A Visionary Future of Software Engineering with Consumer Virtual Reality

Anthony Elliott
North Carolina State University
Raleigh, North Carolina, USA
anthony_elliott@ncsu.edu

Brian Peiris
Toronto, Ontario, Canada
brianpeiris@gmail.com

Chris Parnin
North Carolina State University
Raleigh, North Carolina, USA
chris.parnin@ncsu.edu

ABSTRACT

Virtual reality (VR) has recently been used to great effect by the gaming industry and offers orders of magnitude of improvement in software engineering in the years to come. A VR environment completely surrounds and immerses the user and uses stereoscopic rendering to enable depth-sensing. This enables the user to process more information and in a more natural way than traditional environments on a monitor.

We describe potential VR environments for code review, live coding, remote collaboration, and simulation. We also consider open research questions in this area.

1. INTRODUCTION

Quality virtual reality (VR) is finally inexpensive enough for consumers, including software engineers, to invest in. No longer will we have to go to a physical room to be fully immersed in a virtual reality, within the next few years we will be able to simply reach over and put on our fully immersive VR sunglasses and headphones. Virtual reality is often applied only to the entertainment industry but we argue that it will greatly change software engineering as well.

VR devices render a slightly different image for each eye (stereoscopic rendering) which allows humans to sense depth in the images just like in the physical world. VR devices also completely immerse the user in the environment, so that the user's head has six degrees of freedom. This realistic imitation of the human body enables easier and more natural navigation as well as the ability to display more information at once in a meaningful way. We get the naturalness of the physical world and the malleabillity of the virtual world.

We believe that these affordances provide a platform for new kinds of software engineering tools. We imagine a future in which many software engineers find it most productive to design and build software within a VR environment.

To immerse the user in a virtual environment we used a head-mounted display (Oculus Rift - Development Kit 2) and a Leap Motion Controller for gesture recognition. These devices are inexpensive (total is less than $500) and future iterations of them could feasibly be used by a number of developers within five years.

(a) Oculus Rift

(b) Leap Motion Controller

There have been substantial amounts of VR research done in the past, but very little of it has shown how VR can aid in software engineering. In this paper we present a vision of four software engineering activities that could benefit from VR.

We illustrate how immersive 3D environments could make software engineering tools easier to learn and use, and to display more information without overwhelming users. We show that such environments enable much tighter feedback loops in development which helps software engineers develop faster and with fewer mistakes. We describe shared virtual environments that could increase communication among remote teams. We also illustrate how virtual reality can simulate fields that have physical analogs.

2. RELATED WORK

Virtual environments called CAVEs have been around since the 1990's in which a person can be in a physical room with displays covering all surfaces [3]. Users often use head-mounted displays with head tracking to update the displays and may use hand-held devices to enable interaction with objects in the CAVE.

Andrew Bragdon has implemented a system called Code Bubbles that enables the user to pull out methods from a file. The user is then able to move and group the methods as desired. Code Bubbles also can display the call graph for the selected method and allows the user to open called methods [2].

Code Canvas added semantic zoom to the Code Bubbles tool, allowing the user to gain a better understanding of how the current section of the system fits into the system as a whole [4].

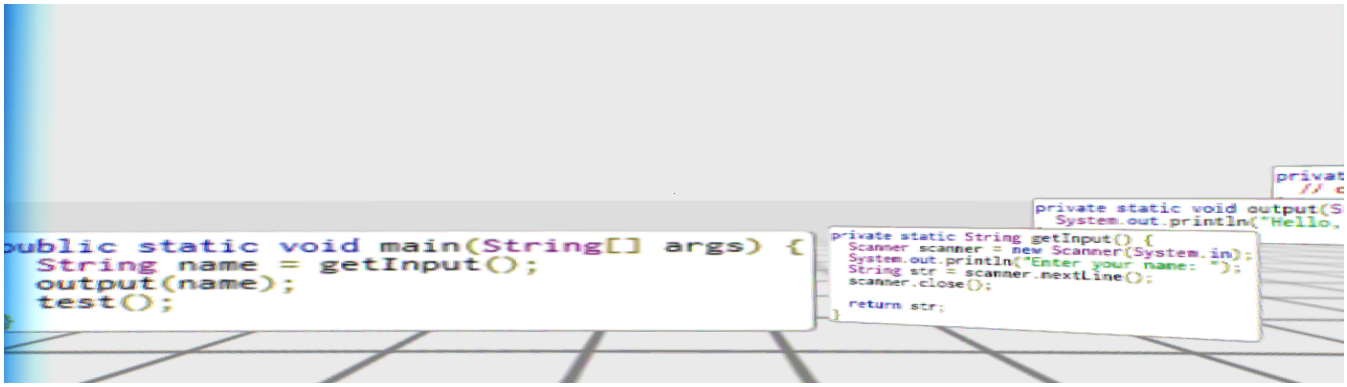

Figure 2: The active fragment is in the middle of the user's view. The user is currently looking at the implementations of the methods (shown to the right) called by the active fragment. The user is able to easily read 3 of these fragments at once.

Chris Parnin developed a tool called NosePrints which allows users to quickly see if a code smell is relevant [6].

RoomAlive by Microsoft Research is similar to a CAVE but uses six Kinect sensors around the room to track the user. This eliminates the need for wearable devices by the user but requires significant amounts of space. [5]

3. APPLICATIONS

Virtual environments can use the immersive property to display more information than on a computer monitor. We are still limited by the field of view of the human eye, so putting more information on the display is not novel, but what is novel is being able to distribute that information around the user in a meaningful way. This immersion is possible because the Rift uses stereoscopic rendering (creates a slightly different image for each eye) to enable depth sensing.

In this section we describe how VR could aid in code review, live coding, remote collaboration, and simulation. Our main contribution is not these specific ideas, but to show the possibilities of software engineering with VR.

3.1 Code Review

The primary reason for modern code reviews not resulting in more defects found is a lack of understanding of the code being modified [1]. Immersion is a prototype of a VR tool that seeks to use the immersive 3D environment to enable reviewers to gain a better understanding of the code being modified. The user views the environment through an Oculus Rift and interacts through hand gestures recognized by a Leap Motion Controller.

Immersion shows code fragments similar to Code Bubbles [2], but each fragment consists of exactly one method. The reviewer initially sees the active fragment in the center of the screen with other relevant information distributed around the floor in piles with more relevant piles closer to the reviewer. The reviewer is able to scan the label of the pile and number of fragments in each pile to quickly verify if this pile is indeed relevant. The reviewer then makes a grabbing motion towards the most interesting pile and pulls their hand up to show all of the fragments in a circular expanded view to the right of the active fragment.

The reviewer is now able to read the foremost fragment in the circle and can make horizontal finger swipes to rotate the circle and read other fragments. Immersion uses semantic zoom to make method names readable even when at the back of the circle and the fragment retains its height so that the reviewer can quickly see how many lines of code it has. The reviewer can select an especially large method using a pinch motion on the top and bottom and move it to become the active fragment. The piles then change to include code relevant to the new fragment. The reviewer can move their hand as if clearing off a desk to return to the previous active fragment.

Natural interaction paradigms are not new, but are even more powerful in a immersive 3D environment where the user truly feels like they are *inside* the environment. This feeling of presence could make immersive VR tools easier to use when compared to a 3D tool displayed on computer screen.

Immersion also displays overview diagrams of this changeset to allow the reviewer to see where this active fragment fits into the changeset as a whole. Similarly an overview of the cyclomatic complexity also helps the reviewer to see which areas of the code would be most relevant to see. These diagrams illustrate how VR is able to show multiple overview and detail diagrams without overwhelming the user.

Immersion is merely a vehicle to demonstrate the advantages that VR can bring to software engineering. Most software engineering tools don't display information in 3D because its not needed and merely looks cute. An immersive 3D environment enables more information to be displayed which should increase developer productivity. Additionally, such a VR environment allows users to interact more naturally with the tool which would make VR tools easier to learn and use.

3.2 Live Coding

RiftSketch is a live coding environment built for VR. It allows a user to describe a 3D scene using geometries, materials, textures, lights, meshes and other primitives provided by the Three.js library, which is a convenient wrapper for the WebGL APIs available in modern browsers. RiftS-

ketch presents a user with a simple text editor, floating in front of them in an otherwise empty VR world. As the user types code into the editor, the world around them updates instantly to display the 3D scene dictated by their code. RiftSketch also allows the user to animate their scene via a callback function which is executed on every frame. The user can manipulate the state of the 3D scene in this looped block of code in order to add behaviour to the objects in their scene.

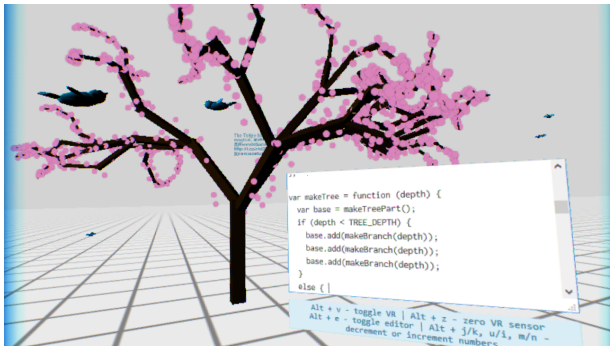

Figure 3: RiftSketch screenshot

Furthermore, RiftSketch provides the user with shortcuts and input methods to quickly edit numbers in the code that they write. Keyboard shortcuts allow the user to increment or decrement numbers in the code by increments of 0.1, 10 or 100. Integration with the Leap Motion Controller provides users with the ability to manipulate numbers using natural input: the user places the editor's cursor over a number and then can move their hand up and down above the Leap Motion controller in order to increase or decrease the number. Numbers in the code could represent anything, from the X,Y or Z components of a position or rotation vector, the red, green or blue components of an object's material or a component in the calculation of an object's animated speed.

Although the current implementation of RiftSketch is little more than a creative toy, the experience of using it shows the potential of practical applications. Largely inspired by Bret Victor's seminal lecture, "Inventing on Principle" [7][8], RiftSketch demonstrates the potency of giving a creator, or programmer, a direct and immediate connection to their creation. This tight feedback loop allows the programmer to better understand the effect of every part of the code that they write and to quickly experiment with various solutions, algorithms and calculations. RiftSketch is also very effective as a learning tool since users can see their mistakes immediately and correct themselves without an intermediate compile step that might otherwise act as a roadblock. These benefits are especially evident in RiftSketch when the code describes a VR scene. Watching the entire virtual world change around you as you type can be an extremely powerful and engaging experience.

In practical applications, one could extend the concept of live coding and tight feedback loops in order to create customized programming environments. VR programming environments could allow the user to take advantage of the malleability and expanse of virtual worlds. For example, a user could live code a widget that indicates some statistic of the code they are currently editing, such as the cyclomatic complexity, test coverage or test result. The widget would update itself as the user typed, not unlike existing code feedback tools such as NCrunch [1], and they could position the widget in their virtual periphery.

3.3 Remote Collaboration

Co-located development teams are able to communicate in person. Distributed teams can try, but the technology is too slow and it doesn't feel good. New tech is coming (High Fidelity) that is fast enough for developers to be able to sense body language and hear audio in real time. Distributed development teams will have the foundational issues of remote communication solved which enables a shared VR environment to become truly powerful.

A benefit of co-location is pair programming, where two developers program at the same computer and can catch each others errors. VR and body sensing with a device like PrioVR would enable remote developers to be at a shared virtual desk right next to each other. They would be able to see the same files, talk in real time, and even point to which line of the code they are talking about.

Sharing a live coding environment enables all participants to have a tight feedback loop. This could be especially useful when the primary developer of a section needs to explain their modifications to the rest of the team. Everyone could meet in the shared VR live coding environment and watch while the developer walks through modification of their code. The team would be able to see how the system changed rather than looking through large amounts of code.

Co-located teams can design together around a whiteboard in a conference room as they talk through the problems of the system. A shared VR environment could have a similar room with sketching space that everyone can see. Developers could drag-and-drop files or code snippets to display on the walls around them. They could then discuss and annotate the code. Additionally, visualizations such as structure diagrams or architecture of related systems could be overlaid on a wall for reference.

Remote workers could experience immersive visualizations together. A control flow graph could be represented by a maze of rooms through which the team could walk. The team members could trace different paths and yet still be shown an overlay of where each of the other members are and how these positions relate to each other. Currently remote workers can each trace separate paths of the code but have a hard time communicating what those paths are and how they relate to each other.

Remote workers often miss out on many benefits that in-person communication can bring. Shared VR spaces tailored to developers' activities can fill some of the gaps caused by distance and help the team to function better together.

3.4 Simulation

[1]NCrunch automated concurrent testing tool: http://www.ncrunch.net/

VR worlds are also well-suited to applications with physical analogs that can be simulated in 3D. For example, if a programmer was working on avionics software in a live coding environment, the VR world could include a simulation of a plane with moving control surfaces, propellers or a visualization of the aircraft's fuel injection system. The live coding environment would then give the programmer immediate visual feedback about the control flow and logic that they were programming. The programmer could scale the virtual plane to a manageable size to give her an overview of its behaviour, scale it to normal size and walk around the plane to inspect a particular control surface or scale it to larger than life-size, transport herself into the guts of the plane to watch a particular fuel valve respond to the state of the program.

[include rest of Brian's draft here]

4. OPEN RESEARCH QUESTIONS

We have presented virtual environments that completely immerse the user. Other devices such as Google Glass, aim to help the user complete tasks in the physical world by adding information overlays (augmented reality). Augmented reality seeks to help the user in the physical world while virtual reality seeks to completely replace physical reality. Is it more useful to immerse the user in a completely virtual, dynamic environment, or to enhance the physical world of developers?

Virtual reality is able to make users feel 'presence', that is, to trick their mind into thinking they are physically in the environment. This is critical to entertainment experiences but would creating such a compelling experience have tangible benefits like improved defect rate in a code review environment? Similarly, could a boring task like code review have better results if it were to be gamified and provide a more engaging experience?

The main advantage of VR is being able to display more information at once to the user. At what point does the extra information cause overload and begin to hinder productivity? Is this balance between amount of information and processing speed the same for each person (other aspects of VR can differ per person)?

Is it better to always have the area in front of the user's initial position display the most important information with secondary information displayed to the sides and rear? Would allowing the user to rotate their environment provide a different experience? Currently the equipment has physical limitations such as cords that prevent the user from easily rotating 360 degrees. As technology improves and this limitation is removed, will this affect how users desire information to be displayed?

What is the best way for the user to provide input to a VR system? The games industry has found a console controller to work well for navigation and limited action support but those controllers are poorly suited for text entry. However, while the keyboard is well suited for text entry, many people find it hard to type while wearing a head-mounted display. Gesture recognition devices like the Leap Motion Controller and Kinect sensor enable the user to perform more natural gestures, but often put too much strain on the user's arms. Voice added to gesture recognition could help to reduce that arm strain, but speaking to one's computer may feel disruptive and awkward in a shared office space.

What is needed for many developers to have a Rift? How many years before a developer or company would invest in such a device just for programmer productivity? What domains should be targeted for research now so that at least some developers could feasibly use such a device within two years? (thinking specifically of games developers)

5. CONCLUSIONS

We have shown that virtual reality enables a new interaction paradigm that can improve developers' lives. The immersive environment allows users to process more information at once and stereoscopic rendering enables depth-sensing. Together, these affordances provide a novel platform for new software engineering tools.

We have illustrated how immersive 3D environments could make software engineering tools easier to learn and use, and to display more information without overwhelming users. We have shown that such environments enable much tighter feedback loops in development which helps software engineers develop faster and with fewer mistakes. We have shown that shared virtual environments could increase communication among remote teams. We have also illustrated how virtual reality can simulate fields that have physical analogs.

6. REFERENCES

[1] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721. IEEE Press, 2013.

[2] A. Bragdon, S. P. Reiss, R. Zeleznik, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, Jr. Code bubbles: Rethinking the user interface paradigm of integrated development environments. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 455–464, New York, NY, USA, 2010. ACM.

[3] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The cave: Audio visual experience automatic virtual environment. *Commun. ACM*, 35(6):64–72, June 1992.

[4] R. DeLine and K. Rowan. Code canvas: zooming towards better development environments. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2*, pages 207–210. ACM, 2010.

[5] B. Jones, R. Sodhi, M. Murdock, R. Mehra, H. Benko, A. Wilson, E. Ofek, B. MacIntyre, N. Raghuvanshi, and L. Shapira. Roomalive: Magical experiences enabled by scalable, adaptive projector-camera units. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 637–644, New York, NY, USA, 2014. ACM.

[6] C. Parnin, C. Görg, and O. Nnadi. A catalogue of lightweight visualizations to support code smell

inspection. In *Proceedings of the 4th ACM symposium on Software visualization*, pages 77–86. ACM, 2008.

[7] B. Victor. Inventing on Principle (video). `http://vimeo.com/36579366`, 2012. [Online; accessed 11-Nov-2014].

[8] B. Victor. Inventing on Principle (transcript). `https://github.com/ezyang/cusec2012-victor/blob/master/transcript.md`, 2013. [Online; accessed 11-Nov-2014].