Pre-print version, the final IEEE publication is DOI: 10.1109/VISSOFT55257.2022.00028

UML-based Live Programming Environment in Virtual Reality

Jakub Kučečka, Juraj Vincúr, Peter Kapec, Pavel Čičák
Faculty of Informatics and Information Technologies, Slovak University of Technology in Bratislava
Bratislava, Slovakia
Email: {xkuceckaj, juraj.vincur, peter.kapec, pavel.cicak}@stuba.sk

***Abstract*—Understanding and developing large applications often leads to a situation where the developer becomes lost. Software visualization aims to help with these problems by providing visual insights and helps to navigate, and, when properly done, also to visually model software. Although the advantages of virtual reality in software visualization have already been demonstrated, relatively limited research has been done in the area of programming.**

In this paper, we present a virtual reality programming environment, with built-in round-trip engineering support and runtime compilation of source code that allows live programming. The study carried out shows that users are capable of solving complex programming tasks in our VR environment, which performed better in terms of usability and user experience, but the evaluation results also point to still persistent VR related problems.

***Index Terms*—Virtual Reality, Model-Driven Engineering, Live Programming, VR Keyboard, Hand Tracking.**

I. INTRODUCTION

The feasibility of using Virtual Reality (VR) in software engineering has been the subject of many research projects. These projects focus mainly on visual analysis of software engineering data and demonstrate the advantages of VR in the context of spatial cognition [1], comprehension [2]–[4], navigation and interaction [1], [4]. Although most of these approaches visualize the source code, they do not provide an efficient way of editing it. To do so, programmers must switch from the virtual environment to the real environment. Although it seems that the ability to directly edit code in an efficient way would be crucial for any virtual environment focused on software development, relatively little research has been done in this area.

By combining VR software modeling tools that use the Unified Modeling Language (UML) notation [4] with code generation [5] we could potentially completely avoid direct code editing; however, developers would need to model the software using multiple views (types of diagrams). Such focus switching might be counterproductive, and we believe that developers should always have the ability to edit the code directly or use only a subset of diagrams.

To bring the code editing features to virtual reality, it would be necessary to completely redesign the traditional 'bento box' development environment, which was originally designed to be displayed on a single monitor. Some research projects propose alternative designs of the development environment to overcome the issues of traditional editors [6]–[8] and these might also be more suitable for virtual reality. UML-based approaches are particularly interesting since fragments of code are attached to the corresponding elements of UML diagrams, providing a fair level of abstraction, supporting round-trip engineering, and developers should be already familiar with this notation.

In this paper, we present a virtual reality programming environment that enables developers to write, generate, edit, compile, and run source code directly in virtual reality. The organization of source code is inspired by Octo Bubbles; therefore, the structure of the project is given by the UML class diagram, and the fragments of source code are attached to its elements. To evaluate our approach in terms of usability, we conducted a study with 20 participants in which we compared programming in our environment with programming in Unity.

II. RELATED WORK

Elliott et al. proposed a live coding environment built for VR [1], which allows users to describe a 3D scene using JavaScript code. Developers in this environment are presented with a simple text editor, where they can type using a physical keyboard. A similar approach in terms of code editing has been presented by Oberhauser and Lecon [9]. In this case, the code editor is not restricted to a simple file or fragment, but developers can navigate through a more complex structure visualized using space or a city metaphor. In both cases, we consider traditional, purely text-based code editing, in combination with relatively poor typing capabilities, as the biggest shortcoming.

Several innovative 2D non-VR Integrated Development Environments (IDE) have been proposed, e.g., Octo Bubbles [7], Code Bubbles [6], [10], Code Canvas [8]. All of these are based on the idea of organizing fine-grain code fragments on a 2D canvas to enable programmers to focus only on relevant parts of the code and juxtapose them in a more efficient way. In Code Bubbles, selected code fragments are visualized via fully editable and interactive views. During the evaluation, the developers noted similarities with UML diagrams and thought that UML integration would be natural and useful. Following these findings, Jolak et al. proposed Octo Bubbles, where code fragments are attached to elements of the UML class diagram.

Multiple systems have been presented that utilize 3D visualizations to display UML diagrams, ranging from the simple transfer of 2D UML diagrams into a 3D space [11], using

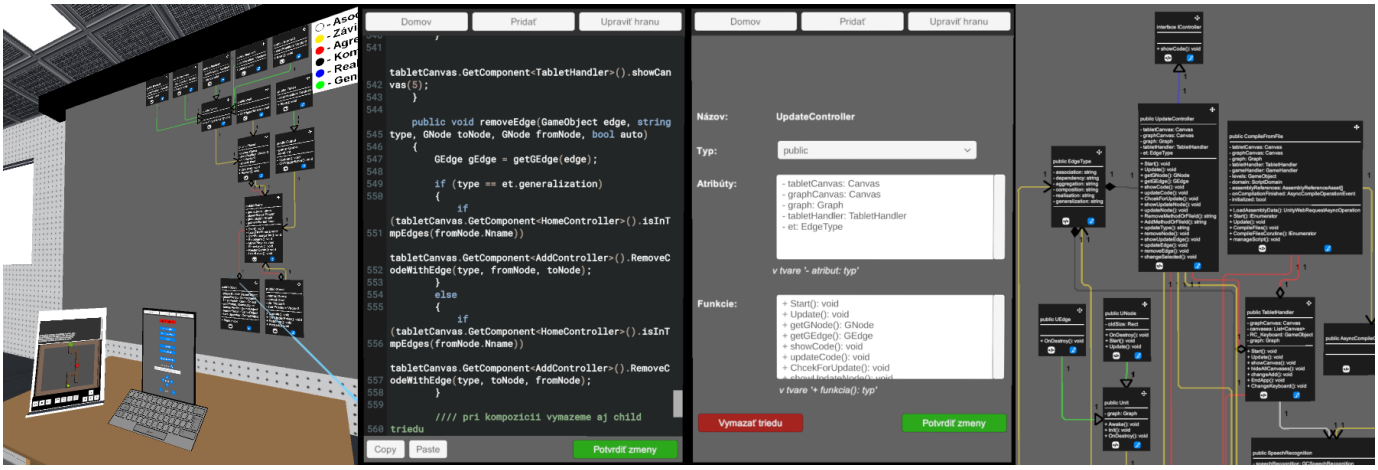

Fig. 1. VR programming environment from left: the whole VR scene, details of VR tablet with source code and class editor, UML class diagram on the wall.

multidimensional UML visualizations [12], to approaches using 3D geometry [13] or 3D geon diagrams [14].

Software visualization applications that use VR systems have been proposed to help analyze the structure [15]–[17] and architecture [18], [19] of existing systems and combine VR systems with UML diagrams [20], [4].

Although live programming is quite popular in the music industry or recently also in architectural design [21], live programming and/or modeling generic software in VR systems is still a challenge [9], [22]–[24], as most existing solutions struggle with effective source code editing.

III. LIVE PROGRAMMING ENVIRONMENT IN VR

The proposed VR programming environment was created in the Unity 3D game engine and its implementation is available online[1]. The solution is primarily designed for VR headsets with hand tracking, keyboard tracking, and passthrough capabilities (we used Oculus Quest 2).

The environment consists of three main parts. The first is the diagram canvas (Fig. 1, on the right), which is the area where the UML class diagram is rendered. The second is a virtual tablet (Fig. 1, center) that allows users to modify the selected element of the class diagram or the source code attached to it. It also provides a basic menu that allows users to save, load, or compile the project. The third is the developer area (Fig. 1, on the left), which consists of the sandbox application and the output window. The behavior of the sandbox application is given by the produced source code, and the user can freely interact with it using the interface in the bottom area.

INTERACTION

Users navigate through the virtual environment by walking, leaning, crouching, and approaching scene objects. Due to the size of the scene, other locomotion techniques (e.g., teleportation, flying) were not required.

Interactions with scene objects are performed using the left or right controllers. 2D / UI elements are selected by a ray originating from the controller, and selection is confirmed by the trigger button. 3D objects can be grabbed and moved by a controller while holding the grip button.

To provide an efficient way of text input, we utilize multiple techniques. The first is speech recognition. After analyzing different options and comparing publications [25], [26], we used the Google Cloud Speech API for speech recognition. The resulting input method was not suitable for longer texts and code fragments, so we kept it as an option only for single-line entries. The second implemented method utilizes passthrough and keyboard tracking capabilities of the selected VR headset. Passthrough mode enables users to see their hands and physical keyboard, but due to the low resolution of passthrough cameras, key labels are not readable. Thus, keyboard tracking is being used to provide a higher-resolution overlay. Unfortunately, incompatibility between the Roslyn C# compiler and the IL2CPP scripting back-end prevents us from using the passthrough mode (IL2CPP required), so we were forced to temporally visualize hands using a less accurate 3D representation based on hand tracking data (Fig. 2), which negatively affects typing speed and overall user experience. Since keyboard tracking was supported with selected models only, as the last input method, we provide traditional, canvas-based, virtual keyboard, where keys are selected in the same way as 2D / UI elements.

UML CLASS DIAGRAM RENDERING

The UML class diagram rendered on the diagram canvas consists of standard UI primitives offered by the selected game engine. The diagram is internally represented as a graph, where the nodes are placed according to the Sugiyama layout, and the edges follow rectilinear routing. We use the MSAGL library[2] for graph layout and edge routing.

[1]https://jakubkucecka.github.io/UML-based_VR_LiveProgrammingEnvironment/

[2]https://github.com/microsoft/automatic-graph-layout

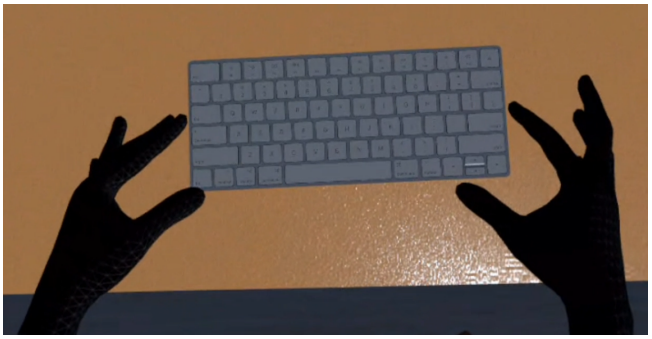

Fig. 2. VR Magic Keyboard with hand model.

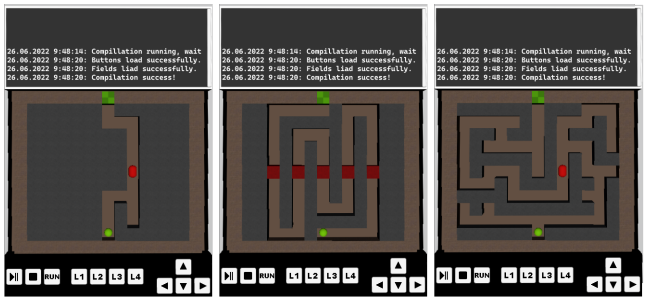

Fig. 3. Levels of the game used for evaluation.

ROUND-TRIP ENGINEERING & LIVE PROGRAMMING

Any change to the UML class diagram is reflected in the source code and vice versa. This is achieved using a detailed analysis of the source code provided by the Roslyn C# library[3] as follows: 1) If the UML has been modified, we apply the changes to the AST representation of the source code and convert them to compilable source code. 2) If we modify the source code directly, we use the AST representation to modify the internal JSON object representation of the UML class diagram. To create the initial UML class diagram from the source code, we extract these six relations: *association*, *dependency*, *aggregation*, *composition*, *realization*, and *generalization*. To extract these relations, we defined a set of custom rules that are specific to scripts developed in Unity 3D. These rules still comply with the UML specification, which was confirmed by senior software engineers who actively work with this notation.

The Roslyn C# library also provides means for live programming by speeding up the compilation of modified source code and allowing one to change the actual implementation of scripts at runtime.

IV. EVALUATION

In the evaluation, we sought answers to the following research questions:

1) Can software development in VR (using controllers, speech, and a VR keyboard) be on a par with traditional software development on a 2D monitor (using a keyboard and mouse)?
2) Can software development in VR improve the user experience?
3) Can software development in VR reduce the time required to code computer programs?

We perform the evaluation in the context of game development, on a prepared project of a simple escape game. The goal of the game is to navigate the runner through the environment (from start to finish) while avoiding static and moving obstacles (barriers and ghosts). The game has 3 levels (Fig. 3) with increasing complexity. These levels are impossible to complete without modification of the source code.

[3]https://assetstore.unity.com/packages/tools/integration/roslyn-c-runtime-compiler-142753

In *Level 1*, it is necessary to adjust the speed of the ghost, as it is too fast for the player to avoid it. This requires finding the speed attribute in the script responsible for ghost behavior, adjusting it to a smaller value than the speed of the runner, saving the changes, and compiling the new version. In *Level 2*, we prepared a series of barriers of different types that needed to be removed or disabled, since when the runner touched the barrier, the game returned to the initial state. Thus, the participant has to find a script for each of the barriers, ideally only the base class (referenced by a generalization relationship in the diagram), modify it, save the changes, and compile modified scripts. *Level 3* is relatively long and contains a section that is possible to pass only with great difficulty. Thus, it is necessary to create a shortcut. One possibility is to create a new object with a behavior similar to the finishing zone. To achieve this, it is necessary to implement a procedure that spawns a new prefab into the scene and adds proper components to it. Afterwards, it is again required to save and compile the changes.

The same game was implemented in two environments: VR IDE and Unity 3D. Participants solve 3 tasks in both environments, where in each task we ask them to modify the source code of a single level, so that they are able to complete it. Since the tasks were identical for both environments, all even participants first solve the tasks in Unity 3D and the odd participants first solve the tasks in VR IDE.

All participants successfully complete the tasks in both environments. After completion of the tasks in one of the environments, each participant completes two questionnaires. The System Usability Scale (SUS) [27], which is an effective tool for assessing the usability of an application, and the User Experience Questionnaire (UEQ) [28], which is a fast and reliable questionnaire to measure the user experience of interactive applications. We also measure the time it takes for each participant to complete specific tasks and collect feedback. There were 20 participants in this evaluation. 16 participants were students from different faculties focused on computer science, and 4 participants were from industry.

The results of the SUS questionnaire show that the VR IDE scored 75.875 (STD = 3.8), which corresponds to grade B, while Unity3D scored 52.25 (STD = 6.99), suggesting grade D. Therefore, our environment performed better in terms of usability. The second in a series of questionnaires is the UEQ.

Here, VR IDE scored significantly better than Unity3D on the scale of attractiveness, perspicuity, stimulation, and novelty (Fig. 4 and Table II). From these results, we can conclude that VR IDE is more attractive than the traditional 2D approach.

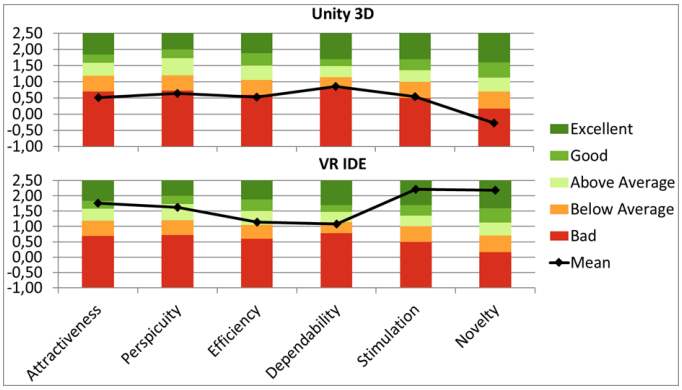

Fig. 4. UEQ VR IDE versus Unity 3D.

In terms of task completion time (Fig. 5), VR IDE performed worse in each task. The participants solved the first task in VR IDE in an average time of 2:35 min, while in Unity 3D it took 1:37 min. The second task, where minimal text input was required, was solved in VR IDE in 2:20 min, in contrast to Unity 3D, where it took 1:43 min. In the third, the most complex task, where the largest text input was required, it took them 9:02 minutes to complete it. In Unity 3D, the same result was achieved in 5:01 min.

When we compared the results of the third task for participants who had experience with Unity 3D (14 participants), the difference between the two groups decreased markedly (Fig. 5 and TableI). This was due to the fact that some of the previous outliers were included in the comparison of smaller samples. The difference is still statistically significant, but it may suggest that an experiment with a larger number of participants should be carried out.

We can see that developing in our VR IDE does not reduce the time required to code computer programs; however, in task 2 the difference was not statistically significant (Table I). The differences in completion time were relatively small in each of the 3 tasks.

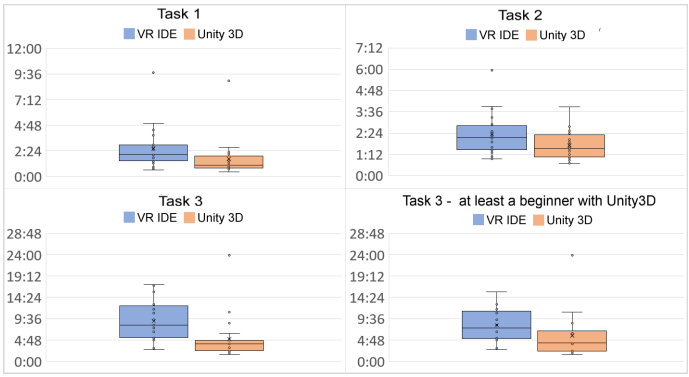

Fig. 5. Comparison of times for each task; times are in minutes.

TABLE I
MANN-WHITNEY U TEST TO CALCULATE THE STATISTICAL SIGNIFICANCE OF INDIVIDUAL TASK TIMES, ALPHA=0.05.

Task	U	p-value	Z-index
1	273.5	0.006905	2.7014
2	245.5	0.05962	1.8836
3	309	0.00002427	4.2214
3 at least a beginner	149	0.00526	2.7907

TABLE II
UEQ TWO-SAMPLE T-TEST ASSUMING UNEQUAL VARIANCES, ALPHA = 0,05.

Scale	p-value	Result
Attractiveness	0,0027	Significant Difference
Perspicuity	0,0239	Significant Difference
Efficiency	0,1884	No Significant Difference
Dependability	0,6011	No Significant Difference
Stimulation	0,0001	Significant Difference
Novelty	0,0001	Significant Difference

V. DISCUSSION

In the comments collected as feedback, the participants saw the importance of incorporating UML into software development. They stated that it seamlessly interconnects business logic with development. It can also quickly reveal the hierarchy or arrangement of a program and its parts, making it easier to find individual methods and attributes. They also appreciated the sample scene where they could test their own implementation at run-time. They also stated that as they developed the game, they often forgot about the fact that they were developing and felt as if they were part of the game itself. Regarding the VR keyboard, the participants considered it as the best way to write in VR, but it lacked a certain degree of certainty, which was caused by a slight shift of the hand model, which also affected the results. This could be eliminated by the passthrough mode as previously discussed. Surprisingly, users did not use speech recognition. Instead, they entered short texts using a ray-cast keyboard, which they judged to be ideal for such a purpose. They were discouraged by the volume at which they had to give commands while using this feature. An interesting comment regarding possible improvements was the incorporation of highlighting of classes in the UML diagram after selecting a game object in the sample scene. The last of the suggestions was to incorporate hand gestures. This feature was requested due to the long time required by the application to switch from VR controllers to hand tracking that was utilized by the VR keyboard. We could not affect this behavior as it is the current limitation of the Oculus Integration library. Adding support for hand gestures would avoid this switching completely and thus significantly reduce time required for text input.

VI. CONCLUSIONS & FUTURE WORK

We are convinced that there is a great potential of software development in virtual reality. It is more attractive to users, as we were especially convinced by the results of the UEQ

questionnaire. With the currently available VR technologies, it is possible to get closer to the effectiveness of classical development in a common real-world environment with 2D displays, although the development time is slightly longer. After improvements, both on the hardware and software side, this aspect could be eliminated in the future, and software development in VR will be competitive, if not outperform traditional approaches.

Future work can be dedicated to further explore the benefits of allowing collaborative software development in virtual offices and experimentation with advanced UML visualizations in VR systems such as multidimensional UML [12] or 3D UML [4].

VII. ACKNOWLEDGMENT

This publication has been written thanks to the support of the Operational Programme Integrated Infrastructure for the project: Research in the SANET network and possibilities of its further use and development (ITMS code: 313011W988), co-funded by the European Regional Development Fund (ERDF). This work was also supported by the Scientific Grant Agency of Slovak Republic (VEGA) under the grant No. VG 1/0759/19; and in cooperation (Financial support) with Siemens Healthineers Slovakia.

REFERENCES

[1] Anthony Elliott, Brian Peiris, and Chris Parnin. Virtual reality in software engineering: Affordances, applications, and challenges. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 547–550, 2015.

[2] Akihiro Hori, Masumi Kawakami, and Makoto Ichii. Codehouse: Vr code visualization tool. In *2019 Working Conference on Software Visualization (VISSOFT)*, pages 83–87, 2019.

[3] Roy Oberhauser and Carsten Lecon. Gamified virtual reality for program code structure comprehension. *International Journal of Virtual Reality*, 17(2):79–88, Jan. 2017.

[4] Roy Oberhauser. Vr-uml: The unified modeling language in virtual reality – an immersive modeling experience. In Boris Shishkov, editor, *Business Modeling and Software Design*, pages 40–58, Cham, 2021. Springer International Publishing.

[5] Maryam I. Mukhtar and Bashir Shehu Galadanci. Automatic code generation from uml diagrams: The state-of-the-art. *Science World Journal*, 13:47–60, 2018.

[6] Steven P. Reiss, Jared N. Bott, and Joseph J. LaViola. Code bubbles: A practical working-set programming environment. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1411–1414, 2012.

[7] Rodi Jolak, Khan-Duy Le, Kaan Burak Sener, and Michel R.V. Chaudron. Octobubbles: A multi-view interactive environment for concurrent visualization and synchronization of uml models and code. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 482–486, 2018.

[8] Robert DeLine and Kael Rowan. Code canvas: zooming towards better development environments. *2010 ACM/IEEE 32nd International Conference on Software Engineering*, 2:207–210, 2010.

[9] Roy Oberhauser. Immersive coding: a virtual and mixed reality environment for programmers. In *Twelfth International Conference on Software Engineering Advances (ICSEA 2017)*, pages 250–255, 2017.

[10] Andrew Bragdon, Robert Zeleznik, Steven P. Reiss, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola. Code bubbles: A working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, page 2503–2512, New York, NY, USA, 2010. Association for Computing Machinery.

[11] Paul McIntosh, Margaret Hamilton, and Ron Van Schyndel. X3d-uml: 3d uml state machine diagrams. In *Inter. Conference on Model Driven Engineering Languages and Systems*, pages 264–279. Springer, 2008.

[12] Lukáš Gregorovič, Ivan Polasek, and Branislav Sobota. Software model creation with multidimensional uml. In Ismail Khalil, Erich Neuhold, A Min Tjoa, Li Da Xu, and Ilsun You, editors, *Information and Communication Technology*, pages 343–352, Cham, 2015. Springer International Publishing.

[13] Claudia Susie C Rodrigues, Cláudia ML Werner, and Luiz Landau. Visar3d: an innovative 3d visualization of uml models. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, pages 451–460. IEEE, 2016.

[14] Pourang Irani and Colin Ware. Diagramming information structures using 3d perceptual primitives. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 10(1):1–19, 2003.

[15] Jonathan I Maletic, Jason Leigh, and Andrian Marcus. Visualizing software in an immersive virtual reality environment. In *Proceedings of ICSE*, volume 1, pages 12–13. Citeseer, 2001.

[16] Filipe Fernandes, Claudia Susie Rodrigues, and Cláudia Werner. Dynamic analysis of software systems through virtual reality. In *2017 19th Symposium on Virtual and Augmented Reality (SVR)*, pages 331–340. IEEE, 2017.

[17] Roy Oberhauser and Carsten Lecon. Immersed in software structures: A virtual reality approach. In *Tenth International Conference on Advances in Computer-Human Interactions (ACHI 2017)*, pages 181–186, 2017.

[18] Lisa Nafeie and Andreas Schreiber. Visualization of software components and dependency graphs in virtual reality. In *Proceedings of the 24th ACM Symposium on Virtual Reality Software and Technology*, pages 1–2, 2018.

[19] Andreas Schreiber and Marlene Brüggemann. Interactive visualization of software components with virtual reality headsets. In *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 119–123. IEEE, 2017.

[20] Martin Misiak, Doreen Seider, Sascha Zur, Arnulph Fuhrmann, and Andreas Schreiber. Immersive exploration of osgi-based software systems in virtual reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1–2, 2018.

[21] Renata Castelo-Branco and António Leitão. Algorithmic design in virtual reality. *Architecture*, 2(1):31–52, 2022.

[22] Enes Yigitbas, Simon Gorissen, Nils Weidmann, and Gregor Engels. Collaborative software modeling in virtual reality. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 261–272. IEEE, 2021.

[23] Diogo Amaral, Gil Domingues, Joao Pedro Dias, Hugo Sereno Ferreira, Ademar Aguiar, and Rui Nóbrega. Live software development environment for java using virtual reality. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering*, 2019.

[24] Nikolai Suslov. Livecoding. space: Towards p2p collaborative live programming environment for webxr. In *Proceedings of the Fourth International Conference on Live Coding (ICLC 2019), Medialab Prado, Madrid, Spain, http://iclc. livecodenetwork. org/2019/papers/paper133. pdf*, 2019.

[25] *Implementing Speech Recognition in Virtual Reality*, volume Volume 1: 22nd Computers and Information in Engineering Conference of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 09 2002.

[26] Nenny Anggraini, Angga Kuniawan, Luh Wardhani, and Nashrul Hakiem. Speech recognition application for the speech impaired using the android-based google cloud speech api. *Telkomnika (Telecommunication Computing Electronics and Control)*, 16:2733–2739, 12 2018.

[27] J. A. Wagner Filho, C.M.D.S. Freitas, and L. Nedel. Virtualdesk: A comfortable and efficient immersive information visualization approach. *Computer Graphics Forum*, 37(3):415–426, 2018.

[28] Martin Schrepp, Andreas Hinderks, and Jörg Thomaschewski. Design and evaluation of a short version of the user experience questionnaire (ueq-s). *International Journal of Interactive Multimedia and Artificial Intelligence*, 4:103, 01 2017.