# VR-GitCity: Immersively Visualizing Git Repository Evolution Using a City Metaphor 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 command line or two-dimensional graphical Git tooling can hamper repository comprehension, analysis, and collaboration across one or multiple repositories when a larger stakeholder spectrum is involved. This is especially true for depicting repository evolution over time. This paper contributes VR-GitCity, a Virtual Reality (VR) solution concept for visualizing and interacting with Git repositories in VR. The evolution of the code base is depicted via a 3D treemap utilizing a city metaphor, while the commit history is visualized as vertical planes. Our prototype realization shows its feasibility, and our evaluation results based on a case study show its depiction, comprehension, analysis, and collaboration capabilities for evolution, branch, commit, and multi-repository analysis scenarios.**

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

## I. INTRODUCTION

This paper is an extended version of our original paper on VR-Git [1] and extends our solution to VR-GitCity, which incorporates a city metaphor.

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 [2] with over 91m users [3]. 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 [4]. Over 25m professional software developers worldwide [5] 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 insights to 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. Furthermore, while repositories are dynamic and retain historical information, it nevertheless can be challenging to readily convey these aspects intuitively using conventional Git tooling. Especially as the size of a repository grows in number of elements (subfolders and files), it becomes difficult to comprehend the "big picture" as to how it has been evolving, which areas were the focus for changing or adding code when, and depicting the final state.

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 [6] provides VR-based visualization of the Unified Modeling Language (UML) and VR-SysML [7] for Systems Modeling Language (SysML) diagrams. Our original paper described VR-Git [1], a solution concept for visualizing and interacting with Git repositories in VR. This paper contributes VR-GitCity, which extends our VR-Git visualization capabilities to incorporate a 3D treemap using a city metaphor to convey repository evolution of relative files sizes in Lines of Code (LOC), while using vertical planes for branch and commit analysis. 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 [8] 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 [9] provides a VR Ready 3D contribution graph as an animated skyline that can be annotated.

For non-VR based Git visualization, RepoVis [10] provides a comprehensive visual overview and search facilities using a 2D JavaScript-based web application and Ruby-based backend with a CouchDB. Githru [11] 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 [12] 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 [13] 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.

The city metaphor is a well-known software visualization paradigm. An early paper to apply it was the File System Navigator (FSN) [14], and although it did not explicitly use the word 'city,' it nevertheless used a landscape paradigm with a network of roads, buildings, and towns. MediaMetro [15] applies the metaphor to media documents. CodeCity [16] is a 3D software visualization approach based on a city metaphor with the Moose reengineering framework implemented in SmallTalk. In this context, Buildings represent classes, districts represent packages, and visible properties depict selected metrics. ExplorViz [17] uses a city metaphor for live trace exploration, implemented in JavaScript as a browser-based WebVR application using Oculus Rift together with Microsoft Kinect for gesture recognition. Code2City$_{VR}$ [18], which is a VR implementation of the previously mentioned CodeCity [16], focuses on metrics and smells for Java code.

In contrast to the above work, VR-GitCity utilizes a city metaphor for Git repositories in VR, depicting their dynamic evolution with regard to LOC size, while mapping 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.

## 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 [14]). 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-GitCity (a superset of our VR-Git) and the aforementioned VR-UML [6] and VR-SysML [7]. 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 [19] for visualizing EA ArchiMate models; VR-ProcessMine [20] for process mining and analysis; and VR-BPMN [21] for Business Process Modeling Notation (BPMN) models. VR-EAT [22] integrates the EA Tool (EAT) Atlas to provide dynamically-generated EA diagrams, while VR-EA+TCK [23] integrates Knowledge Management Systems (KMS) and/or Enterprise Content Management Systems (ECMS).

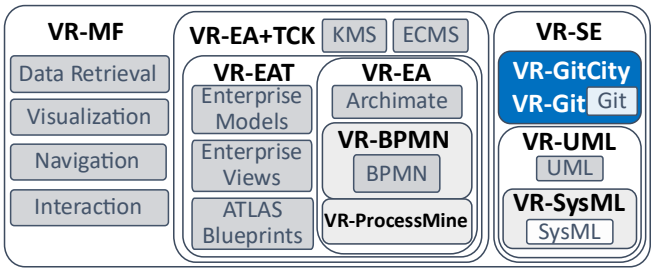


Figure 1. Conceptual map of our various VR solution concepts.

In support of our view that an immersive VR experience can be beneficial for a software analysis, Müller et al. [24] 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 relates, and to readily detect merges. Accordingly, the commit

planes are slightly offset vertically, as they dock to a branch, thus "deeper" or "higher" commits indicate how close or far they relatively are 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*

A navigation challenge resulting from VR immersion is supporting intuitive spatial navigation while reducing potential VR sickness symptoms. We thus incorporate two navigation modes in our solution concept: gliding controls for fly-through VR (default), 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 vertical commit planes.

## IV. Realization

The logical architecture for our VR-GitCity 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 [25] to access the Git commit history of one or more repositories from within Unity. Thus, data about the repository is not stored in a separate database but accessed on-the-fly, avoiding synchronization, data-loss, storage format, transformation, and other issues. Note that only realization aspects not explicitly mentioned in the evaluation are described in this section to reduce redundancy.

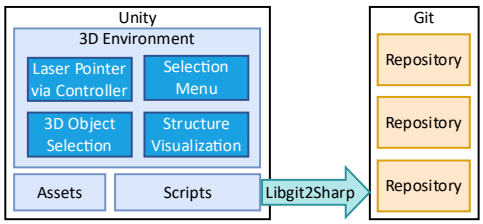


Figure 2. VR-GitCity logical architecture.

Internally, a tree data structure is used to represent a repository. Directory nodes represent a folder characterized by a name and path, and may contain files or subfolders (children). File nodes keep references to the parent directory node that contains them. Dictionaries are used to track the state and the size of a node, with the commit SHA serving as a key and used to identify the time of a change. With each SHA commit stored in the dictionary, the difference via a file compare is performed and the delta as Lines of Code (LOC) is retained.

For our city metaphor visualization, a directory node is a cuboid with equal x and z (depth) lengths. A bit matrix is used to distribute any children (folders or files) as compactly as possible, and these are stacked as cream-colored common height blocks in the y dimension (height). To follow the city metaphor, a building should be used to represent modules or in our case files. However, we wanted to primarily convey the dynamic evolution of the size of repository elements over time, rather than depicting any structural modularization. So, to be intuitive in visually transmitting the size information dynamically over time, we chose to utilize the metaphor of a glass of water and its fullness to represent the size state of any file. However, to not break with the city metaphor, these can be viewed as a glass skyscraper or glass elevator as shown in Figure 3. It is a cuboid in blue glass tone, with the maximum LOC file size ever reached (relative to all others) represented by its height. Its current size is transparent glass with a glass level to show how full it is currently (relative to max), with a more greyish tone above to differentiate anything less than full. The final size is marked by a grey fat slab (like concrete). Selecting a certain commit via the VR-Tablet will depict its changes to the repository by coloring the aqua cuboid green for added files, red for deleted files, and blue for changed files.

As to color choices, transparent (glass) was used as the default for building sides, in order to avoid buildings from hiding objects behind it, to better see the foundation, and for objects involved in a commit (opaque colors), to be more pronounced. This also permits the metaphor of a water glass with its fullness represented by slab levels. However, the building colors could readily be randomly distributed or custom-defined per object or folder by the user via a configuration file or the VR-Tablet. Also, in place of opaque colors, alternatively the border outlines of a building could be color-coded for the commit accordingly.

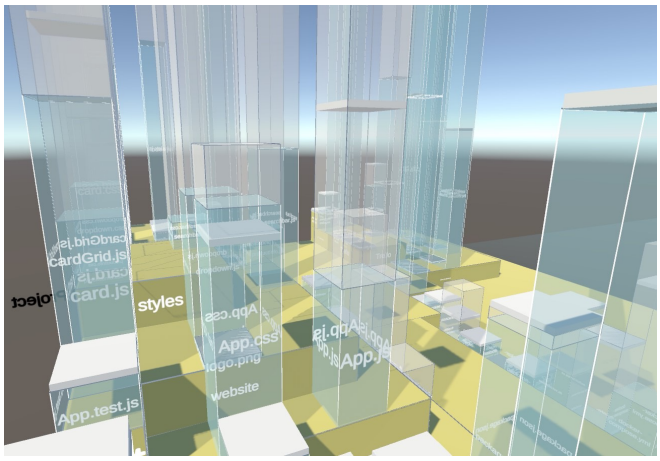


Figure 3. VR-GitCity: files as glass buildings stacked on their containing folders (box height is its all-time max LOC size, aqua slab its current size, and thick grey slab the final size).

The VR-Git view functionality supports detailed commit information visualization, with commits represented on vertical planes. To support interaction on specific commits, 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, as shown in Figure 4. 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.

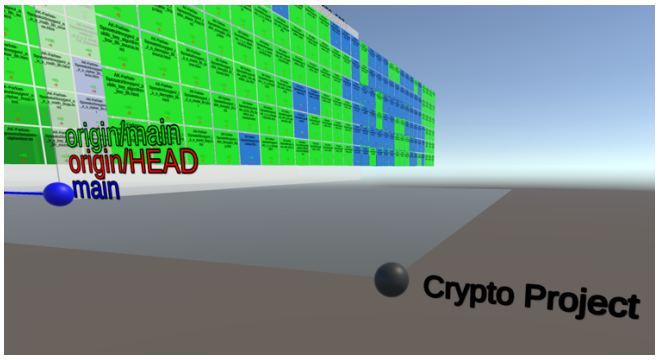


Figure 4. Vertical commit planes on hyperplane with anchor affordances.

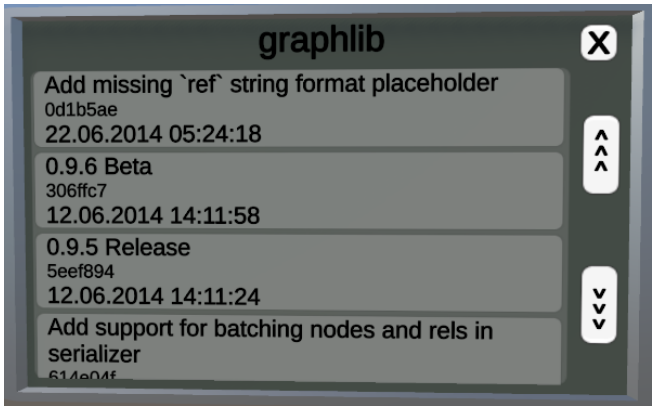


Figure 5. Viewing Git commit messages in VR-Tablet.

To support navigation, projects can be selected via the VR-Tablet and provide a teleporting capability to its project hyperplane or a specific commit plane. 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, as shown in Figure 5.

## V. Evaluation

For the evaluation of our solution concept, we refer to the design science method and principles [26], in particular, a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). For this, we use a case study applying four Git repository scenarios in VR:

A. Repository evolution scenario
B. Branch analysis scenario
C. Commit analysis scenario
D. Multi-Repository Analysis Scenario

### *A. Repository Evolution Scenario*

To support repository evolution comprehension and analysis, the VR-GitCity city metaphor depiction is used. Two Repositories, denoted as R1 and R2, were used as the basis for the scenario depictions. Note that the counting utility ignored file types and code unfamiliar to it.

- R1 consists of approximately 23 (sub)folders, 99 files, and at least 47K LOC (across ~50 files with JSON, HTML, JS, XML, CSS, and Markdown), as can be seen in Figure 6.
- R2 consists of approximately 660 (sub)folders, 2700 files, and at least 123K LOC (across ~377 files with C# and JSON), as shown in Figure 7.

```
      65 text files.
classified 65 files
      65 unique files.
      25 files ignored.

github.com/AlDanial/cloc v 1.90  T=0.14 s (395.2 files/s, 353263.6 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
JSON                            15              0              0          39269
HTML                             3            985             84           6108
JavaScript                      17            131             94            974
XML                              4              2              0            640
CSS                              6            116              6            527
YAML                             3              5              0             56
Bourne Shell                     2             11              7             40
Markdown                         1             32              0             38
Dockerfile                       3             13              0             24
SVG                              1              0              0              1
-------------------------------------------------------------------------------
SUM:                            55           1295            191          47677
```

Figure 6. R1 cloc report.

```
|    2720 text files.
classified 2720 files
Duplicate file check 2720 files (1043 known unique)
    2698 unique files.
    2205 files ignored.

github.com/AlDanial/cloc v 1.90  T=1.10 s (841.0 files/s, 344980.1 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Unity-Prefab                   410             23              0         177356
C#                             344          12318           7747          68221
JSON                            33              2              0          55119
XML                             19             83              1          45408
SVG                             77            138             77           5105
HLSL                            31            931            228           3585
XSD                              5            341             53           2129
Markdown                         4            259              0            683
C# Generated                     4             63             36            368
-------------------------------------------------------------------------------
SUM:                           927          14158           8142         357974
-------------------------------------------------------------------------------
```

Figure 7. R2 cloc report.

In VR, the front and side views of the repository show aligned stacked cream-colored blocks as (sub)folders that provide a single quick overview of the involved (sub)folders and their containing parent, as seen in Figure 8.

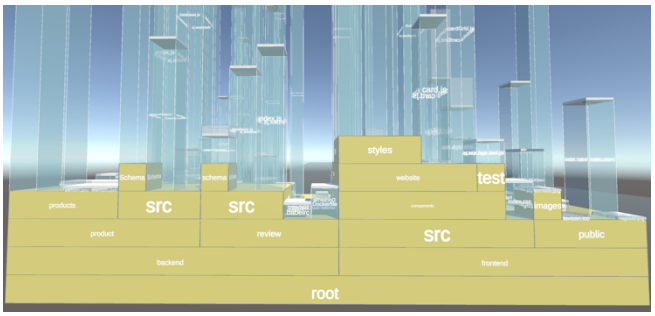


Figure 8. VR-GitCity R1 front view showing aligned stacked (sub)folders.

The deepest path of subfolders is placed on the front, thus the rear view typically shows a declining set of folder blocks with regard to height, as seen in Figure 9. Thus, the greatest depth of folder containment can be readily determined.

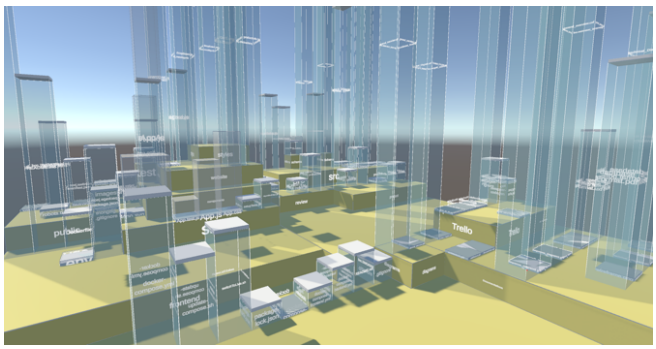

Figure 9. VR-GitCity R1 rear view with declining (sub)folder foundation.

Where applicable, additional parallel subfolders may be viewed from the left or right side based on placement, as seen in Figure 10. A closeup from the rear is shown in Figure 11.

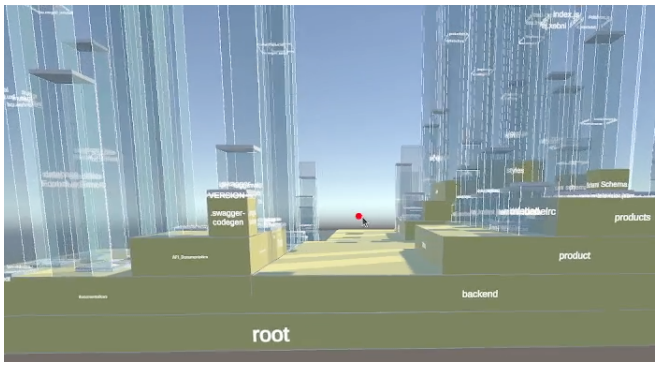


Figure 10. VR-GitCity R1 side view with parallel folder depiction.

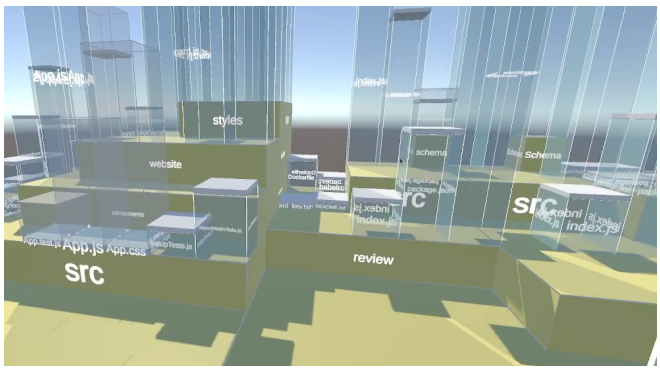


Figure 11. VR-GitCity R1 closeup from the rear.

When conveying the elements affected by a specific commit selected in the VR-Tablet, green is used to indicate that files were added, as shown in Figure 12. Red is used for deleted files, as shown in Figure 13. The glass thin slab level shows the current relative size in LOC compared to other files, with the max size being the cuboid overall height, with a thick grey slab conveying the final file size.

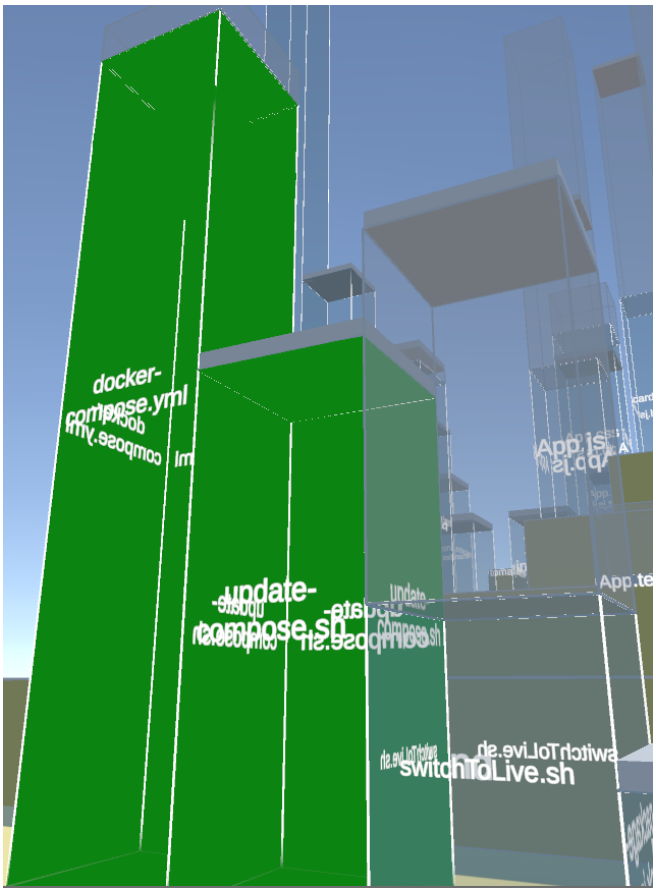


Figure 12. VR-GitCity R1 showing added files in green.

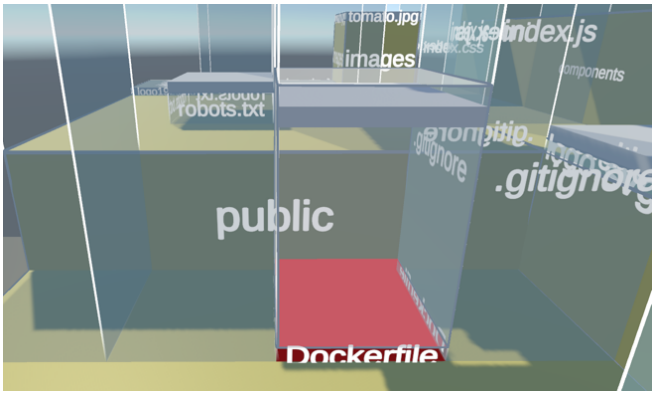


Figure 13. VR-GitCity R1 showing deleted files in red.

Blue is used to convey files that were changed by a commit, as shown in Figure 14. Thus, by scrolling through commits on the VR-Tablet, a dynamic changing picture equivalent to a video of the repository evolution is presented. In portraying the maximum size as well as the end final size, it can be understood to show the evolution of any single element to its maximum as well as end target size, while not forgetting the maximum it once had if it later shrunk (as a type of ghosting) or a file was removed.

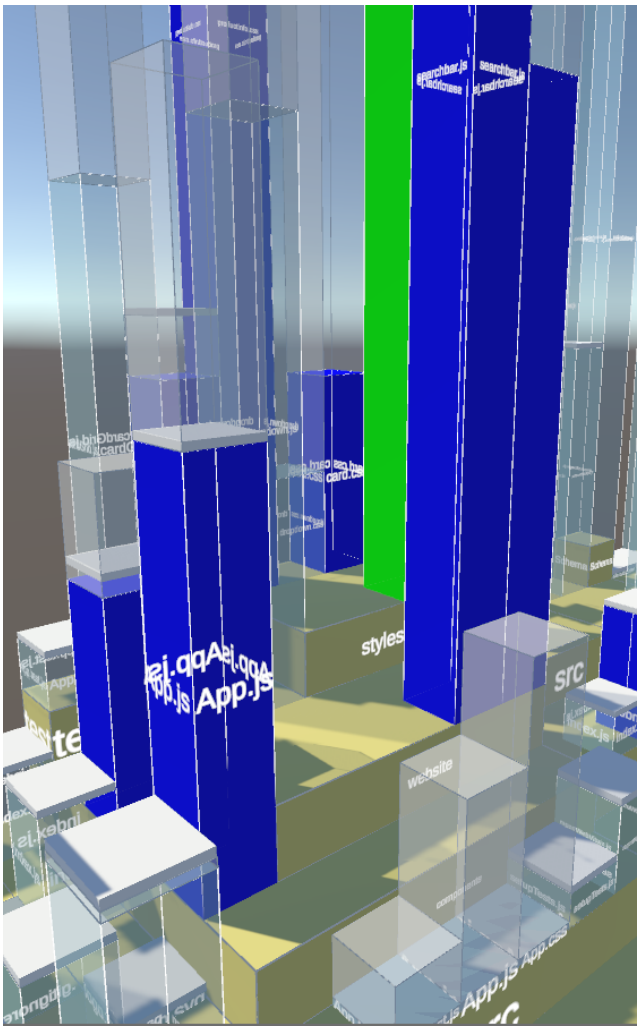


Figure 14. VR-GitCity R1 showing changed files in blue.

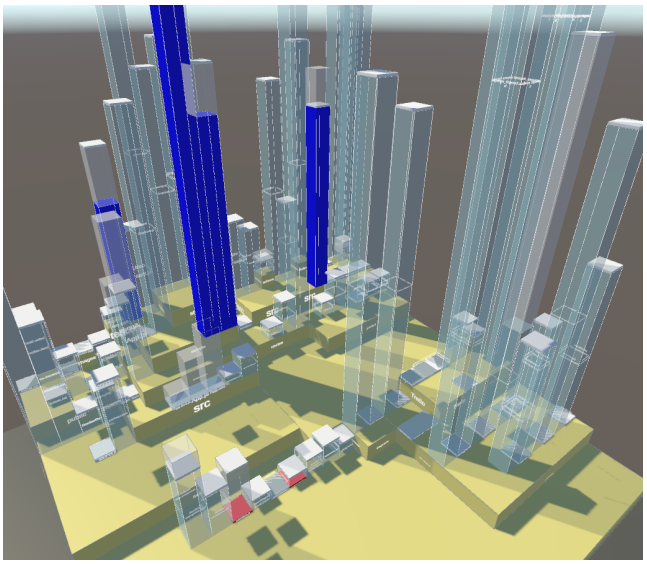

Figure 15. VR-GitCity R1 overview.

An overview of repository R1 can be seen in Figure 15. It visually and immersively conveys the grouping and containment of elements in subfolders, the number of files, the maximum relative sizes achieved, the current size as a fill level, and the affected elements by a specific commit via colors.

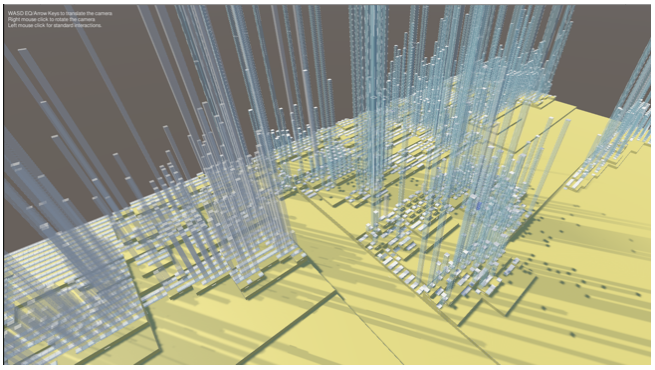

Figure 16. VR-GitCity R2 overview.

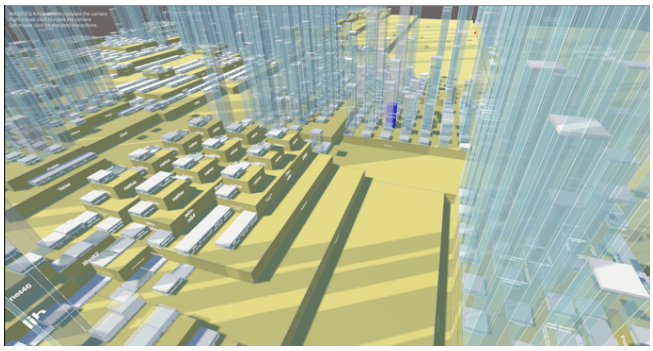

Figure 17. VR-GitCity R2 indicating a changed file in center as blue.

To test the scalability of the concept, the repository R2 was used and is shown in Figure 16 and Figure 17. Note that R2 consists of over 600 subfolders and almost 10K files. While these screenshots are not intended to be legible in this paper, they portray the capability of VR-GitCity to scale to very large repositories and provide a "big picture" of their evolution over time. Up close via immersion, fly-through navigation, selection, and the VR-Tablet, additional detailed information about an element or affected files in a commit can be determined.

### *B. Branch Analysis Scenario*

To support branch analysis, our hyperplane concept with vertical planes is used. To the left side of a hyperplane, 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, seen in Figure 18. 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. We chose not to repeat the branch labels throughout the graph to reduce the textual visual clutter.


Figure 18. 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 19. 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. A merge of two branches is shown in Figure 20.

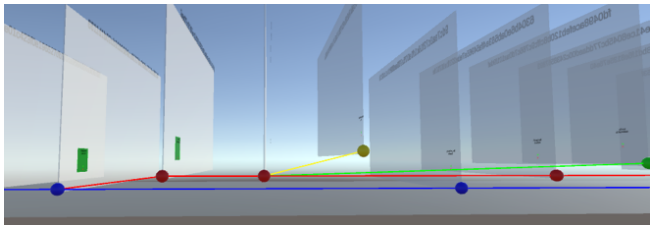

Figure 19. VR-Git branch tree graph.

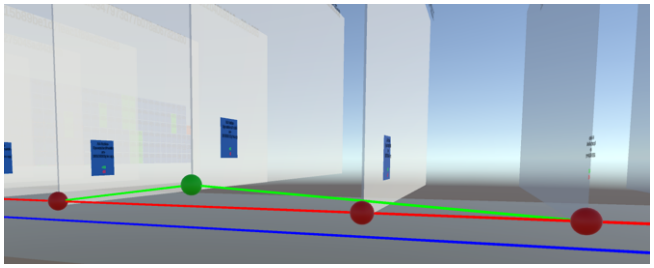

Figure 20. Branch merge.

```
* 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
| * 98b819b regenerated icons, finished initialization
| * fc2a0b3 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 21. Example Git log terminal output

As a reference, the terminal output in Git is shown in Figure 21. In contrast, 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, which includes the commit messages, timestamp, and commit ID (SHA). Note that the commit ID is displayed at the top of each commit plane to both differentiate and identify commits.

### *C. 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, 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. This is shown in Figure 22. 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.

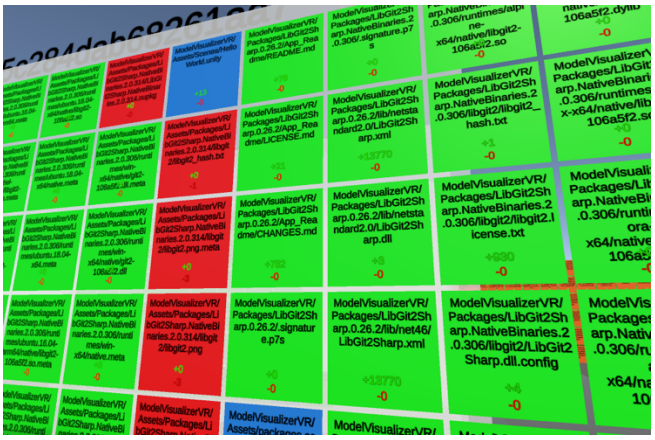

Figure 22. Commit files added (green), changed (blue), deleted (red); number of lines affected indicated in each tile at the bottom.

The ability of VR to visually scale with commits affecting a very large number of files is shown in Figure 23. 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.

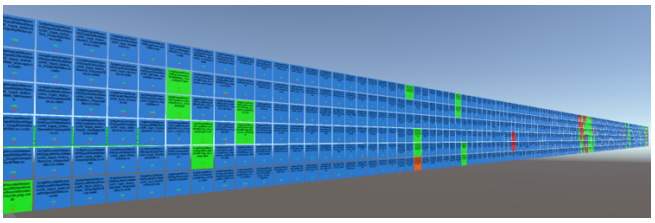

Figure 23. VR-Git commit visual scaling example for a very large file set.

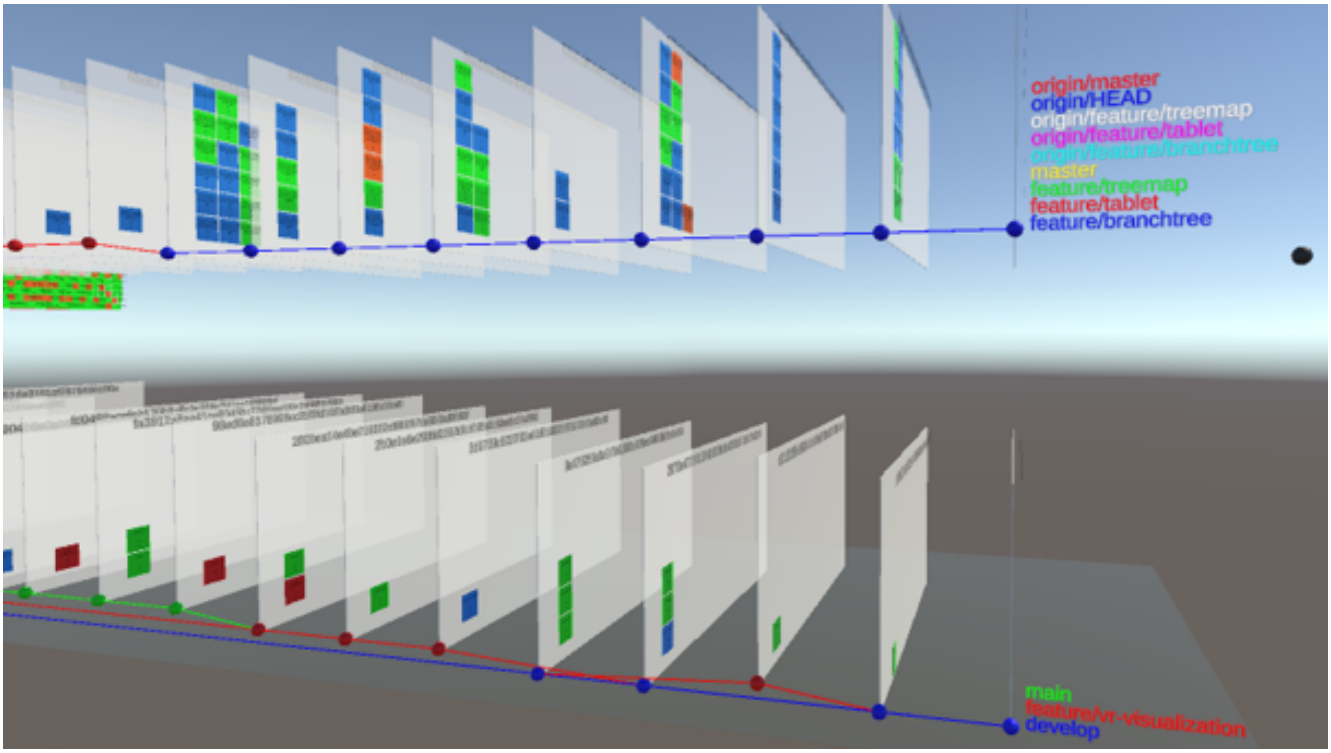


Figure 24. Dual repository comparison with a branch focus.

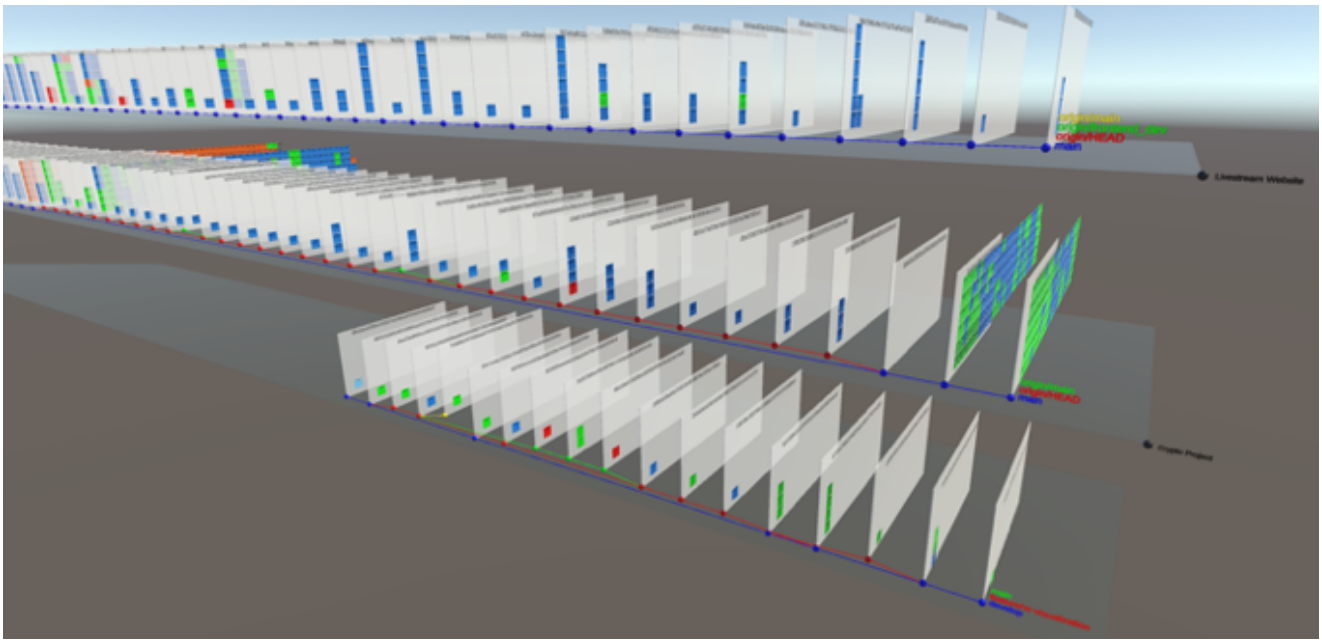

Figure 25. Multiple repositories from a wide perspective.

Commits affecting a large number of files can be readily determined, as seen in Figure 26. This can support analysis to quickly hone in on commits with the greatest impacts.

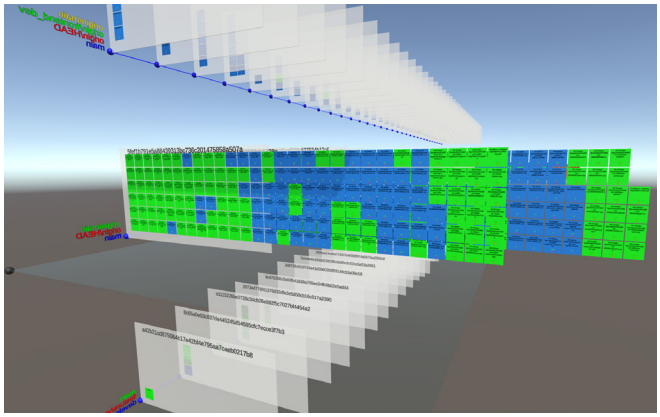

Figure 26. Multiple repositories showing commits affecting a large number of files.

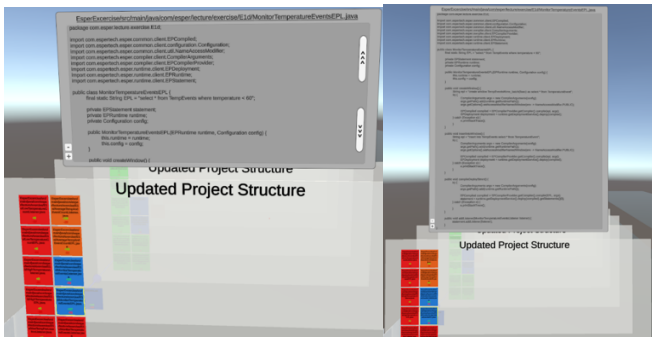


Figure 27. Code View: collapsed and scrollable (left) and expanded (right).

By selecting a specific tile (file), a file contents plane (i.e., code view) pops up displaying the contents of that file for that commit, seen in Figure 27. 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.

### D. *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. Branch and commit comparisons can be made from the branch perspective, with visual cues being offered by the element tiles, as shown in Figure 24. Here, one can see how the branches developed with their commits. A larger visual depiction of multiple repositories is shown in Figure 25. This shows how the VR unlimited space can be used, e.g., to determine which ones involved more commits, or where larger commits with more elements were involved via the extended commit planes.

### E. *Discussion*

In summary, the evaluation showed that VR-GitCity supports comprehension and analysis for key Git scenarios in VR, including a repository's evolution, commits, branches, and scalable multi-repository comparisons. The city metaphor is used to convey the dynamic evolution of a repository visually and immersively, depicting the grouping and containment of elements in subfolders, the number of files, the maximum relative sizes achieved, the current size as a fill level, and colors affected elements of a specific commit. 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-GitCity 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 Nikolas Lindenmeyer, Jason Farkas, and Marie Bähre for their assistance with the design, implementation, figures, and evaluation.

## REFERENCES

[1] R. Oberhauser, “VR-Git: Git Repository Visualization and Immersion in Virtual Reality,” The Seventeenth International Conference on Software Engineering Advances (ICSEA 2022), IARIA, 2022, pp. 9-14.

[2] GitHub repositories [Online]. Available from: https://web.archive.org/web/20220509204719/https://github.com/search 2023.12.01

[3] GitHub users [Online]. Available from: https://web.archive.org/web/20220529205506/https://github.com/search 2023.12.01

[4] 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/ 2023.12.01

[5] Evans Data Corporation. [Online]. Available from: https://evansdata.com/press/viewRelease.php?pressID=293 2023.12.01

[6] 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.

[7] 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.

[8] H. Bjørklund, “Visualisation of Git in Virtual Reality,” Master’s thesis, NTNU, 2017.

[9] GitHub Skyline [Online]. Available from: https://skyline.github.com 2023.12.01

[10] 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.

[11] 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.

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

[13] 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.

[14] Rogers, B., Cunningham, S. J., & Holmes, G., "Navigating the virtual library: A 3D browsing interface for information retrieval," In: Proceedings of ANZIIS'94-Australian New Zealand Intelligent Information Systems Conference, IEEE, 1994, pp. 467-471.

[15] Chiu, P., Girgensohn, A., Lertsithichai, S., Polak, W., and Shipman, F., "MediaMetro: Browsing multimedia document collections with a 3D city metaphor," In: Proceedings of the 13th annual ACM international conference on Multimedia, 2005, pp. 213-214.

[16] R. Wettel et al., "Software systems as cities: A controlled experiment," in Proc. of the 33rd International Conference on Software Engineering, ACM, 2011, pp. 551-560.

[17] F. Fittkau, A. Krause, and W. Hasselbring, “Exploring software cities in virtual reality,” Proc. IEEE 3rd Working Conference on Software Visualization (VISSOFT), IEEE Computer Society, 2015, 130-134.

[18] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza, “On the use of virtual reality in software visualization: The case of the city metaphor,” Information and Software Technology, 114, 2019, pp.92-106.

[19] 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.

[20] 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.

[21] 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

[22] 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

[23] 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, Springer, Cham, 2022, pp. 122-140. https://doi.org/10.1007/978-3-031-11510-3_8

[24] 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

[25] Libgit2Sharp. [Online]. Available from: https://github.com/libgit2/libgit2sharp 2023.12.01

[26] 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