beams were used to show dependencies between classes or packages (see Figure 8 and Figure 9) To highlight a selected object, we utilized a 3D pointer in the form of a rotating upside-down pyramid. Graph Layout was used for placement of the visual objects, which is done automatically by the system.

To allow the user to remember objects, tagging is supported, which allows any text label to be entered and placed on an object (e.g., the 'Important' Tag in Figure 5).

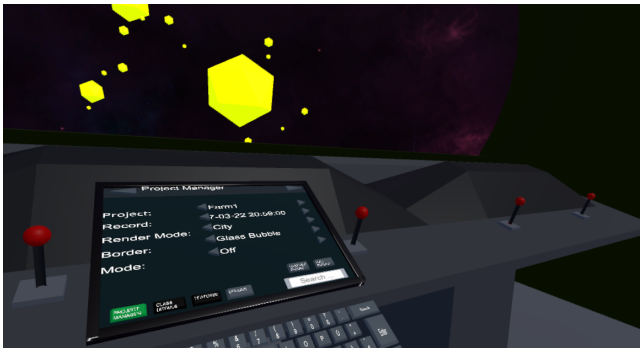

Figure 2. VR-FlyThruCode software architecture.

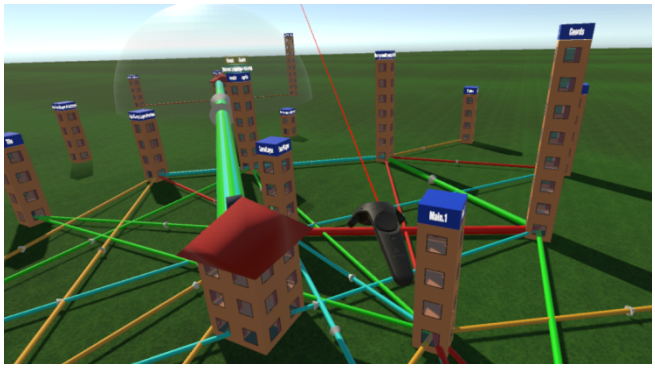

Figure 3. The city metaphor, with buildings as classes and the VR controller visible.

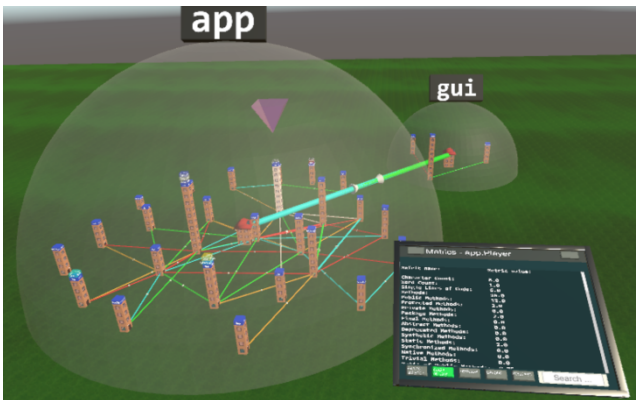

Figure 4. City metaphor showing glass bubbles with oracle visible.

Figure 5. Universe metaphor showing tagged planet (class) and oracle.

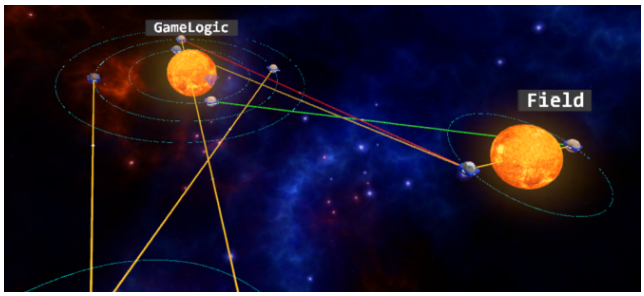

Figure 6. Universe metaphor showing solar systems.

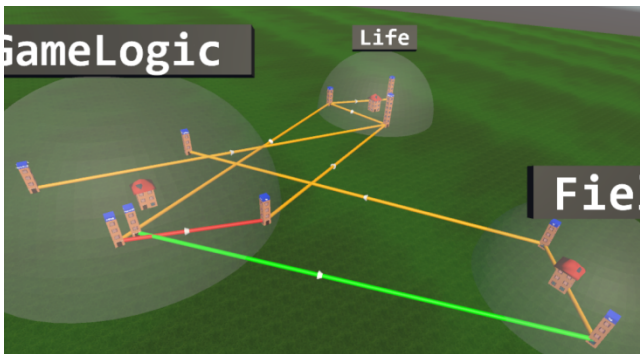

Figure 7. City metaphor showing glass bubbles.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

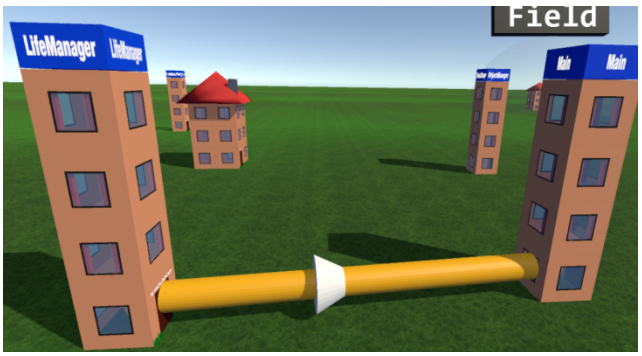

Figure 8. A directed dependency in the City metaphor.

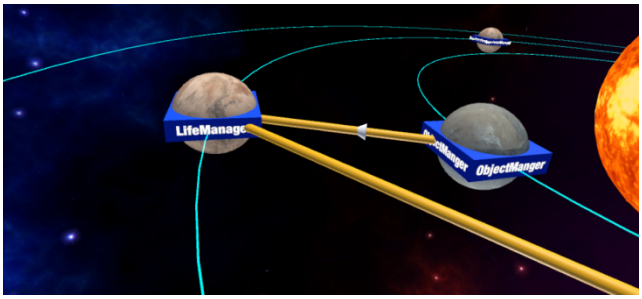

Figure 9. A directed dependency in the Universe metaphor.

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 to a screen or menu one may turn around and lose track of where the object was, especially if the object was small relative to its surrounding objects.

E. Virtual Reality Interaction

On the HTC Vive the touchpad on the left controller controls altitude (up, down) and the one on the right hand the direction (left, right, forward, backward), which realizes *P:Flythrough* navigation by moving the camera position. The controllers are shown in the scenery when they are within the view field, as shown in Figure 10. A virtual laser pointer was created for selecting objects, as was a virtual keyboard (see Figure 10) to support text input for searching, filtering, and tagging. To implement *P:Oracle*, menus and screens showing source code, code metrics, UML diagrams, tags, filtering, and project data are accessible via a virtual tablet.

Figure 10. VR controller using the virtual keyboard.

F. Mixed Reality Keyboard, Mouse, and Tablet

To implement *P:MR interfaces* in the MR-FTC variant, access to a real keyboard and mouse in MR was achieved via a live camera view, which was integrated into the VR landscape using a virtual plane object (see Figure 11). A Logitech C920 webcam with a 1080p resolution was used instead of the Vive Front camera to achieve better resolution. With this option, the user can utilize their favorite keyboard and mouse that they are already accustomed to.

Figure 11. MR-FTC variant with a MR keyboard (mouse out of view).

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.

Figure 12. Android tablet use as augmented virtuality.

To support a tablet, we created an Android app with no visible user interface (it only needs to detect the finger location as shown in Figure 12) and tested it on a Sony XPERIA Z2 Tablet with Android 6.0.1. When a user touches the App screen, a UDP packet consisting of the finger location coordinates and a tap event flag to our MR-FTC Unity application on the PC via the wireless network. A mouse pointer in the form of a white cube is then shown on the oracle (virtual tablet) at the equivalent position. A double-tap results in an OnClick-Event. Figure 12 shows a user holding the tablet and "seeing" the location on the

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

virtual tablet. Figure 13 shows a closeup of the oracle and shows the cube indicator of the finger location. Note that no orientation information is sent (e.g., tilt angle) and that for the experiment the tablet was fixed to the desktop in a known location for the user, so they do not need to hold it. The app is blank except for displaying the coordinates because the user in VR does not see the real tablet; it is not under the camera as the MR keyboard and mouse but rather in a fixed position on the desk known to the user before putting the VR goggles on. A Sony XPERIA XZ with Android 8.0.0 was also tested.

Figure 13. Closeup of the oracle showing menus and the source code view.

G. Voice Directed Control

To implement a *P:Voice interface* in our VRVoc-FTC variant - specifically voice-directed control of the oracle menu, we evaluated various openly available speech recognition options. Our constraints were that we not be required to pay for any service, which constrained certain well-known cloud options. We then evaluated various popular libraries but our requirements were that little to no training would be required. After various attempts that did not result in satisfying solutions, we settled on Windows System Speech (WSS) and Unity Speech (Cortana). For the evaluation, only WSS was used since network access to Cortana was blocked by campus IT. Off-campus, Windows 10 Pro (with Fall Creators Update) was tested with Cortana.

Because WSS support is not integrated in Unity 5.6, we used a client-server program to send the commands from WSS to Unity, where it is processed by the SpeechHandler class. Thus, any Speech API (Application Programming Interface) could send the commands over TCP/IP to VR-FTC.

The microphone was mounted on a headset, so it is close to the mouth and thus reduces unrelated noise inputs. Currently no indication is given if a command is not understood, but because the response is normal fast, after a second the user should notice that the command was not executed and will likely try again.

The following words are supported at this time:

"class information", "class details": opens the view "Class Details"

"source code", "source": shows the source code for the selected class

"project manager", "project": opens the view "Project Manager"

"feature screen", "feature": opens the view "Feature Screen"

"option screen", "options screen", "option", "options": opens the configuration options view

"left", "previous": changes the view to the previous window

"right", "next": changes the view to the next window

Under Unity Cortana, we also support "search <classname>", which shows the program code of that class. In WSS this functionality was not possible since only predefined words can be used, and its free speech recognition mode performance was unusable for SE-specific tasks.

WSS does not improve automatically over time. While one can manually improve WSS, we noticed no improvements after a half hour session. Unity Speech (Cortana) probably improves automatically but we were unaware of an option to manually improve it via sessions.

V. Evaluation

After showing the feasibility of the solution with our implementation, our technical evaluation focused on assessing the implementation's viability on current VR hardware options. To compare our VR-FTC solution's suitability, effectiveness, and efficiency with non-VR and provide an overall picture, empirical evaluations were performed as indicated below. Section D includes new evaluations focused on the interfaces for programming-centric tasks, where we performed an empirical evaluation of MR-FTC with a keyboard and mouse input compared to non-VR and a virtual keyboard. And to evaluate menu interfaces of the oracle (virtual tablet), we performed an empirical study comparing VR controllers, a MR touchpad-like tablet interface, and voice control.

A. Technical Evaluation

Our technical evaluation performed in [1] utilized an HTC Vive with a 2160×1200 447 PPI resolution, Unity 5.3.5f1 PE, SteamVR 1479163853. The desktop PC had a 4GHz i7-6700K, 32GB RAM, SSD, NVIDIA GeForce GTX980Ti with 6GB GDDR5, Win7 Pro x64 SP1. The notebook was a MSI GS60 2.5GHz i7-4710HQ, 16GB RAM, NVIDIA GeForce GTX870M with 3GB GDDR5, SSD, Win10 Home x64, which did not meet Vive's minimum requirements but allows us to determine if a notebook (popular among software developers) would suffice for our VR application.

1) Resource usage: RAM was allocated for a 64-bit implementation was 220MB (with no project), 250MB (project with 27 classes), and 620MB (project with 95 classes). On the notebook, graphics card load was 80% without a project and went to 90% with a loaded project (for the PC 20%). We determined the CPU was the bottleneck, with load on the PC for a large project almost always at 100%. We believe that scripts attached to each visible class invoke their update method for each frame, and plan to optimize this in future work.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

1) Frame rate: to determine the performance impact of each metaphor and if a notebook would be sufficient, the Saxon XSLT 2.0 and XQuery processor consisting of 300K lines of code and 1635 classes was loaded and the frames per second (FPS) measured via a custom script.

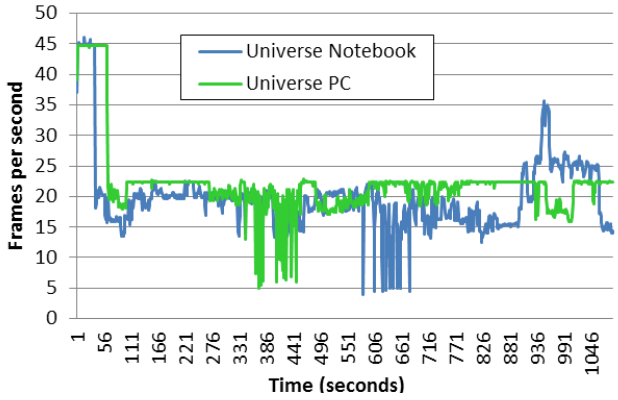

Figure 14. Universe metaphor frame rate over time on notebook and PC.

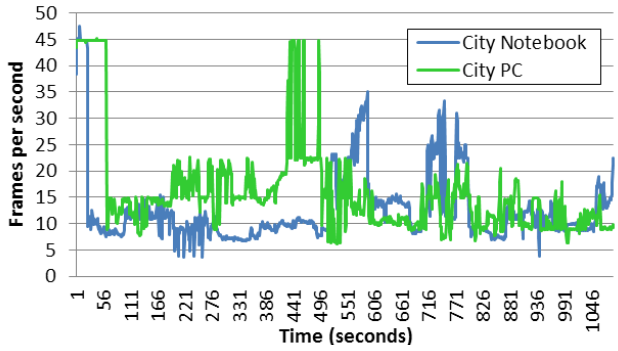

Figure 15. City metaphor frame rate over time on notebook and PC.

Figure 14 and Figure 15 show that the notebook mostly exhibited lower FPS rates than the PC, and that the city metaphor lowered the FPS rate. This is also shown by the average FPS rates for universe (notebook=20.0; PC=22.2) and city (notebook=12.7; PC=16.0). Below 15 FPS is not tolerable (early silent films had 16-20). An initial analysis found the dynamic UML generator - run in a separate process - as a main cause, and this will be addressed in future work. The universe ran better than city, since city included multiple shadows, reflections from the glass bubble, and a terrain. Higher FPS occurred when flying to an outer package such that far fewer objects were in view.

B. *Suitability of VR for SE Tasks*

Our empirical evaluation to compare the SE task suitability of VR vs. non-VR was performed in [1] using the HTC Vive. Our hypotheses were (1) that VR mode is on par with non-VR in effectiveness and efficiency for SE code structure analysis tasks and education, and (2) VR mode offers an immersive and UX quality absent in non-VR.

Resource-constraints such as having only one Vive and the time-intensive 2-on-1 supervision of the experiment with a single subject at a time limited our sample size. A convenience sample of 10 computer science students of various academic semesters (1; 3-4; 6-9 grouped respectively as beginner, intermediate, and advanced) participated and self-rated their programming and UML competency (Figure 16). Object-orientation (OO) is taught in the second and UML in the fourth semester. The one first semester student had work experience in the software industry and thus knew OO and UML. Each received a short tutorial on non-VR FTC (three had prior experience). Project A consisted of 2 packages, 27 classes, and 170 methods, while Project B had 5 packages, 95 classes, and 800 methods.

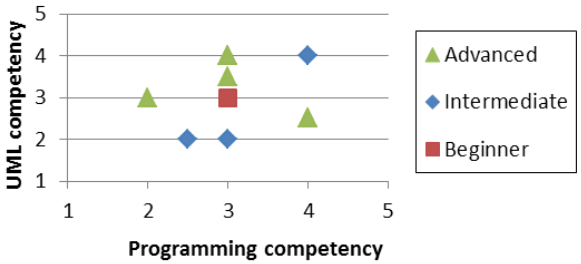

Figure 16. Participant UML/programming self-rating by semester level.

In non-VR mode, project A was loaded in the universe and thereafter the city metaphor, and likewise with B, and the same sequence repeated for VR mode. 8 questions were asked per case dealing with program code structural comprehension requiring navigation (not the same set each time), resulting in 64 questions (see Figure 17); 5 additional general questions followed giving 69 in total. So that the VR glasses need not be removed, and in order not to skew the task durations in non-VR mode, questions were asked and answered verbally and noted by a supervisor.

1) How many connections/dependencies does class X have within the package Foo?
2) How many connections/dependencies does class Y have within the package Bar?
3) Add a tag to the class X
4) Which package is the largest/smallest?
5) How many connections/dependencies does package Foo have?
6) How many connections/dependencies does package Bar have?
7) How many variables are declared in the class Y in package Foo?
8) Which classes are directly connected with the class Y?
9) Name all classes on the shortest path from A to B.
10) How many overloaded functions does the class Z have in package Bar?
11) In what package did you set your tag?

Figure 17. Sample timed task questions and requests.

As to efficiency, on average 92.5 min were needed for the 64 questions, 43.4 in VR mode vs. 39.5 min in non-VR (10% difference), while VR training took 9.4 min. Figure 18 shows the sum of the task durations for each mode per subject, whereby subjects 8-10 had prior FTC familiarity. Although VR mode was 10% slower, this was their first

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

experience using VR. In addition, in non-VR mode the HUD is instantly available and screens can be switched, while in VR mode navigation to a screen is required. In our opinion, more VR practice might reduce this difference further.

With regard to effectiveness, given 32 questions in each mode across 10 subjects, in non-VR 300 (94%) and in VR 296 (93%) were answered correctly.

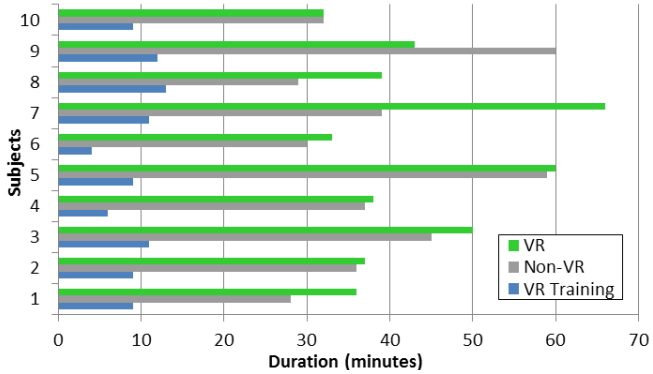

Figure 18. Sum of task durations per subject for VR and non-VR modes.

Subjects considered both FTC application modes suitable for these SE tasks. Comments included liking how information was visually displayed, its closeness to a reality, its clear arrangement, and that head movement could be used for exploring (which non-VR cannot provide). Subjects felt no differently after using non-VR, whereas after VR the feeling was described as impressive for seven of the ten subjects. The other three subjects reported VR sickness symptoms, a type of visually-induced motion sickness exhibiting disorientation. We plan to address the VR sickness in future work, e.g., by increasing the frame rate via optimizations and reducing the speed of camera movement.

C. Mixed-Reality Interface for SE Tasks

An empirical evaluation of the MR-FTC keyboard capability using a convenience sample consisting of Computer Science students was performed as described in [9]. For evaluating typing speed in particular, for program text such as 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 (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

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.

D. Voice, Tablet, and Controller Interface Comparison

To compare menu-centric control of the oracle with the VR-FTC, MR-FTC tablet, and VRVoc-FTC variants empirically, we used a convenience sample of six Computer Science students. During the experiment, one of the subjects exhibited VR sickness symptoms and could not continue, so the results for this student were removed. In future work, we will attempt various optimizations and see if this reduces the likelihood. After the supervised treatments, the subjects filled out a questionnaire and were debriefed.

To ascertain efficiency effects of the different interfaces, each subject was given five different SE tasks by a supervisor to perform in the VR-Based VR-FTC City metaphor with each interface, such as find a certain class, determine how many methods a certain class has, tag a class, add a comment. Similar tasks were given for each case of interface when using primarily Voice control (V), Tablet control (T), or VR Controller (C). A random order of

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

treatments was applied in order to ascertain if the treatment ordering created a training effect (e.g., always faster with the second or third interface), and the order is depicted as labels in Figure 19 which shows the total task duration by subject for each interface. As can be seen, the ordering did not show a clear trend. On average, voice took 353 seconds, the tablet 346 seconds, and the VR controller 197 seconds. we approximate that the tablet and voice are similar in efficiency and averaged together (350s) they are 77% slower than controller use.

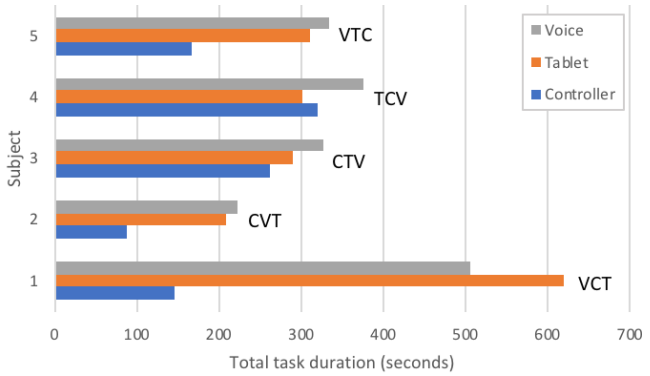

Figure 19. VR-FTC total task duration in seconds by subject for voice, tablet, and controller interfaces.

Figure 20 shows the results of the subjective assessment of suitability and enjoyment by the subjects. Overall one observes that the VR controllers had the most positive suitability and enjoyment ratings, and that voice was had three positive assessments for suitability and enjoyment. The tablet had the least positive suitability and enjoyment assessment.

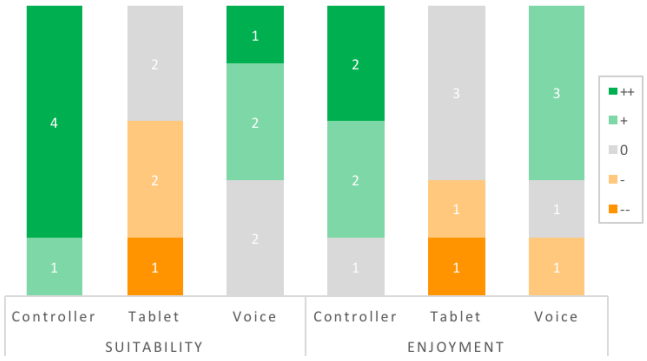

Figure 20. Suitability and enjoyment assessments in VR-FTC for voice, tablet, and controller interfaces from very high (++) to very low (--).

The supervisor gave subjects their tasks via speech, and subjects also spoke with supervisor during their tasks, which caused unintended commands to be executed. Also, the university lab setting included various other students and background noise, which reflect a realistic setting for software developers. From approximately 50 voice commands that were required to perform the SE tasks, the supervisor noted that less than 1 was not heard and less than 1 misinterpreted. For instance, if the subject was naming the functions the speech interpreter might interpret a command based on the name of a function or class. To address this, in future work we plan to provide a clear delineation for voice command mode (e.g., push to talk on a controller) and to inject the experiment directions into the scenery.

We were surprised that the MR-FTC tablet was not found to be that suitable or enjoyable for controlling the virtual tablet. We thought it would be found to be similar to a touch pad on a notebook, some users thought it was a good idea and more stable than holding the controllers.

Voice was the slowest overall. Voice direction almost always requires more time than direct control (e.g., keyboard or mouse on a PC vs. voice control), however it can free up other interfaces, and this was recognized as a benefit in the debriefing by a number of subjects.

E. Discussion

The technical evaluation of Section V.A showed suitable resource usage but pointed out frame rate issues. As to the suitability of using VR-FTC for SE tasks such as answering structural issues like those in Figure 17, Section V.B showed that while VR-FTC was 10% slower on average for untrained VR users, no significant difference in correctness were observed. Thus, our empirical hypotheses were confirmed by our results and the feedback from participants.

One threat to validity is the order effect of application usage in that non-VR followed VR. Thus, non-VR times include the overhead for gaining familiarity with the application concepts, and VR mode did not have this overhead. However, 2D monitor and mouse-centric interaction was a pre-existing competency, while VR display and navigation was a new interaction paradigm for all subjects. Furthermore, subjects 8, 9, and 10 had prior familiarity with the non-VR FTC via a prior experiment, yet their task duration times did not exhibit any clear trend that prior familiarity sped up the non-VR task durations. Furthermore, the 1% difference in correctness might be attributed to mental fatigue since VR was done in the second hour. A further threat to validity is that the positive experience is possibly a novelty effect - VR veterans would be needed to be included to assess this factor. For better external validity, the sample size should be larger and more diverse to include professionals. However, the results can be viewed as indicative and the approach as promising if we can address the VR sickness. We made optimizations for the frame rate issues to address VR sickness in further empirical studies, and only one person in those experiments experienced VR sickness.

With regard to the results in Section V.C of using FTC with a keyboard, a non-VR text editor remains more efficient, yet usage of the MR-FTC keyboard was faster than a purely VR-FTC 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.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

The Section V.D results regarding interfaces for oracle menu control found that the use of the VR controllers was most efficient, suitable, and enjoyable. Since VR controllers are specifically intended for interacting in VR, this result is not surprising. However, our investigation showed that the impacts of alternative interfaces may still be acceptable for certain users, and this efficiency impact is not on the order of magnitudes in scale. VRVoc-FTC indicated that it has potential, with no subject indicating it was not suitable. MR-FTC with a real tablet was not found to be suitable or enjoyable and was about as slow as VRVoc-FTC, which we found surprising since the virtual tablet is shown in VR and one would think it would be enjoyable to hold one while in VR. In future work we will investigate potential improvements to its interface and removing all need for having VR controllers when using the tablet. One threat to validity is the small sample sizes, yet it does provide some indicator as to which interfaces to pursue and investigate further and consider for industrial usage studies.

VI. Conclusion

As VR devices become more commonplace, the potential of VR to assist programmers in program comprehension can provide an immersive alternative to commonly available tools and paradigms. This paper described our VR flythrough software structure visualization approach called VR-FTC. As augmented virtuality we explored alternative interfaces to the VR controllers including MR-FTC (keyboard and tablet) and VRVoc-FTC (voice) variants. It immerses users into multiple and customizable VR metaphors for visualizing, navigating, conveying, and changing program code information interactively to support exploratory, analytical, and descriptive cognitive processes.

Our investigation observed that when comparing SE tasks in VR to non-VR, non-VR (with which the subjects are quite familiar) was more efficient. However, given more VR experience and training these differences could become smaller, and the VR efficiency overhead may be justified by the better and more enjoyable and motivational experience for users. In exploring alternative interfaces in VR, for text input we found that MR-FTC using keyboard and mouse was a viable option and faster than a virtual keyboard. For menu navigation, we found that VR controllers were most efficient and that voice, although less efficient, was an acceptable alternative option. A real tablet interface equivalent to a touchpad was not found to be suitable or enjoyable and was equivalent to voice in efficiency. However, in future work we intend to turn the tablet into a complete replacement for the VR controllers and reevaluate its suitability.

Future work includes further analysis and optimizations to address any remaining VR sickness symptoms, and comprehensive empirical studies in the industry.

Acknowledgment

The authors would like to thank Dominik Bergen, Sinan Emecan, Alexandre Matic, Lisa Philipp, and Camil Pogolski, and Patrick Sprenger for their assistance with various aspects of the design, implementation, and evaluation.

References

[1] 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.

[2] C. Metz, *Google Is 2 Billion Lines of Code—And It's All in One Place*. [retrieved: May, 2018]. Available from: http://www.wired.com/2015/09/google-2-billion-lines-codeand-one-place/

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

[4] C. F. Kemerer and M. C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on PSP data," IEEE Trans. on Software Engineering, vol. 35, no. 4, 2009, pp. 534-550.

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

[6] L. Feijs and R. De Jong, "3D visualization of software architectures," Comm. of the ACM, 41(12), 1998, pp. 73-78.

[7] D. M. Butler et al., "Visualization reference models," in Proc. Visualization '93 Conf., IEEE CS Press, 1993, pp. 337–342.

[8] P. Milgram, H. Takemura, A. Utsumi, and F. Kishino, "Augmented reality: A class of displays on the reality-virtuality continuum," In: Telemanipulator and telepresence technologies, Vol. 2351. International Society for Optics and Photonics, 1995, pp. 282-293.

[9] R. Oberhauser, "Immersive Coding: A Virtual and Mixed Reality Environment for Programmers," In: Proceedings of The Twelfth International Conference on Software Engineering Advances (ICSEA 2017), IARIA XPS Press, 2017, pp. 250-255.

[10] D. Delimarschi, G. Swartzendruber, and H. Kagdi, "Enabling integrated development environments with natural user interface interactions," Proceedings of the 22nd International Conference on Program Comprehension (ICPC 2014). ACM, 2014, pp. 126-129.

[11] S. Lahtinen and J. Peltonen, "Enhancing usability of UML CASE-tools with speech recognition," Proceedings of the 2003 IEEE Symposium on Human Centric Computing Languages and Environments. IEEE, 2003, pp. 227-235.

[12] L. Afonso, P. Dias, C. Ferreira and B. S. Santos, "Effect of hand-avatar in a selection task using a tablet as input device in an immersive virtual environment," 2017 IEEE Symposium on 3D User Interfaces (3DUI). IEEE, 2017, pp. 247-248.

[13] 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.

[14] 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.

[15] 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.

[16] A. Kashcha. *Software Galaxies* [retrieved: May, 2018]. Available: http://github.com/anvaka/pm/

[17] 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.

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

[19] 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.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org

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

[21] A. Krolovitsch and L. Nilsson, "3D Visualization for Model Comprehension: A Case Study Conducted at Ericsson AB," University of Gothenburg, Sweden, 2009.

[22] G. Langelier et al., "Visualization-based analysis of quality for large-scale software systems," in Proc. 20th Int. Conf. on Automated Software Engineering, ACM, 2005, pp. 214-223.

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

[24] J. Maletic et al, "Source code files as structured documents," in Proc. 10th Int. Workshop on Program Comprehension, IEEE, 2002, pp. 289-292.

2018, © Copyright by authors, Published under agreement with IARIA - www.iaria.org