# Virtual Reality Flythrough of Program Code Structures

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

## ABSTRACT
The abstract nature of program code structures is inherently challenging to visualize. As virtual reality (VR) hardware and software become mainstream, VR visualization of program code structures becomes feasible. This application paper describes and evaluates our VR approach for visualizing software code structures that creates an immersive fly-through experience with multiple metaphors. Results show metaphor frame rate differences while interface usability efficiency was equivalent to non-VR.

## Categories and Subject Descriptors
I.3.7 [**Information interfaces and presentation**]: Multimedia Information Systems – *Artificial, augmented, and virtual realities.*

## General Terms
Documentation, Design, Experimentation, Human Factors, Languages.

## Keywords
Virtual reality, software visualization, program comprehension, software engineering, engineering training, computer education.

## 1. INTRODUCTION
The rapid digitalization of society is inexorably linked with an increased demand for and dependence on software. Accordingly, the amount of program source code produced and maintained worldwide by software developers is increasing dramatically. At least 2bn lines of code (LOC) can be accessed by 25k developers at Google [1], and well over a trillion lines of code (LOC) are estimated to exist worldwide with at least 33bn added annually [2]. In one study, 11m professional software developers are estimated to be producing or maintaining program code, excluding hobby and other IT-skilled workers [3].

Given such a volume of code, these developers continue to struggle with comprehending unfamiliar complex code structures and dependencies utilizing common display forms of program source code or the two-dimensional Unified Modeling Language (UML). Reasons for this include cognitive limitations as well as the inherent invisibility of software, which remains an essential difficulty for software construction since the reality of software is not embedded in space [4]. A vision of walking through a 3D visualization of software architecture was described in [5], yet the potential of VR and game engines has not been fully realized in software engineering (SE) tools, and their practicality with off-the-shelf VR hardware remains insufficiently explored.

In prior work [6], we described our non-VR 3D FlyThruCode (FTC) approach for visualizing software structures. This application paper describes various features of our VR-based FlyThruCode (VR-FTC) approach for visualizing, navigating, and conveying program code information interactively in a VR environment to support exploratory, analytical, and descriptive cognitive processes [7]. A prototype demonstrates its viability, and a technical evaluation investigates its resource usage and performance, while an empirical study investigates VR vs. non-VR interface efficiency for SE tasks.

The paper is organized as follows: the following section discusses related work; Section 3 then describes our solution. In Section 4 we provide implementation details. Our evaluation is described in Section 5, and is followed by a conclusion.

## 2. RELATED WORK
[8] provides an overview and survey of 3D software visualization tools across the various software engineering areas. Software Galaxies [9] gives a web-based visualization of dependencies among popular package managers and supports flying, with each star representing a package clustered by dependencies. CodeCity [10] 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, with [11] showing a significant improvement in task correctness and task completion time. X3D-UML [12] provides 3D support with UML in planes such that classes are grouped in planes based on the package or hierarchical state machine diagrams. As to VR approaches, Imsovision [13] visualizes object-oriented software using electromagnetic sensors attached to shutter glasses and a wand for interaction. ExplorViz [14] is a browser-based web application that uses Javascript-based WebVR to support VR exploration of 3D software cities using Oculus Rift together with Microsoft Kinect for gesture recognition.

In contrast, VR-FTC visualizes software structures by leveraging common game engine VR capabilities and a single VR system and controller set (not requiring trained gestures) for an immersive VR software visualization environment. It is also unique in providing multiple dynamically-switchable and customizable metaphors that support tagging, searching, and filtering of visual objects. An oracle in the form of a virtual tablet and keyboard are unique for providing an additional intuitive interaction capability within the VR landscape for accessing diverse external non-VR SE tool data.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.
VRIC '17, March 22–24, 2017, Laval, France
© 2017 Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-4858-4...$15.00
DOI:

## 3. SOLUTION

VR-FTC utilizes a 3D application domain view visualization [8] of program code structure (i.e. the software architecture). On these often invisible abstract structures, it provides a non-textual perspective, arranging customizable symbols in 3D space and enabling fly-through navigation. 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.

Figure 1 shows the VR-FTC game-engine-based architecture. Assets are used by the game engine (Unity in our implementation) and consist of Animations, Fonts, Imported Assets (like a ComboBox), Materials (like colors and reflective textures), Media (like textures), 3D Models, Prefabs, 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 call Tools consisting of General and Language-specific Tools. General Tools currently consist of BaseX, Graphviz, PlantUML, and Graph Layout - our own variant of the KK layout algorithm 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.

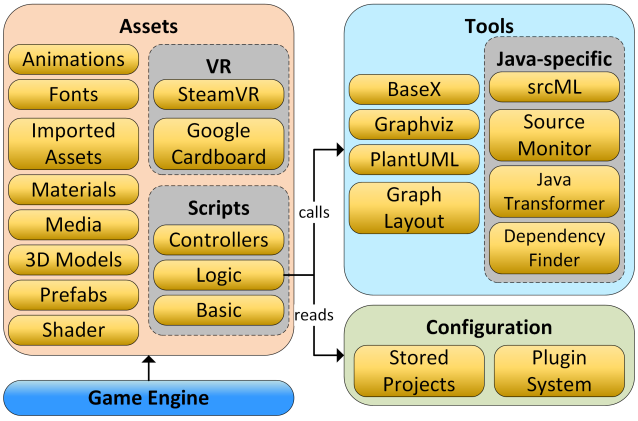


**Figure 1. VR-FTC software architecture.**

Solution principle include: *multiple dynamically switchable and tailorable 3D visual metaphors* (space, terrestrial, custom); *delineated grouping metaphor* of representative objects for code packages or components (solar systems, glass bubble or tree-lined cities); *directed connection metaphor* for object dependencies (pipes, light rays); *flythrough navigation* (i.e. motion) via camera movement by controllers in an anchored scene; an *oracle* to readily access data (virtual tablet); *tagging* support to custom label objects in the landscape; and *immersive background sounds*.

The process consists of (1) modeling generic program code structures, metrics, and artifacts as well as visual objects (2) mapping the model to a visual object metaphor (3) extracting a given project's structure and metrics (4) visualizing a given model instance within a metaphor (5) supporting navigation through the model instance (via camera movement based on user interaction).

## 4. IMPLEMENTATION

The HTC Vive, a room scale VR set, tracks movement of a head-mounted display and two wireless handheld controllers (see Figure 2) using two 'Lighthouse' base stations. Note that the pictured wall is mirroring what the subject is viewing. The touchpad on the left hand-held controller controls altitude (up, down) and the right one direction (left, right, forward, backward) to realize flythrough navigation by moving the camera position. A laser pointer for selection becomes visible when the controller enters the view field, (shown in the universe metaphor in Figure 2) and a selected object (class) changes to a whitish color and is pointed to by a rotating inverted pyramid to track smaller objects during navigation. When a controller is near to and pointing at the oracle (tablet), it changes to a finger for intuitive interaction.

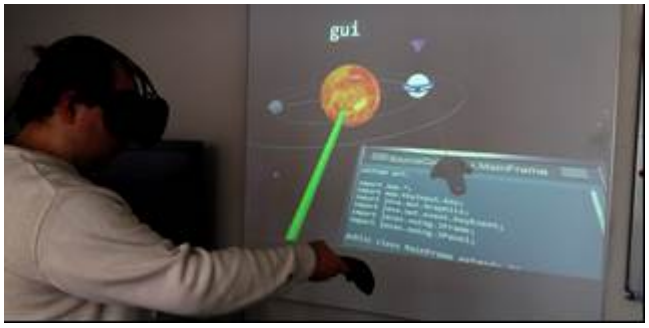


**Figure 2. Subject using Vive HTC headset and controller (visible in a scene making a planet selection in solar system).**

To exemplify support for multiple 3D visual metaphors, a universe (Figure 2 and Figure 3) and a terrestrial metaphor (Figure 4) were implemented, justified in [6]. In the terrestrial metaphor, labeled glass bubble cities represent packages; buildings represent classes (Figure 5 labeled in blue at the top) - the number of stories represent some metric (in this case the number of class methods); and colored pipes (Figure 4) show directed dependencies. In the universe, packages are represented by a solar system (Figure 2), classes via a labeled planet (Figure 3), and dependencies between classes or packages via directed colored light beams.

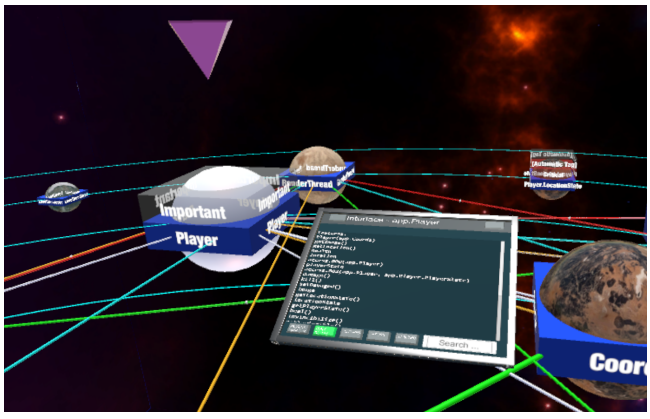


**Figure 3. Space metaphor: selected planet (class) has inverted pyramid as arrow, tags evident, and tablet shows interfaces.**

Our FTC variant (non-VR 3D) utilizes a semi-transparent Heads-Up Display (HUD) paradigm to show various informational screens. In VR mode, however, we determined that a HUD was not practical, since these screens contain large amounts of text

that must be fairly opaque to be readable - hiding the background landscape. Moreover, any head movement shifts the then hidden landscape and can thus cause disorientation, and the focal point must be further inset then on a display. Thus, we chose to use a virtual tablet as an oracle (Figure 6) for providing source code (Figure 2), code metrics (Figure 4), interfaces (Figure 3), UML, tagging, filtering, and project data for a selected object. The two unlabeled buttons at the top support switching to the previous or next screens, a scrollbar is on the right, and various features buttons are at the bottom.

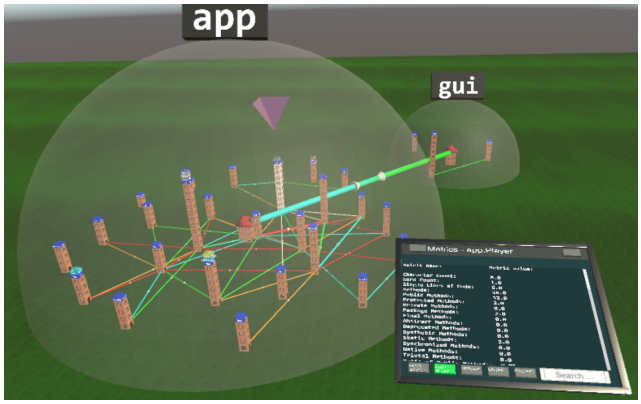


**Figure 4. Terrestrial metaphor with bubbled cities (packages) and oracle with metrics for selected object (pyramid pointer).**

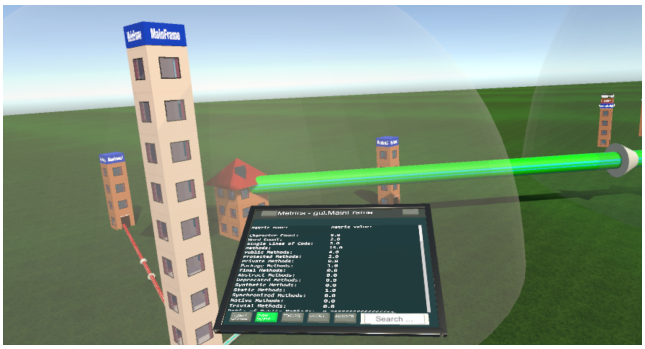

**Figure 5. Terrestrial metaphor with building (class) selected and oracle (tablet) showing metrics for selected object.**

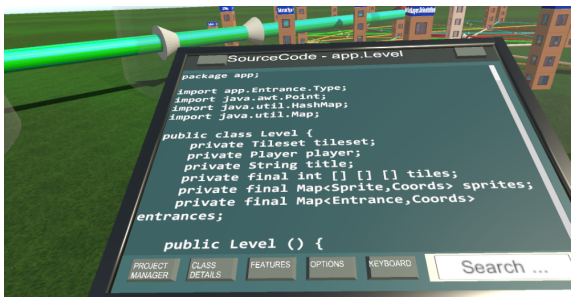


**Figure 6. The oracle (virtual tablet) interface showing source code. A bidirectional (dependency) pipe is in background.**

To enable users to more easily recall objects, persistent tagging permits labels matching automatic patterns or any manually inserted label to be placed on an object (e.g., the Important Tag on the Player class in Figure 3). A virtual keyboard (Figure 7) supports text input for searching, filtering, and tagging. A 2D dynamically generated UML class diagram, which functioned with the prior Unity version, has not as yet been ported but will be shortly.

**Figure 7. Virtual keyboard (shown in terrestrial metaphor).**

To support an inexpensive VR option with limitations, VR-FTC was also integrated with the Android SDK and Google Cardboard SDK for Unity and run with a Google Cardboard visor (see Figure 8). It utilizes smartphone motion sensors for movement. Using the gaze point, the reticle changes from a small closed circle to an open circle to indicate the object can be selected. External SE tools were not ported nor an oracle provided.

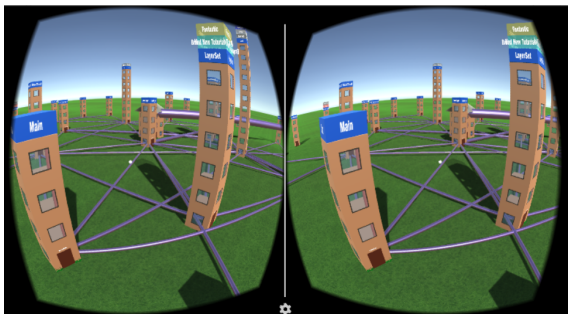

**Figure 8. Google cardboard implementation screenshot.**

XML is used to hold relevant source code, metrics, and metadata. For extracting existing code structure information into our model, srcML [18] is used to convert source code into XML and BaseX used for XML storage. Campwood SourceMonitor and DependencyFinder extract code metrics and dependency data, and plugins with Groovy scripts and a configuration file are used to integrate the various SE tools.

# 5. EVALUATION

The technical evaluation focused on assessing VR-FTC's viability on current VR hardware, whereas an empirical evaluation assessed its usability. The configuration consisted of an HTC Vive with 2160×1200 447 PPI resolution, Unity 5.5.0f3 PE, SteamVR 1481926580, 4GHz PC with i7-6700K, 32GB RAM, NVIDIA GeForce GTX980Ti with 6GB GDDR5, SSD, Win7 Pro x64 SP1.

## 5.1 Technical Evaluation

As to resource usage, RAM allocated for a 64-bit implementation was 220MB (with no project), 250MB (project with 27 classes), and 620MB (project with 95 classes), which scales appropriately.

For Google Cardboard we used a Sony Xperia Z2 2.3GHz Quad-core 5" 1080p 423 PPI display and a Samsung Galaxy A5 1.2GHz Quad-core 5" 720p 300 PPI display with Android 6. On the A5 with 720p, pixels were discernable due to the lower PPI. We observed significant drops in the frame rate. Future work will further optimize to assess smartphone performance sufficiency.

In an initial empirical experiment with 10 subjects on an older VR-FTC implementation based on Unity 5.3.5f1, three subjects had experienced VR sickness symptoms. We thus further

optimized our frame rates to reduce the risk of this discomfort issue recurring. Current common VR community guidance is that 30 frames per second (FPS) per eye provides a minimum for fluid motion, with 45 as a goal to significantly reduce any discomfort risk to users. Vive has a maximum of 90 FPS (45 per eye) (due to HDMI specification limitations), and Steam VR artificially reducing any higher rates to 45 per eye.

Further, we investigated metaphor differences on FPS. The Saxon XSLT 2.0 and XQuery processor (300K lines of code and 1635 classes) was loaded and, while doing common VR tasks, FPS was measured via a custom script. Both metaphors fall below 30, whereby the terrestrial metaphor is significantly worse (Figure 9). As to an explanation, higher FPS occur when fewer objects are visible, like flying to an outer package. The terrestrial view (buildings with windows) is generally visually more complex.

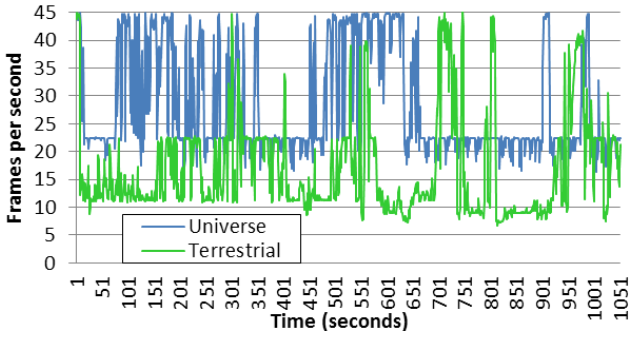


**Figure 9. Measured frame rates per metaphor over time.**

## 5.2 Empirical Evaluation

We measured the usability efficiency of the HTC Vive VR interface versus a conventional display and mouse in performing SE tasks with FTC. A convenience sample of two master students with VR experience were selected (to prevent VR novices from skewing results). Project A consists of 2 packages, 27 classes, and 170 methods; project B has 5 packages, 95 classes, and 800 methods. The SE tasks were: (1) How many dependencies does class X have within the package Foo? (2) Which package is the largest/smallest? (3) How many connections/dependencies does package Foo have? (4) How many connections/dependencies does package Bar have? (5) How many (variables; overloaded methods) are declared in the class Y in package Foo?

Figure 10 compares VR vs. non-VR for cumulative task durations across both projects and metaphors, showing no significant efficiency differences. The exception, Project B Universe in VR, took longer since, for this larger project, not all project objects are seen simultaneously due to a render distance limitation.

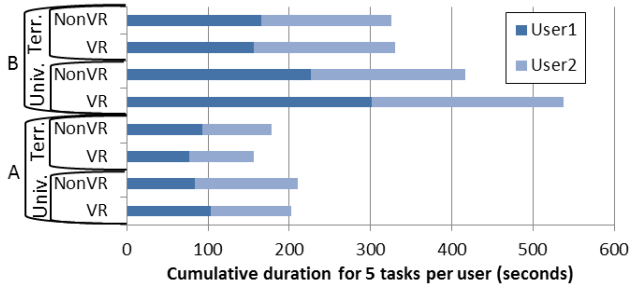


**Figure 10. Cumulative duration for 5 SE tasks in seconds.**

Our prior empirical study [6] already underscored the experiential and motivational benefits of VR-FTC for VR novices.

# 6. CONCLUSION

This paper contributes an immersive VR flythrough approach for program code structures utilizing multiple metaphors for visualizing, navigating, and conveying program code information interactively to support exploratory, analytical, and descriptive cognitive processes. The oracle virtual tablet interface was equivalently efficient and intuitive as the non-VR interface. Metaphors choices exhibited FPS differences and went below 30 FPS at times. Future work includes a more comprehensive empirical study and frame rate and smartphone optimizations.

# 7. ACKNOWLEDGMENTS

The authors thank Alexandre Matic and Camil Pogolski for their assistance with the design, implementation, and evaluation.

# 8. REFERENCES

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

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

[3] Avram, A. 2014. IDC Study: How Many Software Developers Are Out There? *[Online]*. http://www.infoq.com/news/2014/01/IDC-software-developers

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

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

[6] Oberhauser, R., Silfang, C., and Lecon, C. Code structure visualization using 3D-flythrough. In *Proc. 11th Int'l Conf. on Comp. Sc. & Educ.* (ICCSE), IEEE, 2016, pp. 365-370.

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

[8] Teyseyre, A. R. and Campo, M. R. 2009. An overview of 3D software visualization. *Visualization and Computer Graphics, IEEE Transactions on*, 15, 1 (2009), 87-105.

[9] A. Kashcha. Software Galaxies [Online]. http://github.com/anvaka/pm/

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

[11] Wettel, R. 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.

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

[13] Maletic, J. I., Leigh, J. and Marcus, A. 2001. Visualizing software in an immersive virtual reality environment. In *Proc. 23rd Intl. Conf. on Softw. Eng. (ICSE 2001)*. IEEE.

[14] Fittkau, F., Krause, A., and Hasselbring, W. 2015. Exploring software cities in virtual reality. In *Proc. IEEE 3rd Working Conf. Software Visualization (VISSOFT)*, IEEE, 130-134.

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