VR-Based User Interactions to Exploit Infinite Space in Programming Activities

Víctor Stefano Segura Castillo *, Leonel Merino †, Geoffrey Hecht *, Alexandre Bergel *
*ISCLab, Department of Computer Science (DCC), University of Chile, Chile
†DILab, School of Design and School of Engineering, Pontifical Catholic University of Chile

2021 40th International Conference of the Chilean Computer Science Society (SCCC) | 978-1-6654-0956-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/SCCC54552.2021.9650396

***Abstract*—Virtual reality (VR) devices have now become a commodity, and as such, VR is percolating the traditional working environment of software programmers. Current approaches to use VR as the medium to immerse software programmers essentially project classical IDE windows in the virtual environment: the very same VSCode or IntelliJ window is seen through the VR device. As a consequence, the same constraints imposed by a physical screen are found in the VR environment, thus representing a missed opportunity.**

VRIDE is a new VR-based environment for object-oriented programming to let software developers carry out their activities in a full VR-based immersed environment. VRIDE innovates by offering interactions based on Code Cubes that are designed to exploit the infinite space in the VR environment. Through code cube interactions our approach disrupts the traditional mapping of windows from desktop to VR by supporting dedicated actions for navigation, inspection, and space management. Our prototype illustrates the feasibility of having a full-immersive virtual environment for software programmers.

I. Introduction

The field of Virtual Reality (VR) has recently seen numerous significant progresses, including the emergence of sophisticated controllers, broad range of VR-based applications, and innovative application domains.

A number of attempts have been made to incorporate VR devices within the working environment of software programmers [1]. However, the common approach to let a programmer develop application in a VR environment is to just map mainstream classical, non-VR, IDEs into the VR World. Consider Immersed[1], one of the most popular VR environments for programmers. Immersed maps traditional windows, as seen in a plain standard physical computer, into windows projected in VR. As a consequence, the working environment offered to a programmer is still structured in terms of virtual screens and windows. One of the benefits of Immersed is to "spawn up to 5 virtual monitors in VR" and no need to redesign the environment for VR. Whereas the VR world offers an infinite space to exploit, Immersed constraints a programmer to structure her programming environment in terms of windows and screens. We argue that simply projecting non-VR tools into a virtual reality environment represents a missed opportunity by forcing users to interact with the environment the same way than in a non-VR setting.

[1]https://immersed.com

978-1-6654-0956-8/21/$31.00 ©2021 IEEE

VRIDE. This paper describes VRIDE, a VR environment for programmers that offers unique interactions to exploit the infinite space offered by VR. VRIDE provides a complete programming environment for the Pharo [2] and Python programming languages. Essential programming tools, including the code browser, inspector, playground, as well as, navigation operations have been redesigned to benefit from the infinite space. VRIDE supports the motto "programmers need no physical screens" by offering the support to program while being fully immersed. A programmer does not need to remove the VR headset to carry out part of some programming activities.

Contribution. This paper proposes the graphical metaphor of *code cube*, a technique to represent and interact with software artifacts (such as source code, windows and tools) in a VR-environment. A code cube represents a class, which is the primary software component in class-based object-orientation. A code cube can be *unfolded* to inspect and edit a class, and *spawn* to reflect navigation actions.

Our paper contributes to the state-of-the-art by (i) identifying an opportunity that has been missed by classical VR-based environments, and (ii) proposing code cube, a visual metaphor to let programmers exploit a VR-based infinite space.

Outline. The outline of this paper is as follows: Section II describes VRIDE and the code cube metaphor; Section III presents a programming example and how it can be carried out with VRIDE; Section IV highlights some key aspects of the implementation of VRIDE; Section V is dedicated to related works; Section VI concludes and outlines our future work.

II. VRIDE

VRIDE is a VR-based programming environment that supports object-oriented programming. Two programming languages are currently supported by VRIDE, namely Pharo and Python. VRIDE proposes *code cube*, an innovative visual technique to carry out programming activity.

A. Code cube

Code cube is a visual metaphor to represent a class, which is the primarily structural element in object-oriented programming. Figure 1 illustrates a code cube for a particular class, as seen through the VR device. A class is visually rendered as a cube, in which each facet represents a particular aspect of the class. Those aspects could be:

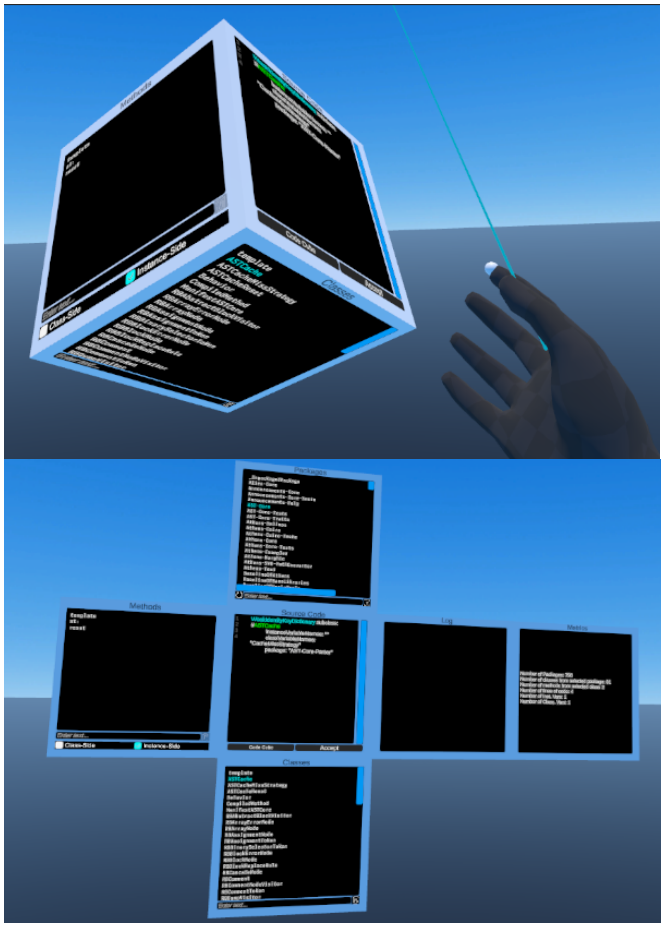

Figure 1. A folded and unfolded code cube

1) *Class definition*: a textual description of the class definition is provided listing the class name, its super class, and the declared instance variables. The definition can be customized to express refactorings (e.g., class renaming, variable renaming), or incremental changes (e.g., adding a new variable).
2) *List of methods*: methods defined by the represented class are listed in an alphabetic order.
3) *Method source code*: if a method is selected in the previous facet, then the actual source code of the method is given in a dedicated facet. A method source code can be read and modified directly within this facet.
4) *Scatterplot*: software metrics are visually represented as a scatterplot to compare the represented class against other classes in the application. Currently, we consider two metrics: number of methods and number of lines of code.
5) *Incoming dependencies*: Code navigation is often expressed in term of following dependencies between software components. This facet lists the components that directly refers to the class associated to the code cube.
6) *Outgoing dependencies*: Similarly, classes that are directly used by the browsed class are listed and accessible through a dedicated facet.
7) *Superclass and subclasses*: class hierarchy can be navigated through using a dedicated facet.
8) *Log Entry*: the output obtained during class definition and script execution.

As programmers spend much time navigating the components of software systems, we designed VRIDE in way that it facilitates navigation. A code cube supports classical interactions expected in a 3D environment: a code cube can be translated and rotated along the three axes. Translating code cubes is useful to manually cluster related classes. A group of related classes can be located in a particular place in the virtual environment.

In VRIDE, grouping classes happens by explicitly locating code cubes in a particular position in the virtual infinite space. Rotating a code cube lets one make a particular aspect more explicit. Oppositely, in a standard tab-based programming environment such as Eclipse, IntelliJ, and Visual Code, grouping related classes happen by manually rearranging tab and windows. Projecting a particular aspect of a class is achieved using perspective (using Eclipse) or views (using IntelliJ). Our approach, based on code cube, enables the same operations, but carried out in a very different fashion, by operating on cube located in a VR world.

B. Code cube unfolding

A code code can be unfolded to show all the facets at the same time. Independently of the rotation angle when folded, all facets are orthogonally-oriented to the users' camera when unfolded. Unfolding a code cube is designed to support a programming activity focused on a particular class. In a traditional and non-VR environment, this operation can be best represented as opening a textual file to read or modify the content of the class. Figure 1, lower part, gives the example of an unfolded code cube. Unfolding is animated and provides a comfortable visual experience by being smooth and fluid.

C. Spawning code cube

Navigating within a software code base is expressed in terms of code cube spawning, i.e., adding new code cubes in the virtual environment. Spawning a new code cube lets one to augment the current working set with a new or existing class. As such, a code base exploration happens by spawning code cubes. Some of the facets described previously are dedicated to exposing the connections with the environment. In particular, the facets *incoming dependencies*, *outgoing dependencies*, *superclass and subclasses* lets one to open a new code cube on an *existing* or *new* class. A new cube is created and located in the virtual environment next to the former class. This new code cube can be individually rotated and drag-and-dropped.

D. Removing code cube

Narrowing the focus of the working environment is expressed by reducing the number of code cubes presented in the virtual environment. Code cubes can be gathered, and eventually removed using a VR broom, a tool to collect code cubes. A programmer can use the broom by drag-and-dropping

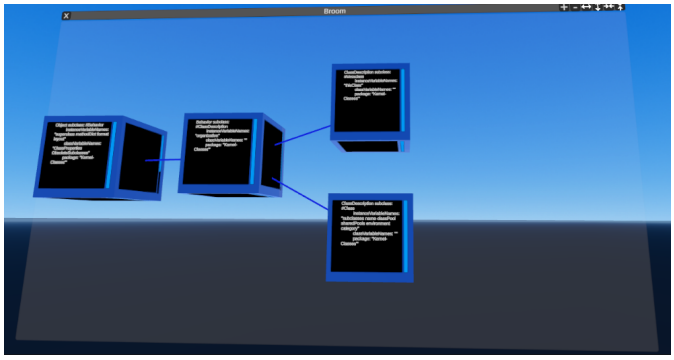

Figure 2. Live demonstration of a broom. The image shows a pair of visualizations attached to the broom

it using a hand controller. The cubes collected by the broom can be translated, scaled, and removed. The VR Broom is a handy tool that allows to attach an arbitrary number of windows and visualizations. The broom operates as a magnetic whiteboard. The user can stick elements to the broom thanks to a collider component put in every code cube. Figure 2 illustrates a VR broom in which a class tree, expressed in terms of connected code cubes, is attached. The broom can then be manipulated as a transparent board. Each cubes are located in the same the same plane and they can be resized, translated, and removed in the same way. The benefit of the broom is to reduce a significant effort to perform the very same action on multiple code cubes. Once cubes are collected, only one action can be applied to all the cubes.

E. Playground Cube

A playground is a tool in which a user can execute scripts in it. Those scripts typically execute arbitrary statements or simply invokes the entry point of a larger application. A *playground cube* is a particular cube that support editing and executing scripts. A playground cube can be folded and unfolded to let one define a particular script on a facet, as shown in Figure 3.

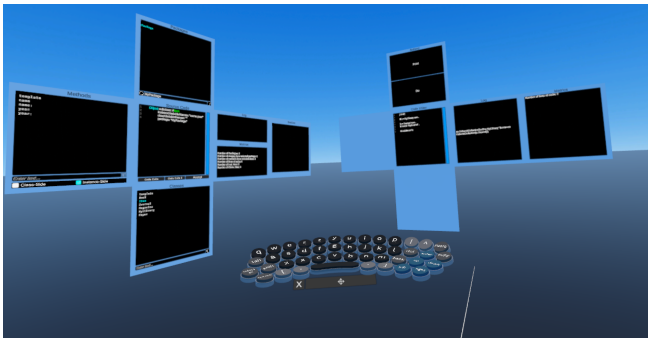

Figure 3. Screenshot of both a browser cube (left side) and a playground cube (righ side). Both cubes are currently unfolded.

III. Usage Example

In a nutshell. Meet Victor, a software engineer. He wants to adopt VRIDE as his programming environment. To this end, Victor decides to define a library that can contain various kinds of items, with the possibility of making queries. This can be done using design patterns, such as visitors, and by defining a class hierarchy. Victor adopts a workflow, shown in Figure 4, as follows:

Step 1. He opens a new code cube to navigate through packages, classes, and methods. This kind of code cube provides all the tools from a Pharo System Browser, while showing other information. The cube is closed at the beginning for a better usage of 3D space.

Step 2. Victor opens the cube by pressing a couple of buttons from the controller. The facets are displayed in the same plane, so he can define a class, its methods and its variables by writing code inside the source code facet. Once he is done, all previous elements will be available inside the method list facet and class list facet.

Step 3. Victor wants to check for dependencies and hierarchy. Therefore, will open a new code cube from the browser, displaying all info related to that class. The cube will display the following information: 1) source code, 2) incoming dependencies, 3) outgoing dependencies, 4) a scatter plot of the number of lines of code vs. number of methods, 5) parent class, and 6) children classes.

Step 4. Then, Victor opens a Halo menu, which allows programmers to navigate through classes that are part of the dependencies and hierarchy. All selected elements will appear as new code cubes, and they will have a connecting line to their related classes.

Step 5. Victor navigates through the newly generated cubes, by hovering the cubes with the pointer or grabbing the cube with his hands.

Step 6. After inspecting and navigating through all the classes, Victor wants to close all of the code cubes at once. As shown in Figure 2 and Figure 6, a new broom is opened, so all elements can be dragged and deleted at the same time.

Figure 5 shows the usage of 3D space by the user and the cubes. We can see that space usage is almost 24 m^3 and points tend to concentrate where cubes and the user are working together. This demonstrates how an approach like VRIDE encourages the user to exploit the infinite space provided by VR. In this case, the user used several cubes scattered in several directions within the 3D space.

IV. Implementation

VRIDE is implemented using Unity and acts as a frond-end to a Pharo and Python interpreter. VRIDE implements a whole user interface library to build and interact with code cubes. Communication with the server is simply performed using POST request that exchange serialized data between the Unity front-end and the server. The project is open-source, and can be found online.[2]

Full and permanent immersion. VRIDE has been designed to let practitioners *fully develop* their application within the

[2]https://github.com/Vito217/VRIDE

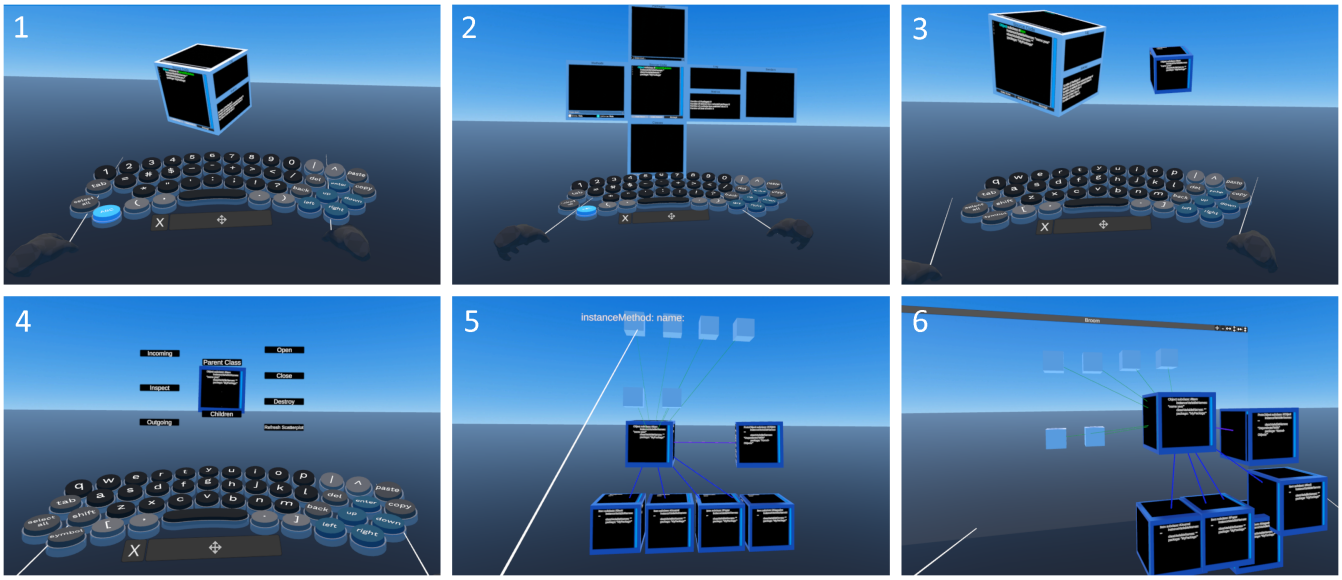

Figure 4. Example shown from left to right, and from up to down. (1) shows a virtual keyboard and a Browser as a code cube. In (2), the cube is fully open for better reading. In (3), a new code cube has spawned, representing a new defined class. (4) displays the halo menu working. (5) shows the inspection of the code cube. Finally, (6) shows the VR Broom, where every cube is attached to it.

the virtual environment. VRIDE offers a set of development tools that allow one to comfortably develop within the virtual environment. Users can use the controllers to interact with the UI elements. They also are given the option to use either a virtual keyboard or a physical keyboard to write code. As such, developers are able to conduct a programming activity without removing the VR device. Consequently, we think that VRIDE can be helpful to address various questions that happen in live programming [3].

Standalone. Most of VR software works with a device connected to a desktop computer or laptop, While this is still the case with VRIDE, it is also compatible with standalone devices, like the Oculus Quest, where there is no need to use a separate machine. Since VR devices are quite expensive, this feature reduces that extra cost.

Session. Session is kept by storing a data file associated to each user. Said file contains distinctive information, such as last opened tools, and a log file that saves all the interactions done by the user.

V. Related Work

As mentioned before, projects like Immersed VR tend to translate desktop views to 3D environments. Another example is CaffeineJS where a WebXR environment maps a Smalltalk browser to the three dimensional space using JavaScript.[3]

Important works have analyzed the use of immersive environments such as virtual and augmented reality to cope with multiple software engineering tasks. Merino *et al.* [4] created CityVR, a software visualization tool that uses the city metaphor to represent code as buildings. Each building can be seen as a cube representing a specific class, showing stuff like metrics and source code. A similar approach is presented by Rüdel *et al.* [5] and Steinbeck *et al.* [6], comparing software navigations using the software called EvoStreets. Sharma *et al.* [7] proposed using augmented reality to incorporate real objects to the workspace. Similar to this, Mehra *et al.* [8] added the capability of choose the surrounding environment. This is also supported by the idea of VR open offices by Ruvimova *et al.* [9]. Elliott *et al.* [10] explores the benefits of VR programming by using RiftSkecth, which allows to generate code as a tree. Sharma *et al.* [11] present Immersive Project Management, focusing on team development. Coworking would be also presented by Fereydooni *et al.* [12], pointing the importance of remote workspaces in the context of COVID-19 pandemic. Hori *et al.* [13] developed CodeHouse, a VR interaction tools that visualizes source code as if it were a house, where each module is represented as a room. Other works with software visualization include Fittkau *et al.* [14] who introduce ExplorViz, a hierarchical visualization used in large software landscapes, which would be later integrated into VR by Zirkel *et al.* [15].

VRIDE builds on these previous works. It aims to not only support software visualization but also other software engineering tasks. VRIDES offers a self-contained environment that can help to overcome space limitations through tailored interactions. For instance, immersive software interaction through cubic representations.

VI. Conclusion and Future Work

Our paper presents code cube as a way to exploit the virtual space, without relying on traditional windowing system. In that way, our approach could be adopted by software programmers

[3]https://caffeine.js.org/

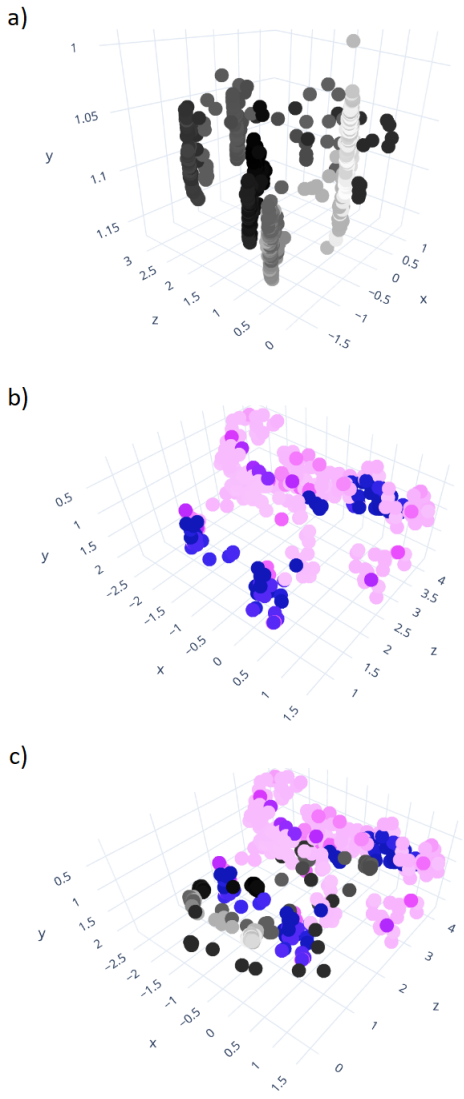

Figure 5. Graphical representation of the 3D space usage, where each point represents a position at a certain time. (a) shows the space used by the user, (b) the space used by the cubes and (c) the space used by the user and the cubes together. Darker colors mean further in time, while each unit on each axis are equivalent to 1 meter approximately.

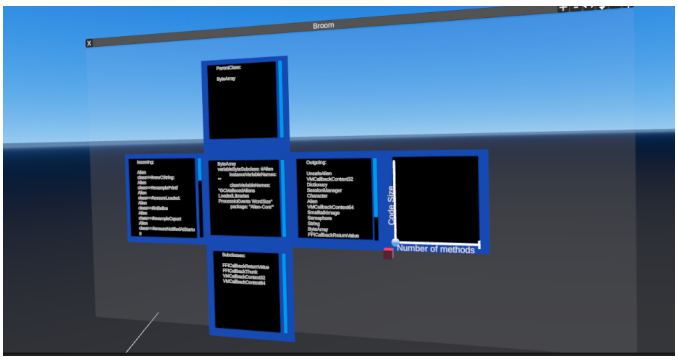

Figure 6. A opened Code Cube, where each facet displays the children classes, parent class, incoming classes, outgoing classes, methods and variables. Last face shows a lines of code vs. number of methods plot.

to incorporate virtual reality in their daily development environment. Our implementation, which is publicly available, demonstrates the feasibility of a VR-based programming environment. Moreover, our usage examples indicate that non-trivial programming tasks can be carried out.

The work presented in this paper is preliminary, and the following roadmap is foreseen: First, we will conduct a usability test to check how well users can move around the 3D space and write some basic code inside VRIDE. Second, we will assess how code cube affects the performance of software programmers. Third, we will identify the limitations of VRIDE in conducting some complex programming tasks, involving functionality and performance debugging.

References

[1] L. Merino and O. Nierstrasz, "The medium of visualization for software comprehension," Ph.D. dissertation, Universität Bern, 2018.

[2] A. Bergel, D. Cassou, S. Ducasse, and J. Laval, *Deep Into Pharo*. Square Bracket Associates, 2013. [Online]. Available: http://books.pharo.org/deep-into-pharo/

[3] J. Kubelka, R. Robbes, and A. Bergel, "Live programming and software evolution: Questions during a programming change task," in *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 2019, pp. 30–41.

[4] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "CityVR: Gameful software visualization," in *International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2017, pp. 633–637.

[5] M.-O. Rüdel, J. Ganser, and R. Koschke, "A controlled experiment on spatial orientation in vr-based software cities," in *Working Conference on Software Visualization (VISSOFT)*. IEEE, 2018, pp. 21–31.

[6] M. Steinbeck, R. Koschke, and M. O. Rüdel, "How evostreets are observed in three-dimensional and virtual reality environments," in *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2020, pp. 332–343.

[7] V. S. Sharma, R. Mehra, V. Kaulgud, and S. Podder, "An extended reality approach for creating immersive software project workspaces," in *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*. IEEE, 2019, pp. 27–30.

[8] R. Mehra, V. S. Sharma, V. Kaulgud, S. Podder, and A. P. Burden, "Immersive IDE: Towards leveraging virtual reality for creating an immersive software development environment," in *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*, 2020, pp. 177–180.

[9] A. Ruvimova, J. Kim, T. Fritz, M. Hancock, and D. C. Shepherd, ""transport me away": Fostering flow in open offices through virtual reality," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, 2020, pp. 1–14.

[10] A. Elliott, B. Peiris, and C. Parnin, "Virtual reality in software engineering: Affordances, applications, and challenges," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 547–550.

[11] V. S. Sharma, R. Mehra, V. Kaulgud, and S. Podder, "An immersive future for software engineering: Avenues and approaches," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 105–108.

[12] N. Fereydooni and B. N. Walker, "Virtual reality as a remote workspace platform: Opportunities and challenges," August 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/virtual-reality-as-a-remote-workspace-platform-opportunities-and-challenges/

[13] A. Hori, M. Kawakami, and M. Ichii, "Codehouse: Vr code visualization tool," in *2019 Working Conference on Software Visualization (VISSOFT)*, 2019, pp. 83–87.

[14] F. Fittkau, A. Krause, and W. Hasselbring, "Hierarchical software landscape visualization for system comprehension: A controlled experiment," in *IEEE 3rd Working Conference on Software Visualization (VISSOFT)*. IEEE, 2015, pp. 36–45.

[15] C. Zirkelbach, A. Krause, and W. Hasselbring, "Hands-on: experiencing software architecture in virtual reality," 2019.