Virtual Reality in Software Engineering: Affordances, Applications, and Challenges

Anthony Elliott
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
anthony_elliott@ncsu.edu

Brian Peiris
Toronto, Ontario, Canada
brian@peiris.io

Chris Parnin
Department of Computer Science
North Carolina State University
Raleigh, North Carolina, USA
chris.parnin@ncsu.edu

***Abstract*—Software engineers primarily interact with source code using a keyboard and mouse, and typically view software on a small number of 2D monitors. This interaction paradigm does not take advantage of many affordances of natural human movement and perception. Virtual reality (VR) can use these affordances more fully than existing developer environments to enable new creative opportunities and potentially result in higher productivity, lower learning curves, and increased user satisfaction. This paper describes the affordances offered by VR; demonstrates the benefits of VR and software engineering in prototypes for live coding and code review; and discusses future work, open questions, and the challenges of VR.**

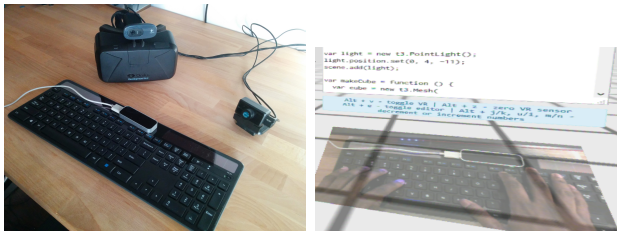

Fig. 1. Leap Motion with webcam attached to Rift (left), keyboard inside VR (right)

I. Introduction

Programming environments from the previous decades still do not address programmer issues despite advances in psychology, neuroscience, and social aspects of software development. As a result, problems still persist. Developers still experience disorientation when navigating code [9]. Developers still experience problems comprehending code [13]. These basic issues also impede other important software engineering activities. For example, in code review, due to insufficient ability to navigate and understand the code under review, developers mostly report issues, such as convention violations, instead of discussing design flaws or defects [2].

Researchers have explored the cognitive issues underlying several problems developers experience [17]. One such issue is *spatial memory*, a memory system in the parahippocampus that supports the ability to retain spatial awareness. Ko et al. [11] observed that developers lost track of relevant code when the cues they relied on, such as position of scroll bars and document tabs, were disrupted as a result of their navigation. Similar disorientation often results from failures to engage a human's innate processing of spatial memory.

Affordances are devices that leverage innate cognitive mechanisms. Researchers have attempted to improve interfaces that incorporate the human capacity for attention, cognition, and memory. For example, one study reduced storage and retrieval time of web bookmarks as well as reduced retrieval failures by positioning screenshots of the pages on various piles in a 3D space [18].

Similarly, researchers have incorporated affordances for spatial memory in programming environments. Code Canvas [6] positions code files on a large scrollable, zoomable plane which preserves stable spatial positions of code. Code Bubbles [3] allows a developer to quickly position related fragments of code on an infinitely scrollable screen, which improves navigation and comprehension of fragments.

Spatial memory is just one such affordance leveraged to improve programming environments, many other affordances could be similarily leveraged. Virtual reality (VR) can use these affordances more fully than existing developer environments to enable new creative opportunities and potentially result in higher productivity, lower learning curves, and increased user satisfaction.

This paper describes how VR provides affordances in spatial cognition; manipulation and motion; and feedback that are not yet fully utilized in programming environments. We then describe two VR systems we have prototyped and how VR can be applied to software engineering. Finally, we provide a brief discussion of future work, open research questions, and challenges of using VR in software engineering with existing technologies.

II. Affordances in Virtual Reality

VR provides affordances in spatial cognition; manipulation and motion; and feedback.

1) Spatial Cognition: Spatial memory is supported by place cells, specialized neurons that fire as a person navigates through a physical space and other contextual cues in the environment. Head-mounted VR displays allow the user to update their view by moving their body or rotating their neck, firing place cells in the process. Additionally, these displays render a slightly different image for each eye (stereoscopic rendering) which enables the human eye to more easily sense

depth of images on the display. These displays can create a sense of *presence*, or 'being there' [20]. We expect users' spatial memory to be more engaged in VR than for 2D displays, especially when viewing 3D visualizations.

VR can directly mimic the affordances offered by physical navigation. Based on electrocorticography (eCoG) recordings on the surface of the brain, place cells could be observed firing in the same sequence as a person navigated a virtual town and again when they later recalled the paths through the town [7].

2) Manipulation and Motion: The affordances provided by manipulation of a physical object can result in improved perception and retention. For example, the affordances offered by turning pages of a book result in increased comprehension and recall as when compared to reading the same text on computer displays [15]. Additionally, the ability to serendipitously browse and relocate material is improved. Motion in a physical space, through exertions such as walking, have important cognitive consequences [16]. Other affordances can be aided by motion. For example, perception of depth is enhanced by self-actuated movement in a space [8].

Researchers have explored integrating natural interactions with existing programming environments [5]. Through input devices such as the Leap Motion, it is possible to physically interact with virtual objects. Physical movement is also possible in virtual spaces. Body harnesses[1] allow free movement in a virtual space such as walking, running, jumping and crouching.

3) Feedback: The Gulf of Evaluation, as described by Norman, arises when there is difficulty assessing the state of the system [14]. VR allows software engineers to be in environments that attempt to eliminate this gulf by eliminating the delay between programmer action and seeing the result of their action. Such fast feedback has previously been implemented in two-dimensional displays [4] but VR extends this ability to three-dimensional displays.

III. Applications

We have built VR prototypes for live coding and code review which concretely demonstrate the benefits of using the affordances of VR. We argue that the benefits described in this section can also be extended to other software engineering activities.

Both of these systems use a head-mounted display (Oculus Rift - Development Kit 2) and a Leap Motion Controller for gesture recognition.

A. Live Coding

RiftSketch[2] is a live coding environment built for VR which allows users to describe a 3D scene using the Three.js library[3].

RiftSketch presents a user with a simple text editor (see Figure 2), 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. This animation makes the user truly feel *inside* the scene in a way not captured by a 2D screenshot.

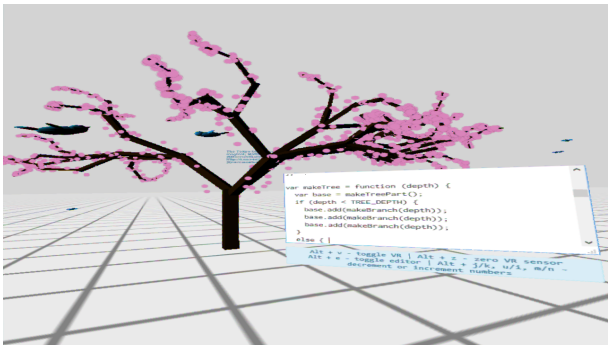

Fig. 2. RiftSketch screenshot. The tree is generated by a recursive algorithm that the user has typed into the floating editor. The flying, animated birds represent tweets pulled in from the Twitter API, also generated by code that the user has entered.

To assist in interaction with the keyboard, we allow reality to shine through by using a web camera mounted on the Rift and project that image in the system (See Figure I).

1) Feedback: RiftSketch provides a tight feedback loop between code written and its effect in a virtual environment, enabling quick experiments with various solutions, algorithms and calculations. RiftSketch is also very promising 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 hindrance. These benefits are especially evident in RiftSketch when the code describes a VR scene. As the authors have previously experienced, watching the entire virtual world change around you as you type can be an extremely powerful and engaging experience.

2) Hand Manipulation of Code: 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 factors of 0.1, 10 or 100. Integration with the Leap Motion Controller provides users with the ability to manipulate numbers using an up and down hand motion. This allows users to continuously modify a number using their hand and instantly see how this affects the scene, enabling faster feedback than manually typing one number at a time.

3) Usage Example: Consider the following scenario: A space mission has just landed a probe on the surface of a comet. After 10 years in flight, the probe lands against all odds but in a position that is unable to receive sunlight. Automated telemetry programs cannot find a feasible flight path. As a programmer, you are tasked with updating the lander's software to reposition itself safely on solid ground and you have 24 hours before its batteries die and the probe deflects off the comet surface. Thankfully the companion

[1] http://cyberith.com/product/
[2] http://www.youtube.com/watch?v=SKPYx4CEIlM
[3] http://threejs.org/

orbiter has gathered detailed information about the surface around the landing site and telemetry data shows exactly where and how the lander is positioned.

You update your simulation with the data and step into RIFTSKETCH to survey the situation. After assessing the lander's options, you iterate on possible solutions, first by using hand gestures to manipulate a possible path and scale thrust settings, and then the keyboard to refine the code. With each solution, you observe the lander's behavior inside RIFTSKETCH. You walk around the lander to inspect its position after each maneuver, leaning in to ensure that its feet are planted firmly in the regolith, and zooming out to an orbital view to verify that its new position maintains a line-of-sight to the orbiter on this lob-sided comet.

Finally, after having run the simulation dozens of times in RIFTSKETCH, you pass the software on to review.

B. Code Review

IMMERSION represents methods as code fragments similarly to Code Bubbles [3] and displays groups of fragments as piles on the floor like BumpTop [1]. Piles can be expanded into a more detailed ring for an overview and detail visualization [19].

1) Spatial Reasoning: The reviewer initially sees the active fragment in the center of the screen (see Figure 3) with other relevant fragments distributed around the floor in piles. Reviewers use spatial cognition to judge the relevance of piles by how far away the pile is as well as the size of the pile. The reviewer is able to scan the labels of the piles and number of fragments in each pile to quickly verify if each pile is indeed relevant.

IMMERSION divides the floor into sections based on packages of the system and color codes the sections to indicate how much that package has been modified by the code under review. By walking between packages, we expect reviewers to have better mental models due to the increased usage of spatial reasoning and thus understand the code better during the review. Similarly, we expect the increased spatial reasoning to enable reviewers to more easily recall review details after the review. This would allow reviewers to provide more useful feedback in future reviews of the same code base.

2) Gesture Interaction: Reviewers can make a grabbing motion to select a pile and then can pull their hand up to transform the pile into a ring of fragments for more detailed inspection. 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. The reviewer can pinch the foremost ring fragment on the top and bottom and move it to the middle of the screen to become the active fragment. If the reviewer wants to return to the previous method, they can move their hand as if clearing off a desk.

We have initially focused on supporting exploration—comments can be added via keyboard input, but we are investigating alternative ways to mark and flag code.

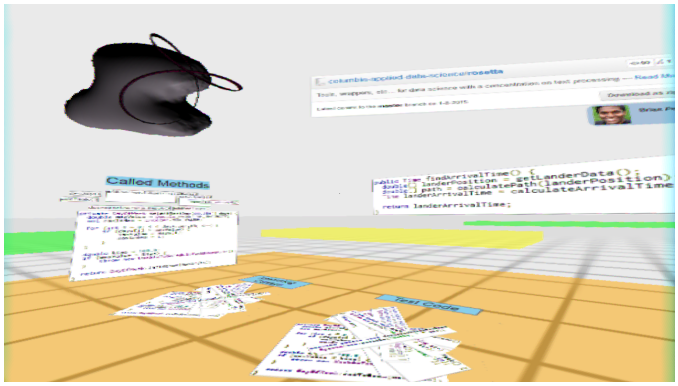

Fig. 3. IMMERSION screenshot. The reviewer is reviewing code to reposition the lander on the comet. The reviewer sees the active method, piles of relevant fragments on the floor, and has expanded one pile into a fragment ring on the left to read the details of those fragments. A model of the comet and the lander's expected flight path is shown in the upper left. The reviewer can walk between code packages on the floor which are color coded according to amount of modification for this review. GitHub details are shown in the upper right.

3) Usage Example: Consider the previous comet scenario where a programmer has implemented a repositioning flight path for the lander.

A reviewer puts on a Rift and enters IMMERSION to ensure this solution will actually work. She notes an edge case which might cause a collision and suggests gently crashing the companion orbiter into the probe to avoid a larger collision. The original programmer implements the suggestion in RIFTSKETCH, ensures the simulation works and celebrates before submitting the correction to the reviewer. However, the reviewer sees a section of code light up in IMMERSION. As she walks over to the section she sees that it is the collision detection section warning that the system will not allow this code to execute outside of the simulator. She realizes she can allow the execution by shutting off the engine just before impact to override the system. They upload the code and the lander repositions itself as expected.

IV. DISCUSSION

A. Simulation

VR has opened the door for software engineers to create systems that increase efficiency and enable previously impossible experiences. Existing applications include NASA who is using VR to control a robotic arm resulting in higher efficiency [4]. Educational experience Titans of Space [5] allows students to experience our solar system in a way that makes them feel like they are truly right next to the sun.

Future research is needed on how to create tools for the software engineers creating these VR systems. What problems do VR software engineers face that have no tool support?

[4]www.engadget.com/2013/12/23/nasa-jpl-control-robotic-arm-kinect-2/
[5]www.crunchywood.com

B. Remote Collaboration

Multiple programmers located around the world could join each other in a VR live coding space to figure out how to land the comet from the motivating example. They would be able to provide extra insight and could arrive at the solution faster.

A different set of programmers could then join each other in a VR code review environment. They are able to see what each person is thinking based on their annotations of the piles of information in their section of the system.

C. Open Questions

Degrees of Immersion. Augmented reality devices such as Google Glass, aim to help the user complete tasks in the physical world by adding information overlays. 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 environment, or to enhance their physical world?*

Input Forms. Gaming console controllers work well for navigation and limited action support but are surpassed by keyboards at text entry. However, such devices require users to interact with both the physical and virtual worlds at the same time. Gesture recognition removes interaction with the physical world but can cause physical strain. Voice recognition could reduce strain but may feel awkward in a shared work space. *What is the best way for the user to provide input to a VR system?*

D. Challenges

1) Separation from the Physical: Putting on a VR headset means blocking out the rest of the physical world, including coworkers. Peers may lack the opportunity to ask questions and physical communication is stifled. Additionally, the VR headset wearer may have trouble interacting with the physical world while in VR. A webcam mounted to the headset enables some interaction with the physical world, as seen in Figure I, but has a limited field of view.

2) 3D Mapping: Some problems don't have an inherent 3D representation which makes display in VR a challenge. As seen in IMMERSION, 2D code can be displayed in VR but the code itself does not have a third dimension and thus loses the expressiveness of 3D. This could be an area well suited to 3D metaphorical programming as suggested by Ko et al. [10].

3) Technology Limitations: The 1080p resolution of the Oculus Rift Development Kit 2 allows for passable text reading, but needs improvement for multi-hour sessions. Each user also needs slightly different configuration which requires time to set up properly.

V. CONCLUSIONS

Two-dimensional development environments have not been able to take full advantage of affordances such as spatial cognition, manipulation, and feedback. This paper describes a vision of how software engineering can use VR for new kinds of tools that can take advantage of these affordances. We described how both live coding and code review could benefit from VR tools but we envision many other software engineering activities can benefit from VR as well.

REFERENCES

[1] A. Agarawala and R. Balakrishnan. Keepin'it real: pushing the desktop metaphor with physics, piles and the pen. In *Proc. of the SIGCHI conference on HFCS*, pages 1283–1292. ACM, 2006.

[2] A. Bacchelli and C. Bird. Expectations, outcomes, and challenges of modern code review. In *Proc. of ICSE*, pages 712–721. IEEE Press, 2013.

[3] 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 *Proc. of the 32nd ACM/IEEE ICSE - Volume 1*, ICSE '10, pages 455–464, New York, NY, USA, 2010. ACM.

[4] M. M. Burnett, J. W. Atwood Jr, and Z. T. Welch. Implementing level 4 liveness in declarative visual programming languages. In *Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on*, pages 126–133. IEEE, 1998.

[5] D. Delimarschi, G. Swartzendruber, and H. Kagdi. Enabling integrated development environments with natural user interface interactions. ICPC 2014, pages 126–129, New York, NY, USA, 2014. ACM.

[6] R. DeLine and K. Rowan. Code canvas: zooming towards better development environments. In *Proc. of the 32nd ACM/IEEE ICSE-Volume 2*, pages 207–210. ACM, 2010.

[7] A. D. Ekstrom, M. J. Kahana, J. B. Caplan, T. A. Fields, E. A. Isham, E. L. Newman, and I. Fried. Cellular networks underlying human spatial navigation. *Nature*, 425(6954):184–188, Sept. 2003.

[8] R. HELD and A. HEIN. Movement-produced stimulation in the development of visually guided behavior. *Journal of comparative and physiological psychology*, 56:872–876, Oct. 1963.

[9] A. Z. Henley and S. D. Fleming. The patchworks code editor: Toward faster navigation with less code arranging and fewer navigation mistakes. In *Proc. of the SIGCHI Conference on HFCS*, CHI '14, pages 2511–2520, New York, NY, USA, 2014. ACM.

[10] A. J. Ko, B. A. Myers, and H. H. Aung. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, pages 199–206. IEEE, 2004.

[11] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32(12):971–987, Dec. 2006.

[12] A. Kuhn, D. Erni, and O. Nierstrasz. Embedding spatial software visualization in the ide: An exploratory study. In *Proc. of the 5th International Symposium on Software Visualization*, SOFTVIS '10, pages 113–122, New York, NY, USA, 2010. ACM.

[13] R. T. K. R. Maalej Walid, Tiarks Rebecca. *ACM Transactions in Software Engineering and Methodology*, 23(4):31:1–31:37, 2014.

[14] D. A. Norman. *The psychology of everyday things*. Basic books, 1988.

[15] J. M. Noyes and K. J. Garland. Computer- vs. paper-based tasks: are they equivalent? *Ergonomics*, 51(9):1352–1375, Sept. 2008.

[16] M. Oppezzo and D. L. Schwartz. Give your ideas some legs: The positive effect of walking on creative thinking. *Journal of experimental psychology. Learning, memory, and cognition*, Apr. 2014.

[17] C. Parnin and S. Rugaber. Programmer information needs after memory failure. ICPC 2012, pages 123–132, June 2012.

[18] G. Robertson, M. Czerwinski, K. Larson, D. C. Robbins, D. Thiel, and M. van Dantzich. Data mountain: Using spatial memory for document management. UIST '98, pages 153–162, New York, NY, USA. ACM.

[19] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Visual Languages, 1996. Proc.., IEEE Symposium on*, pages 336–343. IEEE, 1996.

[20] M. Slater. Place illusion and plausibility can lead to realistic behaviour in immersive virtual environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 364(1535):3549–3557, 2009.

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