VR-Git: Git Repository Visualization and Immersion in Virtual Reality

Roy Oberhauser[0000-0002-7606-8226]
Computer Science Dept.
Aalen University
Aalen, Germany
e-mail: roy.oberhauser@hs-aalen.de

***Abstract*—The increasing demand for software functionality necessitates an increasing amount of program source code that is retained and managed in version control systems, such as Git. As the number, size, and complexity of Git repositories increases, so does the number of collaborating developers, maintainers, and other stakeholders over a repository's lifetime. In particular, visual limitations of Git tooling hampers repository comprehension, analysis, and collaboration across one or multiple repositories with a larger stakeholder spectrum. This paper contributes VR-Git, a Virtual Reality (VR) solution concept for visualizing and interacting with Git repositories in VR. Our prototype realization shows its feasibility, and our evaluation results based on a case study show its support for repository comprehension, analysis, and collaboration via branch, commit, and multi-repository scenarios.**

Keywords – Git; virtual reality; visualization; version control systems; software configuration management.

I. INTRODUCTION

In this digitalization era, the global demand for software functionality is increasing across all areas of society, and with it there is a correlating necessity for storing and managing the large number of underlying program source code files that represent the instructions inherent in software. Program source code is typically stored and managed in repositories within version control systems, currently the most popular being Git. Since these repositories are often shared, various cloud-based service providers offer Git functionality, including GitHub, BitBucket, and GitLab. GitHub reports over 305m repositories [1] with over 91m users [2]. Even within a single company, the source code portfolio can become very large, as exemplified with the over 2bn Lines Of Code (LOC) accessed by 25k developers at Google [3]. Over 25m professional software developers worldwide [4] continue to add source code to private and public repositories.

To gain insights into these code repositories, various command-line, visual tools, and web interfaces are provided. Yet, repository analysis can be challenging due to the potentially large number of files involved, and the added complexity of branches, commits, and users involved over the history of a repository. Furthermore, the analysis can be hampered by the limited visual space available for analysis. It can be especially difficult for those stakeholders unfamiliar with a repository, or for collaborating with stakeholders who may not be developers but have a legitimate interest in the code development. Possible scenarios include someone transferred to the development team (ramp-up), joining an open-source code project, quality assurance activities, forensic or intellectual property analysis, maintenance activities, defect or resolution tracking, repository fork analysis, etc.

Virtual Reality (VR) is a mediated visual environment which is created and then experienced as telepresence by the perceiver. VR provides an unlimited immersive space for visualizing and analyzing models and their interrelationships simultaneously in a 3D spatial structure viewable from different perspectives. As repository models grow in size and complexity, an immersive digital environment provides additional visualization capabilities to comprehend and analyze code repositories and include and collaborate with a larger spectrum of stakeholders.

As to our prior work with VR for software engineering, VR-UML [5] provides VR-based visualization of the Unified Modeling Language (UML) and VR-SysML [6] for Systems Modeling Language (SysML) diagrams. This paper contributes VR-Git, a solution concept for visualizing and interacting with Git repositories in VR. Our prototype realization shows its feasibility, and a case-based evaluation provides insights into its capabilities for repository comprehension, analysis and collaboration.

The remainder of this paper is structured as follows: Section 2 discusses related work. In Section 3, the solution concept is described. Section 4 provides details about the realization. The evaluation is described in Section 5 and is followed by a conclusion.

II. RELATED WORK

With regard to VR-based Git visualization, Bjørklund [7] used a directed acyclic graph visualization in VR using the Unreal Engine, with a backend using NodeJS, Mongoose, and ExpressJS, with SQLite used to store data. GitHub Skyline [8] provides a VR Ready 3D contribution graph as an animated skyline that can be annotated.

For non-VR based Git visualization, RepoVis [9] provides a comprehensive visual overview and search facilities using a 2D JavaScript-based web application and Ruby-based backend with a CouchDB. Githru [10] utilizes graph reconstruction, clustering, and context-preserving squash merge to abstract a large-scale commit graph, providing an interactive summary view of the development history. VisGi [11] utilizes tagging to aggregate commits for a coarse group graph, and Sunburst Tree Layout diagrams to visualize group contents. It is interesting to note that the paper states "showing all groups at once overloads the available display space,

making any two-dimensional visualization cluttered and uninformative. The use of an interactive model is important for clean and focused visualizations." UrbanIt [12] utilizes an iPad to support mobile Git visualization aspects, such as an evolution view. Besides the web-based visualization interfaces of Git cloud providers, various desktop Git tools, such as Sourcetree and Gitkracken, provide typical 2D branch visualizations.

In contrast, VR-Git maps familiar 2D visual Git constructs and commit content to VR to make its usage relatively intuitive without training. In contrast to other approaches that apply clustering, aggregating, merging, metrics, or data analytics, our concept preserves the chronological sequence of commits and retains their content details in support of practical analysis for Software Engineering (SE) tasks. To reduce visual clutter, detailed informational aspects of an element of interest can be obtained via the VR-Tablet. By not storing the data in a database, it avoids issues regarding storage format, transformation, and synchronization.

III. Solution Concept

Our VR-Git solution concept is shown relative to our other VR solutions in Figure 1. VR-Git is based on our generalized VR Modeling Framework (VR-MF) (detailed in [13]). VR-MF provides a VR-based domain-independent hypermodeling framework addressing four aspects requiring special attention when modeling in VR: visualization, navigation, interaction, and data retrieval. Our VR-SE area includes VR-Git and the aforementioned VR-UML [5] and VR-SysML [6]. Since Enterprise Architecture (EA) can encompass SE models and development and be applicable for collaboration in VR. Our other VR modeling solutions in the EA area include: VR-EA [13] for visualizing EA ArchiMate models; VR-ProcessMine [14] for process mining and analysis; and VR-BPMN [15] for Business Process Modeling Notation (BPMN) models. VR-EAT [16] integrates the EA Tool (EAT) Atlas to provide dynamically-generated EA diagrams, while VR-EA+TCK [17] integrates Knowledge Management Systems (KMS) and/or Enterprise Content Management Systems (ECMS).

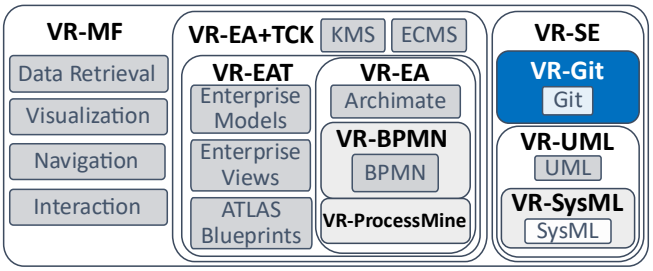

Figure 1. Conceptual map of our various VR solution concepts.

In support of our view that an immersive VR experience can be beneficial for a software analysis, Müller et al. [18] compared VR vs. 2D for a software analysis task, finding that VR does not significantly decrease comprehension and analysis time nor significantly improve correctness (although fewer errors were made). While interaction time was less efficient, VR improved the user experience, was more motivating, less demanding, more inventive/innovative, and more clearly structured.

A. Visualization in VR

A hyperplane is used to intuitively represent and group the commits related to a repository. Each commit is then represented by a vertical *commit plane*. These commit planes are then sequenced chronologically on the hyperplane as a set of planes. Since VR space is unlimited, we can thus convey the sequence of all commits in the repository. Each 2D plane then represents each file involved in that commit as a tile. These are then colored to be able to quickly determine what occurred. Green indicates a file was added, red a file removed, and blue that a file was modified. On the left side of the hyperplane, a transparent *branch plane* (branch perspective) perpendicular to the hyperplane and the commit planes depicts branches as an acyclic colored graph to indicate which branch is involved with a commit. This allows the user to travel down that side to follow a branch, see to which branch any commit is related, and to readily detect merges. In accordance, the commit planes are also slightly offset in height since they dock to a branch, thus "deeper" or "higher" commits indicate how close or far they were relatively from the main branch. Via the anchor, commit planes can be manually collapsed (hidden), expanded, or moved to, for example, compare one commit with another side-by-side. In order to view the contents of a file, when a file tile is selected, a *content plane* (i.e., code view) extends above the commit plane to display the file contents.

B. Navigation in VR

One navigation challenge arising from the immersion VR offers is supporting intuitive spatial navigation while reducing potential VR sickness symptoms. Thus, we incorporate two navigation modes in our solution concept: the default uses gliding controls for fly-through VR, while teleporting instantly places the camera at a selected position either via the VR controls or by selection of a commit in our VR-Tablet. While teleporting is potentially disconcerting, it may reduce the likelihood of VR sickness induced by fly-through for those prone to it.

C. Interaction in VR

As VR interaction has not yet become standardized, in our concept we support user-element interaction primarily through VR controllers and a *VR-Tablet*. The VR-Tablet is used to provide detailed context-specific element information based on VR object selection, menu, scrolling, field inputs, and other inputs. It includes a *virtual keyboard* for text entry via laser pointer key selection. As another VR interaction element, we provide the aforementioned corner *anchor sphere* affordance, that supports moving, collapsing / hiding, or expanding / displaying hyperplanes or commit planes.

IV. Realization

The logical architecture for our VR-Git prototype realization is shown in Figure 2. Basic visualization, navigation, and interaction functionality in our VR prototype is implemented with Unity 2020.3 and the OpenVR XR

Plugin 1.1.4, shown in the Unity block (top left, blue). Scripts utilize Libgit2Sharp [19] to access the Git commit history of one or more repositories from within Unity. To avoid redundancy, only realization aspects not explicitly mentioned in the evaluation are described in this section.

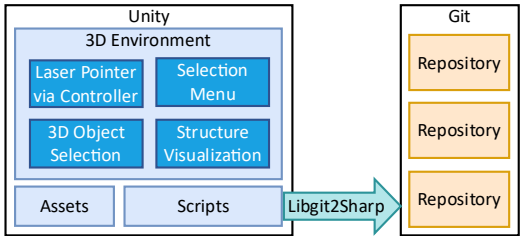

Figure 2. VR-Git logical architecture.

An anchor (ball) is placed on one corner of a hyperplane and is an affordance in order to move or expand/collapse an entire hyperplane (see Figure 3). The anchors are also placed at the left bottom corner of all commit planes and colored and aligned with the branch with which they are associated.

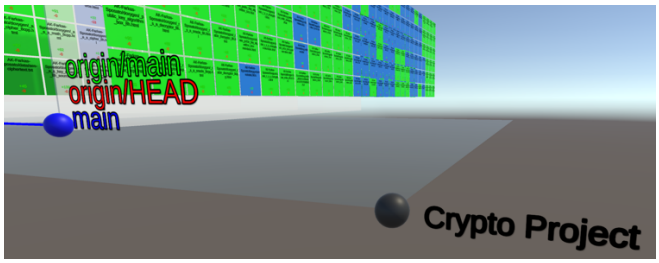

Figure 3. Hyperplane anchor for a repository.

Projects can be selected via the VR-Tablet and provide a teleporting capability to the hyperplane, as shown in Figure 4.

Figure 4. Project list in VR-Tablet.

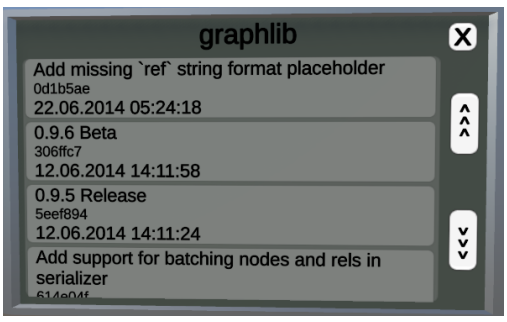

Figure 5. Git commit messages in VR-Tablet.

Figure 5 shows a list of Git commit messages including their commit ID (Secure Hash Algorithm 1 (SHA-1)) and the date and timestamp in the VR-Tablet. This can also be used to teleport to a specific commit plane.

V. Evaluation

The evaluation of our solution concept is based on the design science method and principles [20], in particular, a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). We use a case study based on common Git repository comprehension and analysis scenarios: branch analysis, commit analysis, and multi-repository analysis. Various Git repositories were used to evaluate the prototype.

A. Branch Analysis Scenario

To support branch analysis, at the front of the hyperplane oriented to the left side, an invisible branch graph plane is rendered perpendicular to the hyperplane and a color-coded list of all the branches can be seen next to the first commit plane (see Figure 6). These colored labels can be used for orientation. By selecting a branch label, the user can be teleported to the first commit of that branch. The branches could also be referenced in the VR-Tablet in case one forgets, and can be used for teleporting as well. We chose to repeat the branch labels throughout the graph to reduce the textual visual clutter.

Figure 6. VR-Git branch overview.

The branch perspective of the hyperplane (its left side) shows a contiguous color-coded graph of the branches as shown in Figure 7, with commit plane heights offset based on the branch to which they are associated. This can provide a quick visual cue as to how relatively close or far the commit is from the main branch. Figure 8 shows a merge of two branches.

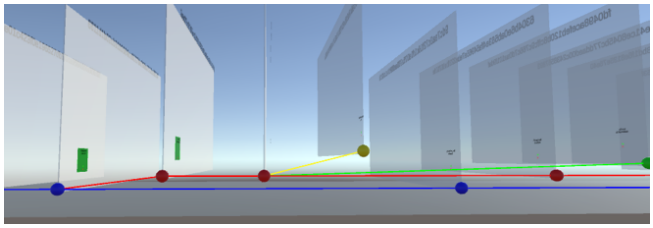

Figure 7. VR-Git branch tree graph.

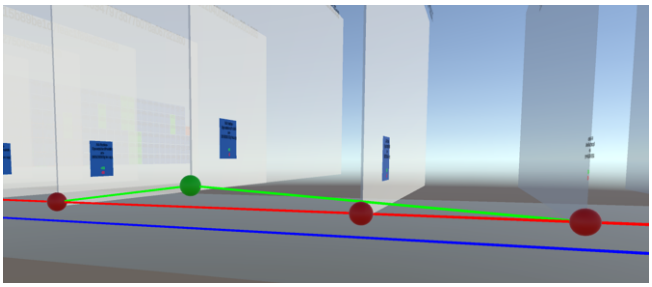

Figure 8. Branch merge.

As a reference, in Figure 9 we see the terminal output in Git. In comparison, VR-Git provides equivalent branch information, providing the labels and also using different branch colors and spatial offsetting to indicate which branch a commit relates to. To reduce visual clutter, commit messages are not shown on the planes, but rather the VR-Tablet (Figure 5) includes the commit messages, timestamp, and commit ID (SHA). Note that the commit ID is also shown at the top of each commit plane to both differentiate and identify them.

```
* cdfb4e2 (HEAD -> development, origin/development) fixed header color
* 9ec0274 modified settings page
* 24ec5ab app icon improvement, design improvements
* fb22d80 updated icons
* 5353133 design improvements
* 76d23be #3 display number of tickets
* cbd1866 #6 changed font-size
* 9ec697f fixed package.json
* 5c4d842 improved refresh after redeem
* 144f743 added new icons
*   1fdf948 (tag: 0.1.3-beta, origin/master, origin/HEAD, master) Merge pull request #1 from Jay895/new-ui
|\
| * 357ff25 added ticket variation to list
| * 98b619b regenerated icons, finished initialization
| * fc2a8b3 updated app id
| * fb67deb new app icon
| * 4ddb897 new app name
| * 8e414f3 changed version number
| * 04b0936 added initial screen and some ui changes
|/
* d4d3cce added settings, multi checkin, ticket overview
* 5fde2cf fixed store access, use auth from store
*   80e5bc6 Merge branch 'master' of https://github.com/Jay895/pretix-checkin
|\
| * ff382fe Delete .DS_Store
| * a913fa7 Update .gitignore
* | 3545d0f added webpack config
* | 43ed54b added scan functions, search function
* | 8f83e05 added scan plugin, API functions, store
* | 06d49eb iOS permissions
* | ccd67c5 added packages
|/
* 954e89f initial commit
```

Figure 9. Example Git log terminal output

B. *Commit Analysis Scenario*

Git commits are a snapshot of a repository. In a typical commit analysis, a stakeholder is interested in what changed with a commit, i.e., what files were added, deleted, or modified. To readily indicate this, as shown in Figure 10, tiles labeled with the file pathname are placed on the commit plane to represent changed files, with colors of green representing files added, blue changed, and red for deleted. In addition, the number of lines of text are shown at the bottom or a tile, with positive numbers in green indicating the number of lines added, and negative red values below it for the lines removed.

Figure 11 shows how commits that affect a large number of files can be readily determined. This can be helpful in analysis to quickly hone in on commit with the greatest impacts.

Figure 12 depicts how VR can visually scale with commits affecting a very large number of files. As we see, there is no issue displaying the data, and VR navigation and the VR-Tablet can be used to analyze the commit further.

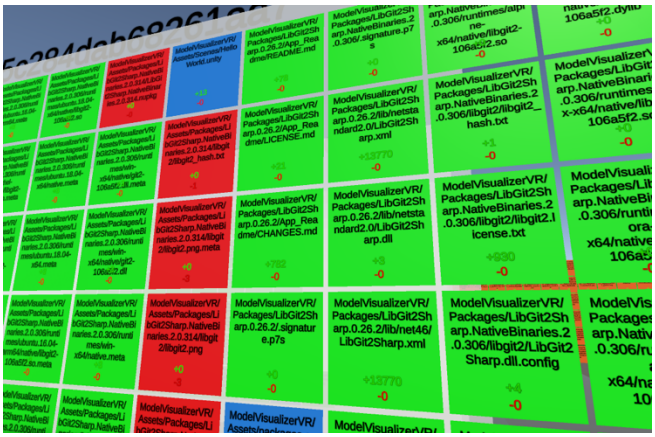

Figure 10. Commit files added (green), changed (blue), deleted (red); number of lines affected indicated in each tile at the bottom.

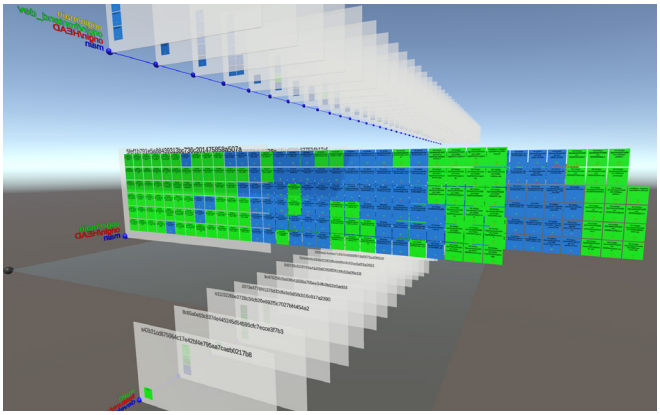

Figure 11. VR-Git showing commits affecting a large number of files.

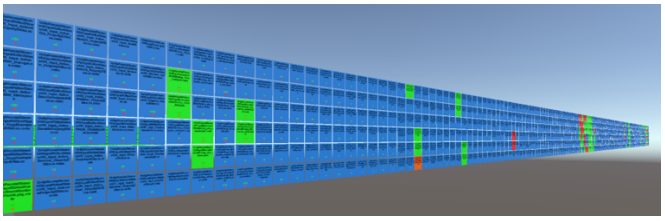

Figure 12. VR-Git commit visual scaling example for a very large file set.

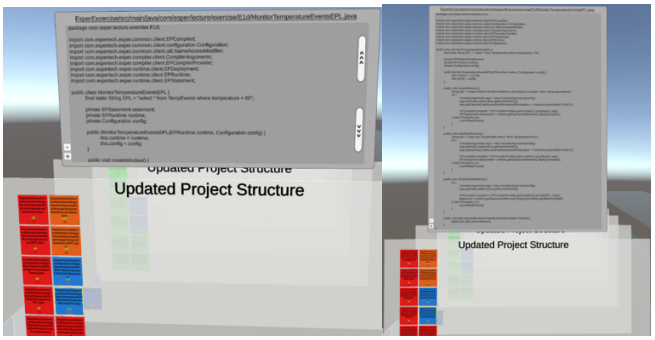

Figure 13. Code View: collapsed and scrollable (left) and expanded (right).

By selecting a specific tile (file), a file *contents plane* (i.e., code view), shown in Figure 13, pops up above the plane displaying the contents of that file for that commit. Since file contents can be too lengthy and wide for practical depiction in our VR-Tablet, we chose to display the plane above the commit plane, providing a clear association. The contents are initially scrollable, and can be expanded with the plus icon to show the entire file contents if desired. Since VR is not limited, one can navigate by moving the VR camera to any part of the code plane to see the code there.

C. Multi-Repository Analysis Scenario

To support multiple repository analysis, hyperplanes are used to represent each separate repository. Via the anchors, these can be placed where appropriate for the user. Figure 14 shows how branch and commit comparisons can be made from the branch perspective, with visual cues being offered by the tiles. Here, one can see how the branches developed with their commits.

On the other hand, Figure 15 shows a larger visual depiction of three repositories and readily indicates which ones involved more commits, and via the extended commit planes where large commits were involved.

D. Discussion

In summary, the evaluation showed that these primary comprehension and analysis scenarios were supported by the solution concept and our prototype. Branch comprehension and analysis were supported via the branch plane. Commit comprehension and analysis were supported via the commit planes, which readily showed the number of files involved in a commit (based on the number of tiles) and via their color if they were added, removed, or changed. The metrics in each tile show the number of lines affected. Multi-repository analysis showed the potential of VR to display and compare multiple repositories, where the limitless space can be used to readily focus and hone-in on the areas of interest or differences between repositories. This type of visual, immersive multi-repository analysis could support fork analysis, intellectual property analysis and tracking, forensic analysis, etc.

VI. Conclusion

VR-Git contributes an immersive software repository experience for visually depicting and navigating repositories in VR. It provides a convenient way for stakeholders who may not be developers yet have a legitimate interest in the code development to collaborate. This can further the onboarding of maintenance or quality assurance personnel. The solution concept was described and a VR prototype demonstrated its feasibility. Based on our VR hyperplane principle, repositories are enhanced with 3D depth and color. Interaction is supported via a virtual tablet and keyboard. The unlimited space in VR facilitates the depiction and visual navigation of large repositories, while relations within and between artifacts, groups, and versions can be analyzed. Furthermore, in VR additional related repositories or models can be visualized and analyzed simultaneously and benefit more complex collaboration and comprehension. The sensory immersion of VR can support task focus during comprehension and increase enjoyment, while limiting the visual distractions that typical 2D display surroundings incur. The solution concept was evaluated with our prototype using a case study based on typical Git comprehension and analysis scenarios: branch analysis, commit analysis, and multi-repository analysis. The results indicate that VR-Git can support these analysis scenarios and thus provide an immersive collaborative environment to involve and include a larger stakeholder spectrum in understanding Git repository development.

Future work includes support for directly invoking and utilizing Git within VR, including further visual constructs, integrating additional informational and tooling capabilities, and conducting a comprehensive empirical study.

Acknowledgment

The authors would like to thank Jason Farkas and Marie Bähre for their assistance with the design, implementation, and evaluation.

References

[1] GitHub repositories [Online]. Available from: https://web.archive.org/web/20220509204719/https://github.com/search 2022.07.25

[2] GitHub users [Online]. Available from: https://web.archive.org/web/20220529205506/https://github.com/search 2022.07.25

[3] C. Metz, "Google Is 2 Billion Lines of Code—And It's All in One Place," 2015. [Online]. Available from: http://www.wired.com/2015/09/ google-2-billion-lines-codeand-one-place/ 2022.07.25

[4] Evans Data Corporation. [Online]. Available from: https://evansdata.com/press/viewRelease.php?pressID=293 2022.07.25

[5] R. Oberhauser, "VR-UML: The unified modeling language in virtual reality – an immersive modeling experience," International Symposium on Business Modeling and Software Design, Springer, Cham, 2021, pp. 40-58.

[6] R. Oberhauser, "VR-SysML: SysML Model Visualization and Immersion in Virtual Reality," International Conference of Modern Systems Engineering Solutions (MODERN SYSTEMS 2022), IARIA, 2022, pp. 59-64.

[7] H. Bjørklund, "Visualisation of Git in Virtual Reality," Master's thesis, NTNU, 2017.

[8] GitHub Skyline [Online]. Available from: https://skyline.github.com 2022.07.25

[9] J. Feiner and K. Andrews, "Repovis: Visual overviews and full-text search in software repositories," In: 2018 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2018, pp. 1-11.

[10] Y. Kim et al., "Githru: Visual analytics for understanding software development history through git metadata analysis," IEEE Transactions on Visualization and Computer Graphics, 27(2), IEEE, 2020, pp.656-666.

[11] S. Elsen, "VisGi: Visualizing git branches," In 2013 First IEEE Working Conference on Software Visualization, IEEE, 2013, pp. 1-4.

[12] A. Ciani, R. Minelli, A. Mocci, and M. Lanza, "UrbanIt: Visualizing repositories everywhere," In 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, 2015, pp. 324-326.

[13] R. Oberhauser and C. Pogolski, "VR-EA: Virtual Reality Visualization of Enterprise Architecture Models with ArchiMate and BPMN," In: Shishkov, B. (ed.) BMSD 2019. LNBIP, vol. 356, Springer, Cham, 2019, pp. 170–187.

[14] R. Oberhauser, "VR-ProcessMine: Immersive Process Mining Visualization and Analysis in Virtual Reality," The Fourteenth International Conference on Information, Process, and Knowledge Management (eKNOW 2022), IARIA, 2022, pp. 29-36.

[15] R. Oberhauser, C. Pogolski, and A. Matic, "VR-BPMN: Visualizing BPMN models in Virtual Reality," In: Shishkov, B. (ed.) BMSD 2018. LNBIP, vol. 319, Springer, Cham, 2018, pp. 83–97. https://doi.org/10.1007/978-3-319-94214-8_6

[16] R. Oberhauser, P. Sousa, and F. Michel, "VR-EAT: Visualization of Enterprise Architecture Tool Diagrams in Virtual Reality," In: Shishkov B. (eds) Business Modeling and Software Design. BMSD 2020. LNBIP, vol 391, Springer, Cham, 2020, pp. 221-239. https://doi.org/10.1007/978-3-030-52306-0_14

[17] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EA+TCK: Visualizing Enterprise Architecture, Content, and Knowledge in Virtual Reality," In: Shishkov, B. (eds) Business Modeling and Software Design. BMSD 2022. Lecture Notes in Business Information Processing, vol 453, pp. 122-140. Springer, Cham. https://doi.org/10.1007/978-3-031-11510-3_8

[18] R. Müller, P. Kovacs, J. Schilbach, and D. Zeckzer, "How to master challenges in experimental evaluation of 2D versus 3D software visualizations," In: 2014 IEEE VIS International Workshop on 3Dvis (3Dvis), IEEE, 2014, pp. 33-36

[19] Libgit2Sharp. [Online]. Available from: https://github.com/libgit2/libgit2sharp 2022.07.25

[20] A.R. Hevner, S.T. March, J. Park, and S. Ram, "Design science in information systems research," MIS Quarterly, 28(1), 2004, pp. 75-105

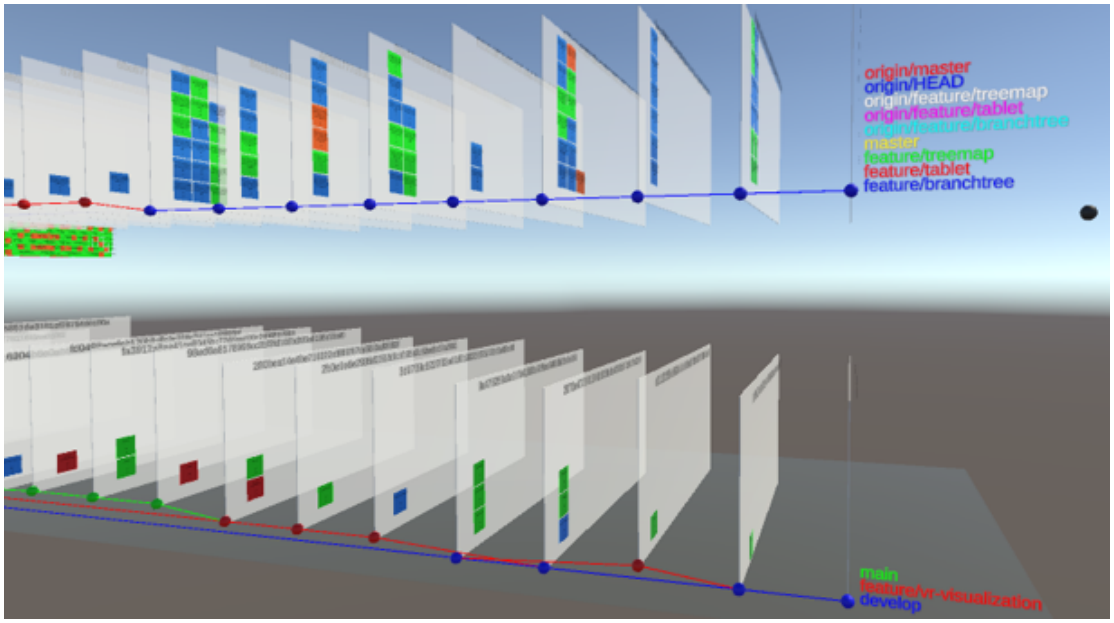

Figure 14. Dual repository comparison with a branch focus.

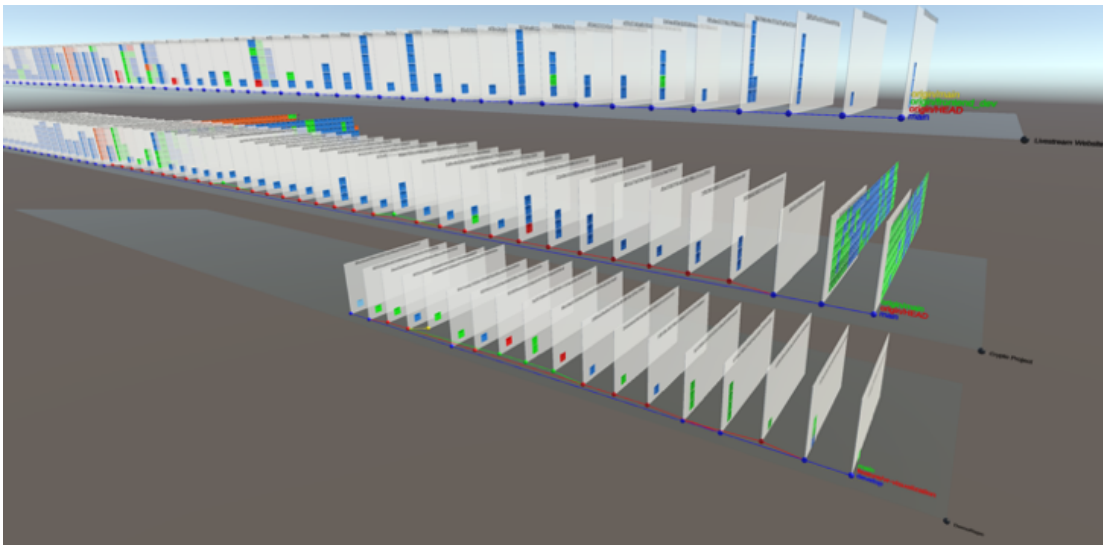

Figure 15. Multiple repositories from a wide perspective.