VR-GitEvo+CI/CD: Visualizing the Evolution of Git Repositories and CI/CD Pipelines in Virtual Reality

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

Abstract – **Source code, together with its dependencies, is constantly under change pressure, and hence the codebase, typically stored and managed in a Git repository, evolves. Similarly, the associated DevOps CI/CD (Continuous Integration and Continuous Delivery) pipelines (automated processes or workflows), which automate the preparation and delivery of software artifacts, must adapt and evolve. However, visualization of the evolution of both the codebase and the associated CI/CD pipelines remains a challenge and hinders comprehension, analysis, and collaboration among DevOps stakeholders. To address this, our solution concept VR-GitEvo+CI/CD contributes an immersive visualization of codebases, dependencies, and CI/CD pipelines in Virtual Reality (VR). Our prototype realization shows its feasibility, while a case-based evaluation provides insights into its capabilities for supporting comprehension and analysis of the state and evolution of codebases and CI/CD pipelines.**

Keywords – Git; DevOps; virtual reality; visualization; software engineering; continuous integration; continuous delivery; CI/CD pipelines; code evolution; automation workflows.

I. Introduction

This paper extends our VR-DevOps paper [1], focusing on visualizing the evolution of both Git source code repositories and CI/CD (Continuous Integration and Continuous Delivery) pipelines [2] (a.k.a. DevOps pipelines); it incorporates VR visualization of Git repository evolution and extends our DevOps integration and capabilities.

"Everything moves on and nothing is at rest" is ascribed by Plato to Heraclitus [3] and reformulated by others in various ways; it essentially expresses that *change* or *dynamicity* is the only *constant* in our world. And yet many if not all of the restraints to change anchored in the physical world are absent in the digital world that software inhabits. As posited by F.P. Brooks Jr., software incurs essential difficulties or challenges that relate to its essence (inherent in its nature): complexity, conformity, changeability, and invisibility [4]. Software is essentially infinitely changeable, highly complex, exacting in conformance, and invisible. For developers, the invisibility of software obscures the underlying dependencies and complexity. This, in turn, hinders its comprehension, analysis, and management during the rapid evolution of both the codebase and the associated CI/CD pipeline, which automates the transformation of code into delivered (invokable) artifacts.

As to the degree of opaque dependencies, a 2024 industry analysis [5] of over 20,000 enterprise applications found they had 180 component dependencies on average (10% had over 400), with modern commercial software consisting of up to 90% Open-Source Software (OSS) components. As to change and evolution frequency, 6.6 trillion downloads were expected for 2024 across 7M OSS projects or components that encompass over 60M releases (on average 16 releases per OSS project annually) [5]. Already back in 2012, Google was said to have an average deployment frequency of 5,500 times a day [6][7], while Amazon was at 23,000 a day on average [6]. While we cannot extrapolate to today's rates for these IT behemoths, a 2021 survey of 1200 professionals indicated elite performers (26%) were deploying on demand multiple times a day [8]. The complex and obscure dependencies and rapid evolution of code and pipelines make comprehension challenging.

DevOps [9][10] is a methodology that combines development (Dev) and operations (Ops) with automation to improve the quality and speed of software deliveries. While there is no universally agreed to definition, key principles include Continuous Integration (CI), Continuous Delivery (CD), shared ownership, workflow automation, and rapid feedback. Both the code and tool integration and automation that DevOps addresses has become indispensable to modern software development. It has been reported that 83% of developers surveyed reported being involved in DevOps-related activities [11]. Lately, Security (Sec) has often been included in DevOps, denoted at the stage where it is primarily considered, e.g., DevSecOps [12]. Despite the popularity of DevOps, no modeling language nor visualization standard currently exists for CI/CD pipelines; each platform and vendor has their own, and it can thus be difficult for non-developers to grasp - and hence collaborate regarding - the current state of pipeline runs, and the processes involved in software development, testing, and delivery.

With the increasing demand for software functionality, large number of source code files are stored and managed in code repositories using Version Control Systems (VCS) like Git. GitHub has over 420M repositories with over 100M users [13]. A repository can become very large and continually evolve. For instance, in 2015 the Google monorepo shared by 25K developers contained 2B Lines of Code (LOC) across 9M source files having a history of 35M commits with 40K commits each workday [14], while the Linux kernel code repository contains over 40M LOC [15] across 60K files. For repositories, especially at such a scale, the dynamism of the changes as the codebase evolves can challenge comprehension and analysis.

Current visualization tools for Git repositories and CI/CD pipelines face inherent visualization limitations. While The potential of Virtual Reality (VR) to address both visual scalability and dynamic evolution has as yet been insufficiently explored. VR offers a mediated visual digital environment created and then experienced as telepresence by the perceiver. In contrast to a 2-dimensional (2D) space, VR enables an unlimited immersive space for visualizing and analyzing models and their interrelationships simultaneously in a 3D spatial structure viewable from different perspectives. In their systematic review of the DevOps field, Khan et al. [16] identified a lack of collaboration and communication among stakeholders as the primary critical challenge. Towards addressing this collaboration challenge, our contribution leverages VR towards enabling more intuitive DevOps visualization and interaction capabilities for comprehending and analyzing CI/CD pipelines, thereby supporting enhanced collaboration and communication among a larger spectrum of stakeholders. A further challenge is the finding by Giamattei et al. [17] that the landscape for DevOps tools is extremely fragmented, meaning stakeholders access various custom webpages or logs. Hence, a further goal of our solution concept is unifying the visualization and information access across heterogeneous Git and CI/CD tools.

In our prior VR work, VR-Git [18] and VR-GitCity [19] focused on Git support in VR with a vertical commit plane or city metaphor respectively, whereas VR-DevOps [1] focused on VR support for depicting pipelines. This paper contributes our solution concept VR-GitEvo+CI/CD, which provides an immersive visualization in VR of the state and evolution of both codebases, dependencies, and cross-platform CI/CD pipelines. Our prototype realization shows its feasibility, and a case-based evaluation provides insights into its potential towards supporting comprehension, analysis, and collaboration among DevOps stakeholders.

This paper is organized as follows: the next section discusses related work. Section III presents our solution concept. Section IV describes our realization, followed by an evaluation in Section V. Finally, a conclusion is drawn.

II. Related Work

Work on VR-based visualization of Git includes our own prior work VR-Git [18] and VR-GitCity [19]: VR-Git uses consecutive vertical commit planes on a hyperplane to represent commits, whereas VR-GitCity uses a city metaphor to convey code sizes across files.. Bjørklund [20], who 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 [21] provides a VR Ready 3D contribution graph as an animated skyline that can be annotated. VRGit by Zhang et al. [22] depicts the non-linear version history via a history graph anchored to the user's arm, where each node is a 3D miniature of that version highlighting changed objects.

As to non-VR based Git visualization, Git-Truck [23] hierarchical visualizations (i.e., Treemaps, Circle packing) to represent files nested in folders, with code metrics mapped to size and color; GitTruck@Duck [24] extends this further for filtering polymetric views scoped to time intervals. Gource [25] and CodeFlower [26] use an animated tree with directories as branches and files as leaves. Githru, [27] enables interactive scalable exploration of large Git commit graphs using graph reconstruction, clustering, and context-preserving squash merge. Evo-Clocks [28] represents repository evolution with each node history depicted as a separate disk clock. RepoVis [29] offers a comprehensive visual overview and search facilities using a 2D JavaScript-based web application and Ruby-based backend with a CouchDB. Githru [30] 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 [31] 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 [32] 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.

Regarding VR-based DevOps-related work, VIAProMa [33] provides a visual immersive analytics framework for project management. DevOpsUseXR is mentioned in the paper as an eXtended Reality (XR) approach for incorporating end users to allow them to directly provide feedback in Mixed Reality (MR) regarding their experience when using a specific MR app. In contrast, our solution concept is independent of the software type being built in the pipeline, and is purely virtual, remaining consistent, app-independent, and focusing on visualizing and collaborating with regard to the DevOps pipeline. The systematic review of DevOps tools by Giamattei et al. [17] does not mention any VR, XR, or MR tools.

Non-VR based DevOps work includes DevOpsML [34], a platform modeling language and conceptual framework for modeling and configuring DevOps engineering processes and platforms. DevOpsUse [35] expands DevOps to collaborate more closely with end users. The authors also state that there is in general a research gap in applying information visualization to software engineering data, and that this needs further investigation. This concurs with our view, as we were not able to find much VR or non-VR work related to DevOps visualization. Zampetti et al. [2] analyzed the pipeline evolution of 4,644 projects in 8 programming languages using Travis-CI, creating a taxonomy of 34 CI/CD pipeline restructuring actions and metric extractor of 16 indicators of how a pipeline evolves.

In contrast to the above work, VR-GitEvo+CI/CD focuses on immersively visualizing the dynamic evolution of both the codebase in Git repositories and heterogeneous CI/CD pipelines.

III. Solution Concept

Our solution approach leverages VR for visualizing the evolution of codebases and CI/CD pipelines via models that can be immersively explored and experienced in 3D.

A. Grounding in VR-Related Research

Our rationale for integrating VR into our solution concept stems from existing VR research in fields we consider relevant to modeling, analysis, and collaboration, several of which are summarized here. Akpan and Shanker [36] in their systematic meta-analysis demonstrate that VR and 3D provide marked advantages in discrete event modeling, including model development, analysis, and Verification and Validation (V&V), with common findings pointing to the positive impact of 3D/VR model analysis and V&V. Across 23 studies on 3D analysis, 95% found 3D more effective and conducive to enhanced analysis compared to 2D, notably when assessing model behavior or conducting what-if scenarios; there was broad agreement that 3D/VR effectively communicates results to decision-makers with clarity and conviction; 74% of 19 papers concluded that 3D/VR significantly enhances model development tasks, benefiting team support, and sharpening precision and clarity. In exploring VR applicability for analytical tasks within an information architecture, Narasimha et al. [37] conducted a card sorting collaboration study. Their findings indicated VR matched or exceeded in-person card sorting for certain variables, surpassing both traditional and video-based settings. Qualitative insights on awareness suggested that during collaboration, participants maintained awareness of tasks, people, and the environment, mimicking in-person interactions, with positive perceptions of VR. These findings indicate that VR facilitates a sense of presence and collaboration akin to face-to-face settings. Fonnet and Prie's [38] survey of Immersive Analytics (IA) reviewed 177 papers and found that for complex graph and spatial data, IA offers advantages over non-IA methods, though for multi-dimensional data, benefits vary with the task. They highlight that while IA supports the exploration of extensive data environments, the underutilization of context-aware navigation techniques is problematic, despite their importance to users. Müller et al. [39] compared VR vs. 2D for a software analysis task, finding no significant decrease for VR in comprehension and analysis time. While interaction time was less efficient, VR improved the user experience, was more motivating, less demanding, more inventive/innovative, and more clearly structured.

Consequently, we infer that an immersive, context-rich VR experience holds significant promise for comprehensively depicting 3D models while enhancing comprehension, awareness, analysis, and the inclusion of and collaboration with stakeholders.

B. Prior VR-Related Research

Our VR-GitEvo+CI/CD solution concept is highlighted in blue relative to our other VR solutions in Figure 1. It is based on our generalized VR Modeling Framework (VR-MF), detailed in [40], which provides a VR-based domain-independent hypermodeling framework addressing four aspects requiring special attention when modeling in VR: visualization, navigation, interaction, and data retrieval/integration.

Our VR-based solutions specific to the SE and Systems Engineering (SysE) areas include: VR-DevOps [1], which this paper extends; VR-Git [18] and VR-GitCity [19] visualize Git repositories; VR-SDLC (Software Development LifeCycle) [41] visualizes lifecycle activities and artefacts in software and systems development; VR-SBOM [42] for Software Bill of Materials and Supply Chain visualization; VR-ISA [43] for visualizing an Informed Software Architecture; VR-V&V (Verification and Validation) [36], for visualizing aspects related to quality assurance; VR-TestCoverage [45] for visualizing in VR which tests cover what test target artefacts; VR-UML [46] supports visualizing UML models; and VR-SysML [47] supports Systems Modeling Language (SysML) models.

In the Enterprise Architecture (EA) and Business Process (BP) space (EA & BP), we developed VR-EA [48] to support mapping EA models to VR, including both ArchiMate as well as BPMN via VR-BPMN [39]; VR-EAT [49] adds enterprise repository integration (Atlas and IT blueprint integration); VR-EA+TCK [50] adds knowledge and content integration; VR-EvoEA+BP [51] adds EA evolution and Business Process animation; while VR-ProcessMine [52] supports process mining in VR. Since DevOps (or DevSecOps or DevOpsUse) can be viewed as inter-disciplinary, for software organizations we view both the EA and BP areas as potentially applicable for VR-GitEvo+CI/CD to support synergies, more holistic insights, and enhanced collaboration across the enterprise and organizational space.

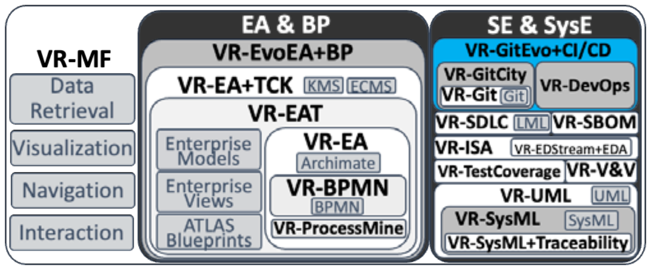


Figure 1. Conceptual map of our various published VR solution concepts with VR-GitEvo+CI/CD highlighted in blue.

C. Visualization in VR

A *pipeline* is represented as a horizontal *pipeline hyperplane*, holding vertical semi-transparent colored boxes called *run planes* (see Figure 2), which are ordered chronologically left to right. A *run plane* represents a pipeline *run*, which is colored based on status (green=success, yellow=in progress, red=error, grey= aborted). Hyperplanes also enable inter-project pipeline differentiation for larger portfolio scenarios involving multiple pipelines. The bottom of each run plane encloses a directed graph of sequential stages (cubes) of the pipeline between a start (black sphere) and an end (black sphere), while vertically stacked smaller cubes linked with lines above each stage show the internal *steps* within a stage. A cube with black borders is used to represent the entire run, and is all that is shown when a run is collapsed (e.g., to reduce visual clutter); on its front various details are depicted (ID, run duration, circular percentage of stages with status). The visualization form remains consistent across DevOps tools.

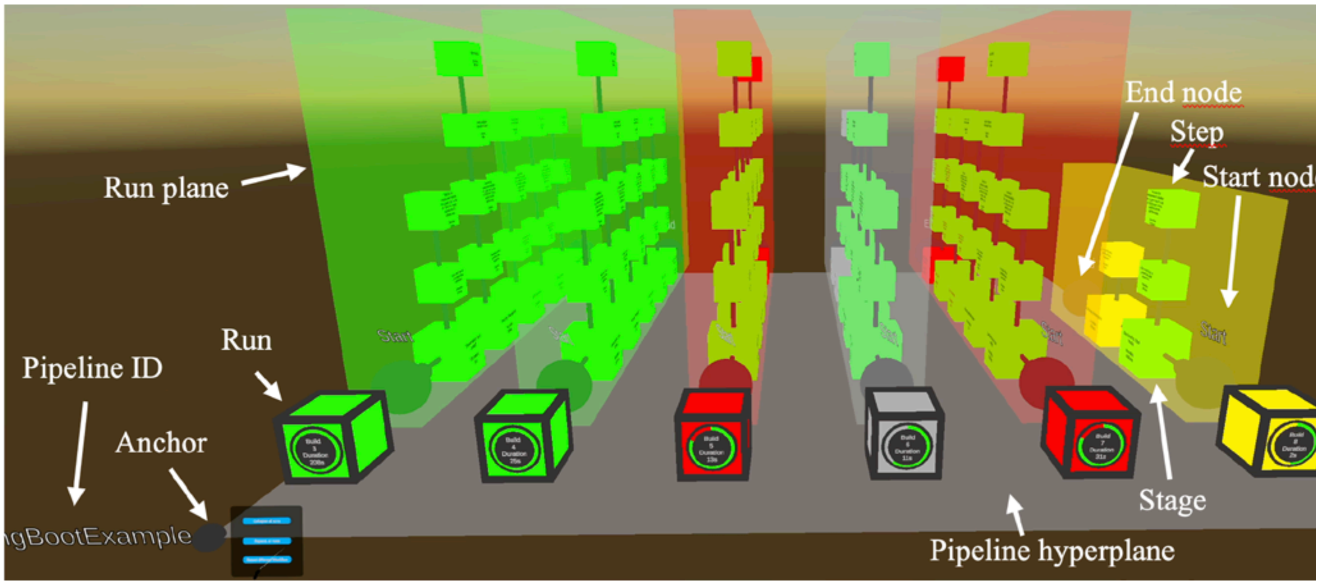


Figure 2. Hyperplane (annotated) of SprintBootExamaple Jenkins pipeline showing vertical colored run planes on a pipeline hyperplane.

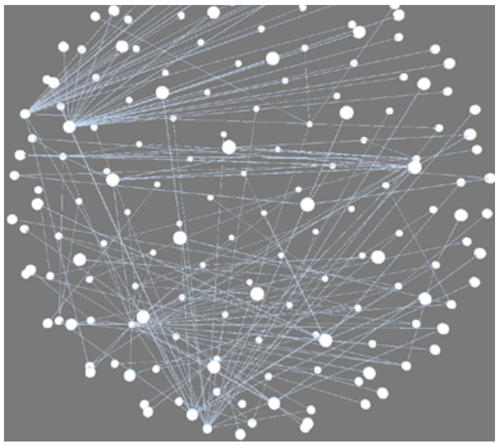


Figure 3. A nexus portraying artifact dependencies in a Git repository.

A Git code repository is depicted as a 3D graph, whereby the spherical nodes, representing artifacts (files), are placed on the surface of a 3D *nexus* sphere and edges connecting the nodes represent dependencies, as shown in Figure 3. Such a 3D nexus is spatially compact for navigating to and between nodes with the inside showing all dependencies. A 3D *radial tree* layout depicts child nodes as hierarchically-layered dependencies; here, all child nodes and their dependencies extend radially outwards from the nexus, leaving only the highest-level (parent or independent) artifacts in the nexus (reducing the number of nodes within the nexus), as shown in Figure 4. This permits dependency groupings and levels to be more easily followed. A separate Lines of Code (LOC) nexus is used to depict the relative sizes of source code files, as this graph structure represents a containing folders tree hierarchy rather than code dependencies; the node size corresponds to the LOC in that file for a selected commit. This LOC box can

files have this relevant metric.

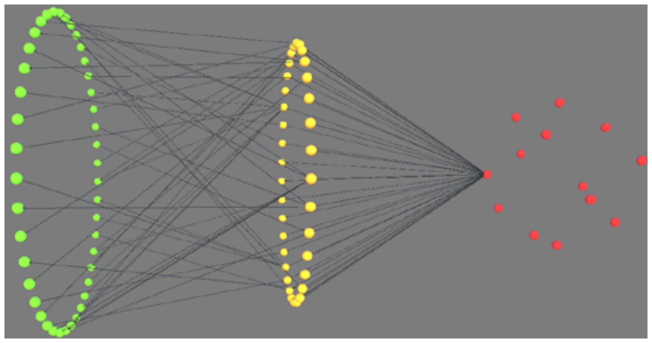


Figure 4. Child dependencies extracted as 3D radial tree from nexus.

D. *Navigation in VR*

Two navigation modes are incorporated in our solution: default gliding controls for fly-through VR, while teleporting instantly places the camera at a selected position via a selection on the VR-Tablet. Teleporting is potentially disconcerting, but may reduce the likelihood of VR sickness.

E. *Interaction in VR*

User-element interaction is supported primarily through VR controllers and a *VR-Tablet*. The VR-Tablet is used to provide detailed context-specific element information. Accordion visual elements permit more detailed information to be offered when desired. It includes a *virtual keyboard* for text entry via laser pointer key selection. On a hyperplane corner, an *anchor sphere* affordance (labeled with its pipeline ID) supports moving, hiding (collapsing), or showing (expanding) hyperplanes, as shown in the bottom left of Figure 2.

IV. Realization

The logical architecture for our prototype realization is shown in Figure 5. In our realization, the VR visualization aspects were implemented using Unity. It is supported by a Data Hub implemented in Python that integrates various Git and CI/CD data sources and stores them in JSON. All integrations with DevOps tools use their Web APIs via our tool-specific Adapters in our Data Hub for data conversion into our internal JSON format. MongoDB (locally or remotely using Atlas) is used for storage and is accessed via the MongoDB .NET/C# Driver from Unity or PyMongo from Python. To demonstrate the CI/CD tool/platform independence (i.e., heterogeneity) of our solution concept, it integrates with a local Jenkins [53] pipeline running in Docker, exemplifying a private cloud CI/CD server, as well as remote Semaphore [54] and Drone [55] CI servers to exemplify public cloud CI/CD tool integration using Web APIs.

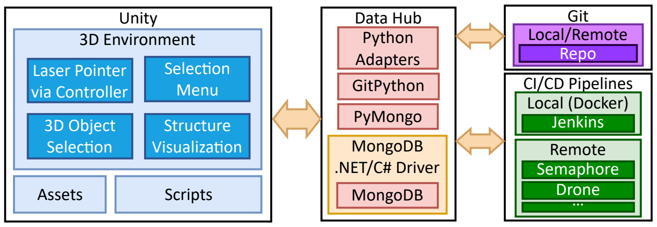
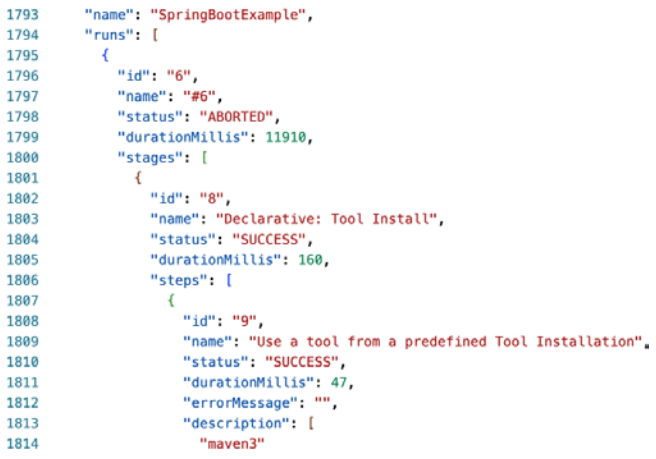


Figure 5. VR-GitEvo+CI/CD logical architecture.

The GitPython library is used to extract Git commit data: commit hash, author, timestamp, message, changed files, and changed files and metrics (insertions, deletions, lines). For Git commits, within the nexus, nodes outlined in red indicate new files while turquoise indicates changed files. Dependencies within JavaScript projects were extracted using the Node.js package manager "npm ls --all" command. For the 3D radial tree visualization, the various depth layers are colored to help differentiate them (yellow, light green, dark green, turquoise, blue, purple, pink, etc.). In the nexus layout, to highlight files affected by a commit, red indicates added and turquoise updated. In the LOC boundary box, black nodes are directories and files are red.

A CD pipeline is an automated expression of the process for preparing software for delivery. A Jenkins pipeline is a set of Jenkins plugins with a set of instructions specified in a text-based Jenkins file using Groovy syntax. It can be written in a scripted or declarative syntax, and typically defines the entire build process, including building, testing, and delivery. Concepts involved can include agents (executors), nodes (machines), stages (subset of tasks), and steps (a single task). Both Semaphore and Drone pipelines are described via a YAML syntax. We created our own common generic JSON format to store pipeline information, see Figure 6. A pipeline instance refers to a run. The refresh rates can be configured for Data Hub state retrieval from Unity and for each Adapter's Web APIs calls.



```
    "name": "SpringBootExample",
    "runs": [
      {
        "id": "6",
        "name": "#6",
        "status": "ABORTED",
        "durationMillis": 11910,
        "stages": [
          {
            "id": "8",
            "name": "Declarative: Tool Install",
            "status": "SUCCESS",
            "durationMillis": 160,
            "steps": [
              {
                "id": "9",
                "name": "Use a tool from a predefined Tool Installation",
                "status": "SUCCESS",
                "durationMillis": 47,
                "errorMessage": "",
                "description": [
                  "maven3"
```

Figure 6. Snippet of VR-GitEvo+CI/CD common run representation in JSON.

V. Evaluation

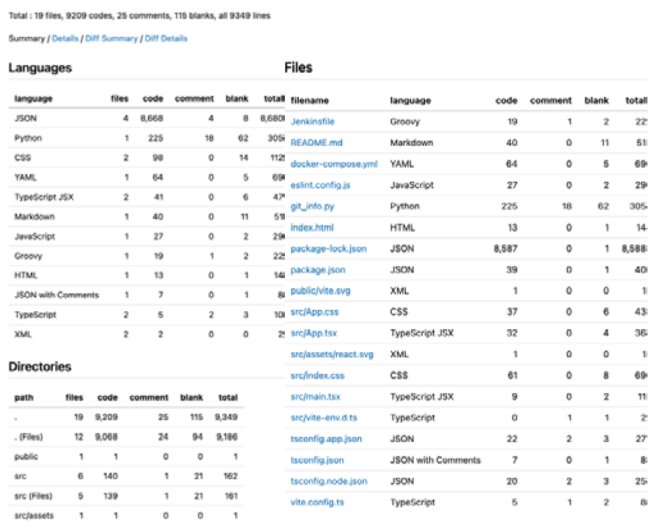
For the evaluation of our solution concept, we refer to design science method and principles [56], in particular a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). A scenario-based case study is used in evaluating the Git and CI/CD pipeline aspects separately to address these particular stakeholder concerns (for various stakeholders, not just developers):

1) *Status scenario:* focuses primarily on conveying status information, so that stakeholders can readily determine the current state,
2) *Analysis scenario:* focuses primarily on supporting analysis and investigation tasks via the provisioning of information towards understanding essential features, relations, constituent elements, issue identification, issue resolution, etc., and
3) *Evolution scenario:* focuses primarily on supporting comprehension of the evolution of the underlying structure (Git repository or CI/CD pipeline) via the provisioning of time-based change information to support comprehension regarding structural differences.

A. Git Code Repositories

1) Evaluation Setup

A very simple Vite-based HelloWorld Node.js app was used for the evaluation, the metrics shown in Figure 7. The app includes 477 Node modules that consist of 18K files and 1.8M LOC. Since one is often oblivious to all included modules, perhaps explicitly including some, which in turn include multiple others, we utilize Node.js to demonstrate the VR-GitEvo+CI/CD concept, in particular the 3D hierarchical radial tree, since such modules exhibit a much deeper dependency hierarchy then what is explicitly specified via inclusion. If modules or their dependencies are not of interest, then these can be hidden.



Total : 19 files, 9209 codes, 25 comments, 115 blanks, all 9349 lines

Summary / Details / Diff Summary / Diff Details

Languages

language	files	code	comment	blank	total
JSON	4	8,668	4	8	8,680
Python	1	225	18	62	305
CSS	2	98	0	14	112
YAML	1	64	0	5	69
TypeScript JSX	2	41	0	6	47
Markdown	1	40	0	11	51
JavaScript	1	27	0	2	29
Groovy	1	19	1	2	22
HTML	1	13	0	1	14
JSON with Comments	1	7	0	1	8
TypeScript	2	5	2	3	10
XML	2	2	0	0	2

Directories

path	files	code	comment	blank	total
.	19	9,209	25	115	9,349
. (Files)	12	9,068	24	94	9,186
public	1	1	0	0	1
src	6	140	1	21	162
src (Files)	5	139	1	21	161
src/assets	1	1	0	0	1

Files

filename	language	code	comment	blank	total
Jenkinsfile	Groovy	19	1	2	22
README.md	Markdown	40	0	11	51
docker-compose.yml	YAML	64	0	5	69
eslint.config.js	JavaScript	27	0	2	29
git_info.py	Python	225	18	62	305
index.html	HTML	13	0	1	14
package-lock.json	JSON	8,587	0	1	8,588
package.json	JSON	39	0	1	40
public/vite.svg	XML	1	0	0	1
src/App.css	CSS	37	0	6	43
src/App.tsx	TypeScript JSX	32	0	4	36
src/assets/react.svg	XML	1	0	0	1
src/index.css	CSS	61	0	8	69
src/main.tsx	TypeScript JSX	9	0	2	11
src/vite-env.d.ts	TypeScript	0	1	1	2
tsconfig.app.json	JSON	22	2	3	27
tsconfig.json	JSON with Comments	7	0	1	8
tsconfig.node.json	JSON	20	2	3	25
vite.config.ts	TypeScript	5	1	2	8

Figure 7. Git vite project language, directory, and file metrics.

2) Git Status Scenario

To determine the state of the Git repository, the LOC nexus on the right shows the files that are included in the project, the edges between them showing the relation between the containing directory and that file, with node size indicating relative LOC size, as shown on the right in Figure 8. In the boundary box on the left, the dependencies are shown between files and modules are shown, making apparent the many (often hidden) dependencies between included modules.

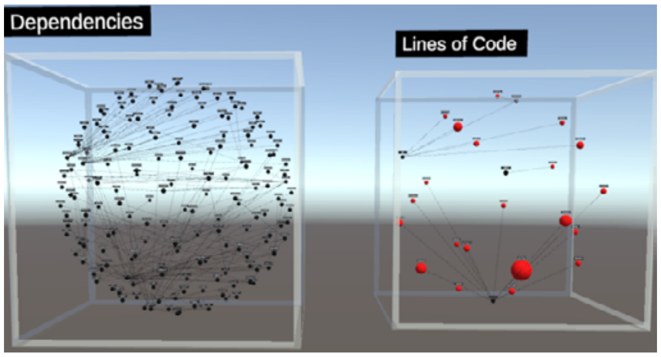
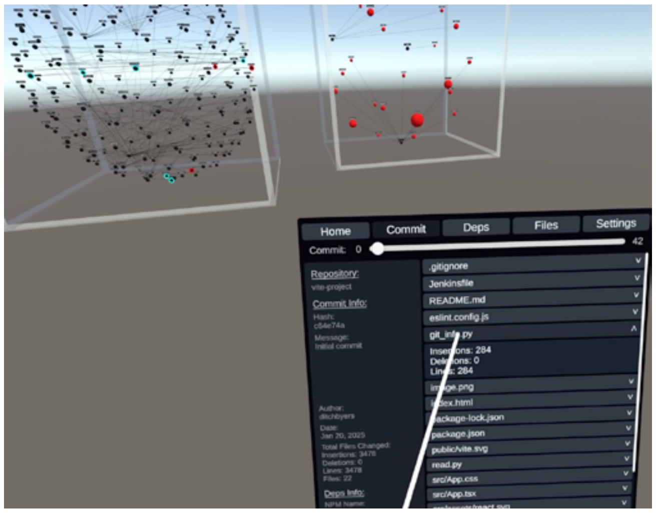


Figure 8. Git Dependency nexus (left) and LOC nexus (right).

To view details about the state of a specific commit, the Commit Tab on the VR-Tablet is shown in Figure 9. It provides commit status information as to the repository, specific commit info, author, date, total files and lines changed, and a scrollable list of the files making up the commit, which with an accordion can be expanded to show additional metrics of lines inserted, lines deleted, and total LOC size. The Dependencies Nexus on the top left shows highlighted nodes, red for files added and turquoise for files updated by this commit.

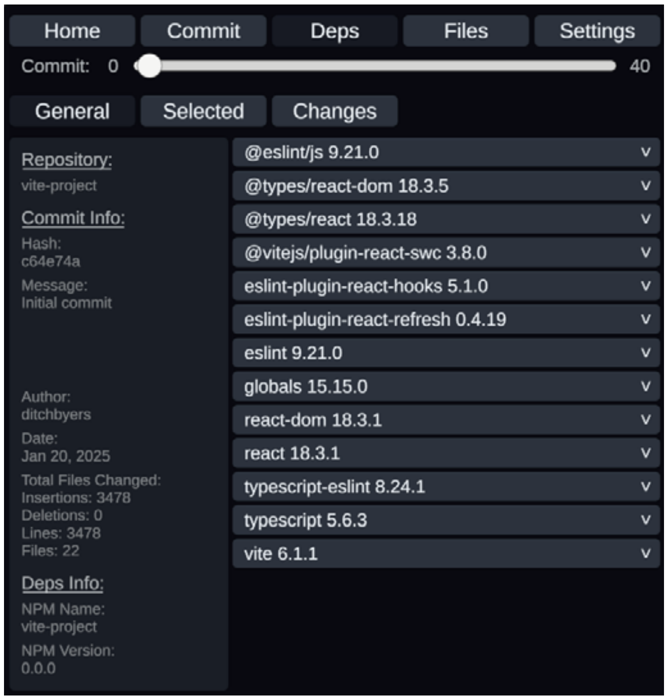


Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:

.gitignore
Jenkinsfile
README.md
eslint.config.js
git_info.py
Insertions: 284
Deletions: 0
Lines: 284
image.png
index.html
package-lock.json
package.json
public/vite.svg
read.py
src/App.css
src/App.tsx

Figure 9. VR-Tablet: Git Commit Tab: Details.

The Dependencies Tab on the VR-Tablet is shown in Figure 10. It lists the explicit (first degree) included modules, and, using the accordion, one can drill down and, in turn, determine the included modules of each of these.

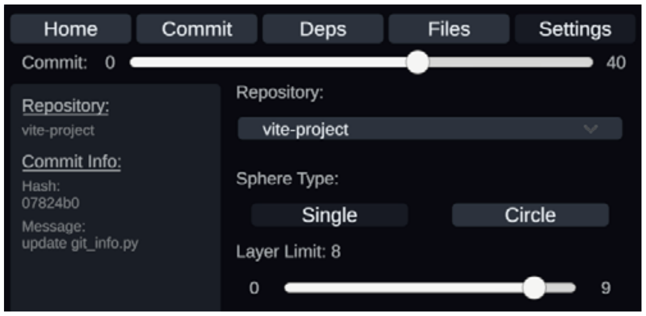


Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

@eslint/js 9.21.0
@types/react-dom 18.3.5
@types/react 18.3.18
@vitejs/plugin-react-swc 3.8.0
eslint-plugin-react-hooks 5.1.0
eslint-plugin-react-refresh 0.4.19
eslint 9.21.0
globals 15.15.0
react-dom 18.3.1
react 18.3.1
typescript-eslint 8.24.1
typescript 5.6.3
vite 6.1.1

Figure 10. VR-Tablet: Git Dependencies Tab: General status.

The Settings Tab on the VR-Tablet is shown in Figure 11. With it, the repository, the boundary box layout, and the depth limit can be specified. For a different repository (vite-project2), the sliders max layer limit and number of commits available are adjusted accordingly, as shown in Figure 12.



Commit: 0 — 40

Repository:
vite-project

Commit Info:
Hash:
07824b0
Message:
update git_info.py

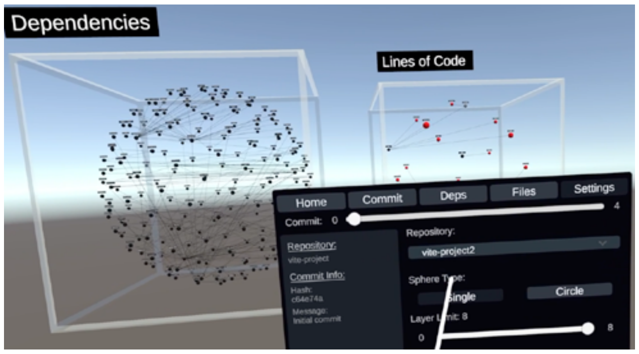
Repository: vite-project

Sphere Type:

Layer Limit: 8

0 — 9

Figure 11. VR-Tablet: Git Settings Tab: selected repo, layout, and layer limit option status.



Commit: 0 — 4

Repository:
vite-project

Commit Info:
Hash:
c64e74a
Message:
Initial commit

Repository: vite-project2

Sphere Type:

Layer Limit: 8

Figure 12. VR-Tablet: Git Settings Tab: selecting another repo adjusts commits available.

3) *Git Analysis Scenario*

To support analysis of Git repositories, multiple options are offered. By default, the Dependency Nexus shows all dependencies to make one aware of the entire set of dependencies, as shown in Figure 13.

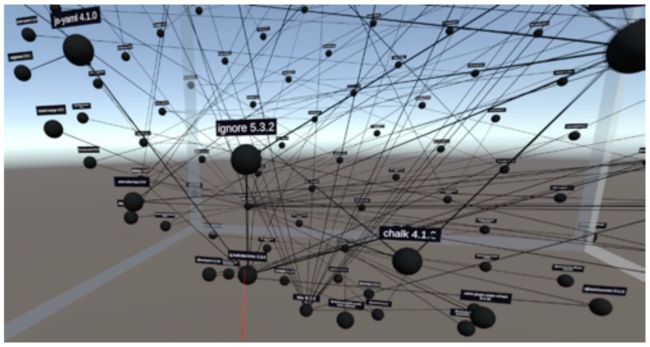
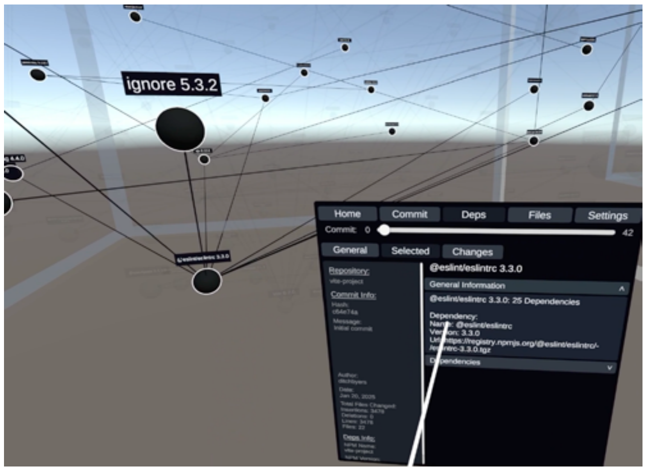


Figure 13. Git: Nexus: closeup of dependencies before node selection.

When a single node is selected, non-related dependencies and nodes are ghosted (to minimize visual clutter) and help focus on its specific dependencies, as shown in Figure 14. Moreover, the Deps Tab switches to Selected and offers specific information on that node and its dependencies. A view from further out shows the highlighted selected nodes and the ghosted can still be faintly seen, as depicted in Figure 15.



@eslint/eslintrc 3.3.0
General Information
@eslint/eslintrc 3.3.0: 25 Dependencies
Dependency:
Name: @eslint/eslintrc
Version: 3.3.0
Url: https://registry.npmjs.org/@eslint/eslintrc/-/eslintrc-3.3.0.tgz
Dependencies

Figure 14. Git: Nexus: selecting a node leaves tree of dependencies and toggles ghosting of rest; VR-Tablet offers detailed information.

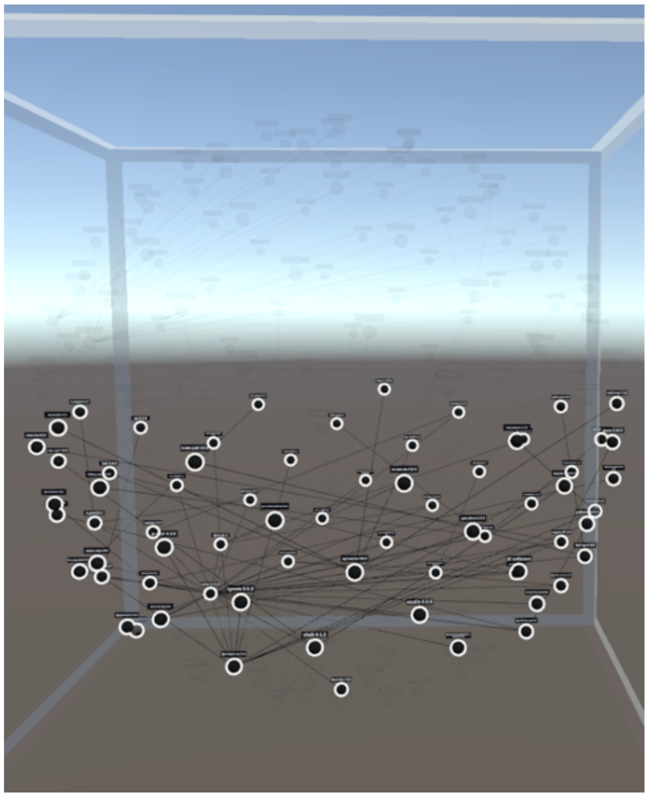


Figure 15. Git: Nexus: selected dependencies and rest of nexus ghosted.

Contingent on the depth of the dependencies, it may be beneficial to view the dependencies hierarchically and limiting the layers, as shown with our 3D radial tree layout (Circle in VR-Tablet) with the slider set to 2 layers of depth, as shown in Figure 16.

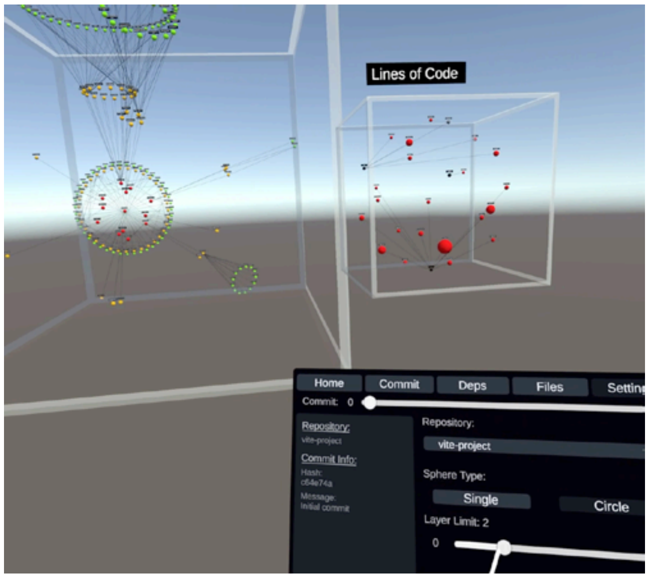


Figure 16. Git: 3D Radial Tree (Circle) layout with depth limit 2.

In contrast, with the setting to its maximum depth (reaches 9 for this repo at a certain commit date) is shown in Figure 17. These dependencies can be hierarchically navigated, with certain nodes having n-m incoming and outgoing dependencies.

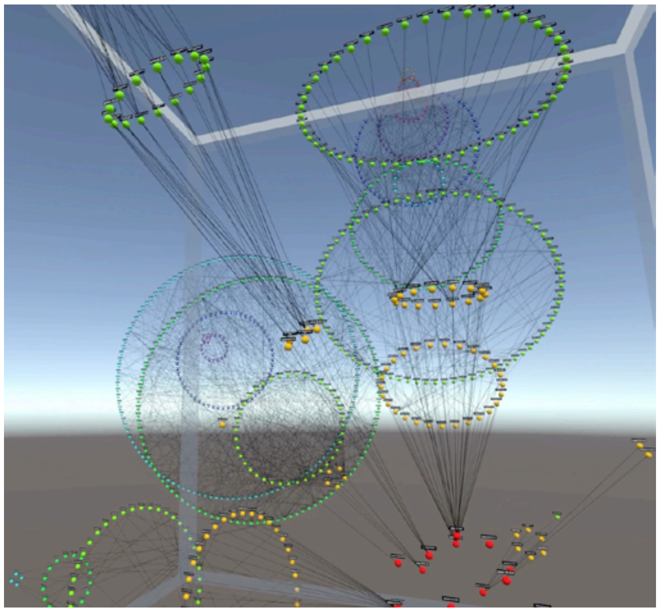


Figure 17. Git: 3D Radial (Circle) with depth limit 9.

Furthermore, to assist with analysis, a subset of dependencies can be highlighted (ghosting other nodes and dependencies) by selecting a specific node of interest, as exemplified with the jest node in Figure 18.

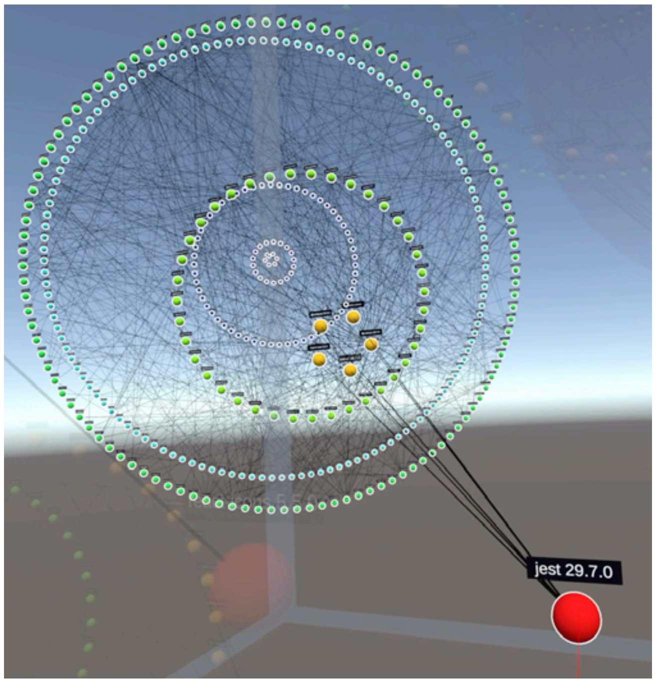
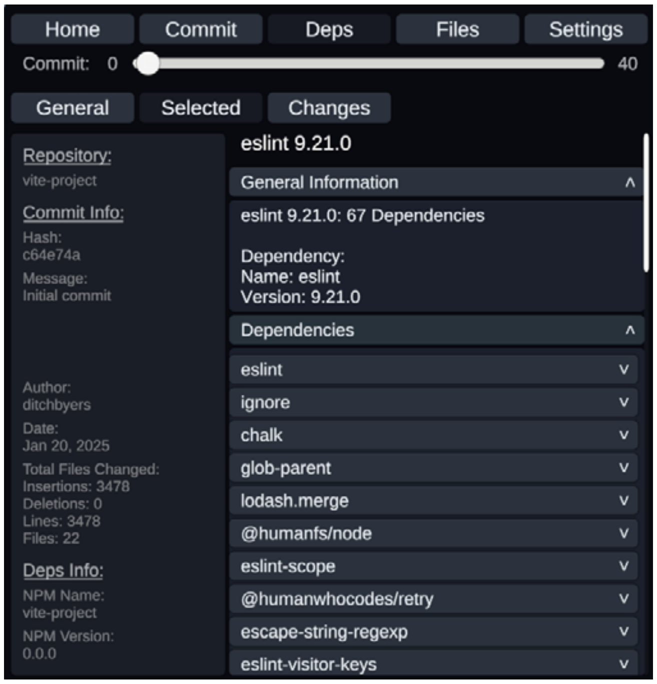


Figure 18. Git: Radial Layout: selecting node shows dependencies and toggles ghosting of rest.

A subset of dependencies can be highlighted (ghosting other nodes and dependencies) by selecting a specific node of interest, as exemplified with the jest node in Figure 18. This can also be done by selecting an element of interest via the VR-Tablet. The full set of dependencies can also be viewed and navigated in the VR-Tablet, as shown Figure 19.



Repository:
vite-project

Commit Info:
Hash:
c64e74a
Message:
Initial commit

Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22

Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

eslint 9.21.0

General Information

eslint 9.21.0: 67 Dependencies

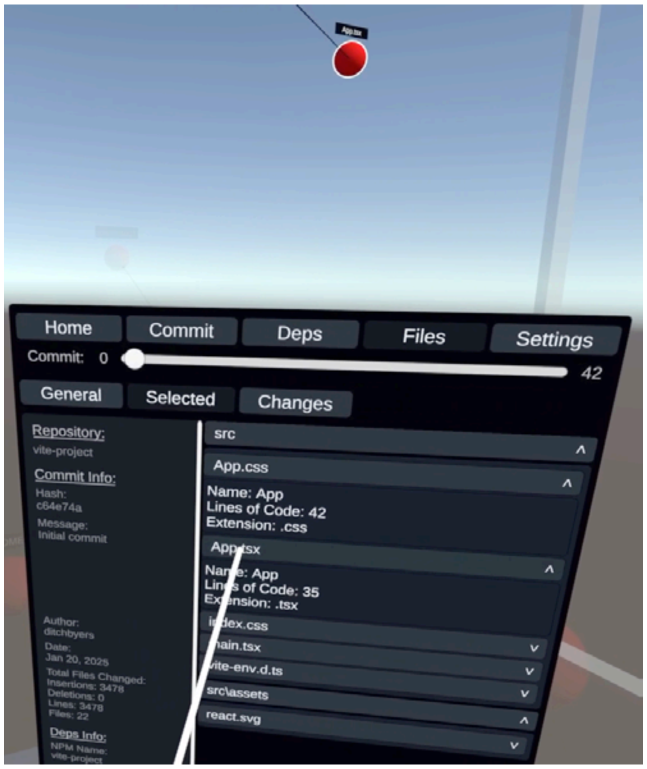
Dependency:
Name: eslint
Version: 9.21.0

Dependencies

eslint
ignore
chalk
glob-parent
lodash.merge
@humanfs/node
eslint-scope
@humanwhocodes/retry
escape-string-regexp
eslint-visitor-keys

Figure 19. VR-Tablet: Git Dependencies Tab: Selected Dependencies.

If one wishes to analyze the size of the files (or any other metric of interest if it were implemented), one can select a node of interest in the nexus in the LOC boundary box and the corresponding data is then shown in the VR-Tablet, as seen in Figure 20. The relation of a file (red node) to its containing directory (black nodes) is shown via edges within the nexus, while the relative size is conveyed via the sphere size, as shown in Figure 21.



Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:
vite-project

src
App.css
Name: App
Lines of Code: 42
Extension: .css
App.tsx
Name: App
Lines of Code: 35
Extension: .tsx
dex.css
ain.tsx
ite-env.d.ts
src\assets
react.svg

Figure 20. VR-Tablet: Git Files Tab: Files contained in selected src folder.

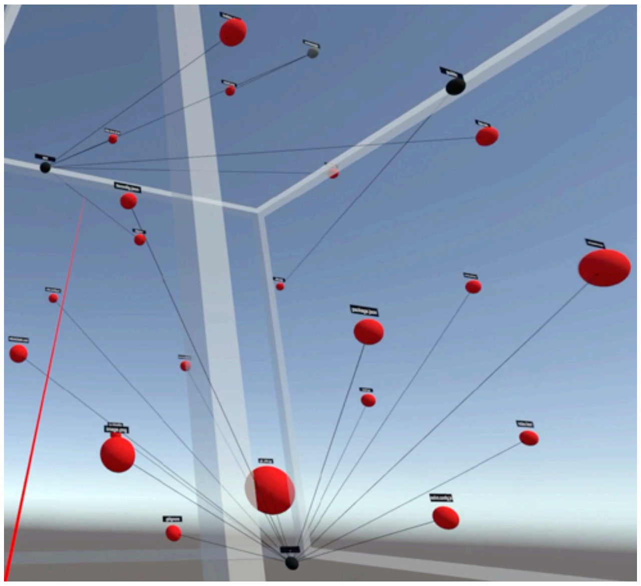
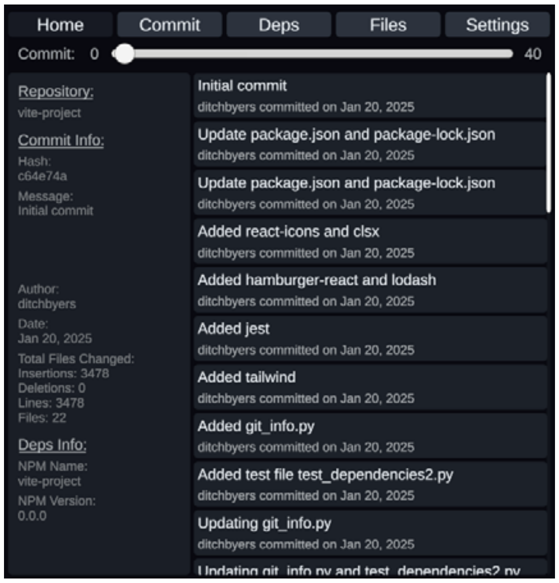


Figure 21. Git: Lines of Code nexus.

4) Git Evolution Scenario

For the evolution scenario, changes and time are of interest to the stakeholders. For this, the VR-Tablet offers on the Home Tab a list of all commits with the commit messages, author, date, and a slider showing the total number of commits, as shown in Figure 22. By moving this slider, the contents in the boundary boxes are redrawn to show the state at that evolution point and the changes to the repo by that commit, as shown in Figure 23. Hence, by sliding the slider, the history of changes of these nexuses are redrawn and thus animated and changes are dynamically apparent. Similarly, the changes to dependencies can be viewed as a 3D radial tree and time-adjusted, as shown in Figure 24.



Repository:
vite-project
Commit Info:
Hash:
c64e74a
Message:
Initial commit
Author:
ditchbyers
Date:
Jan 20, 2025
Total Files Changed:
Insertions: 3478
Deletions: 0
Lines: 3478
Files: 22
Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

Initial commit
ditchbyers committed on Jan 20, 2025
Update package.json and package-lock.json
ditchbyers committed on Jan 20, 2025
Update package.json and package-lock.json
ditchbyers committed on Jan 20, 2025
Added react-icons and clsx
ditchbyers committed on Jan 20, 2025
Added hamburger-react and lodash
ditchbyers committed on Jan 20, 2025
Added jest
ditchbyers committed on Jan 20, 2025
Added tailwind
ditchbyers committed on Jan 20, 2025
Added git_info.py
ditchbyers committed on Jan 20, 2025
Added test file test_dependencies2.py
ditchbyers committed on Jan 20, 2025
Updating git_info.py
ditchbyers committed on Jan 20, 2025

Figure 22. VR-Tablet: Home Tab: Git commit messages

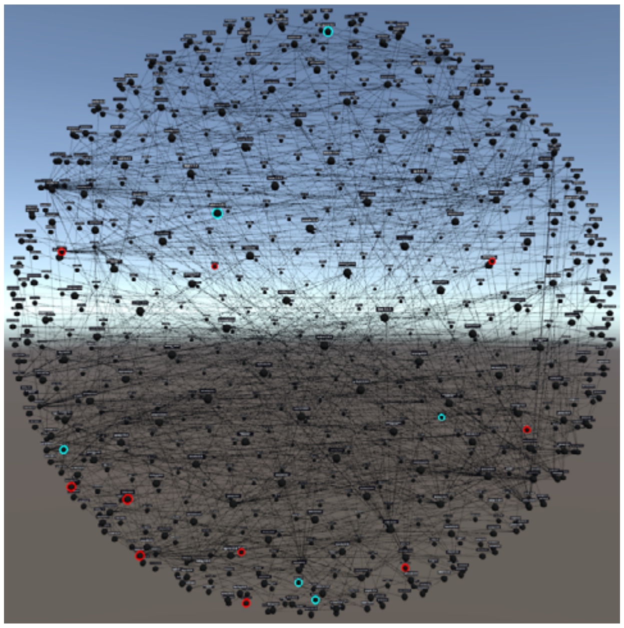


Figure 23. Git: Nexus: new (red) and updated (turquoise) nodes in commit.

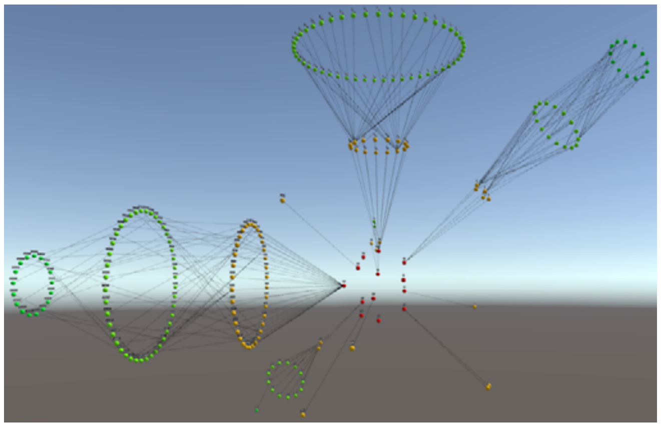
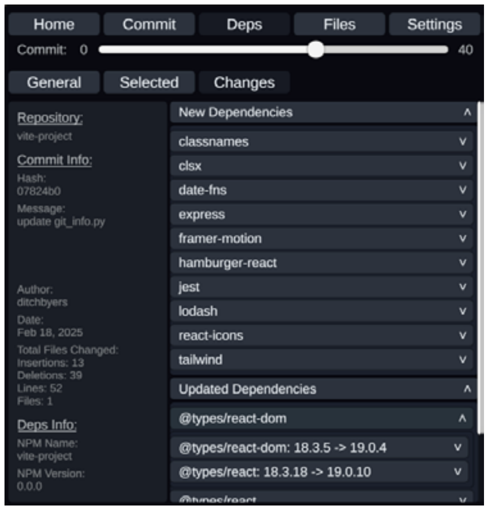


Figure 24. Git: 3D radial: dependencies as 3D radial tidy trees.

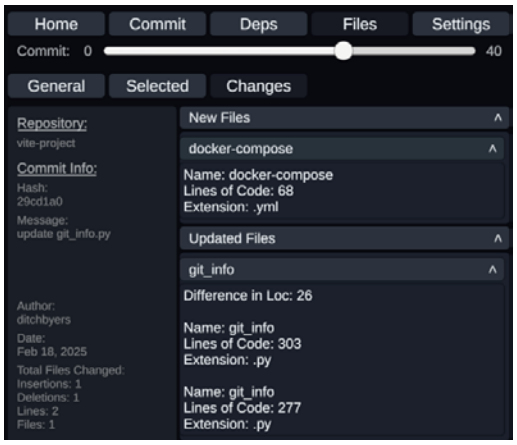
The detailed changes to dependencies can also be viewed in the VR-Tablet in the Deps Tab under Changes providing further details, as shown in Figure 25. If more interested in File changes, then in the VR-Tablet in the Files Tab under Changes further details are provided there, including changes to metrics, as shown in Figure 26.



Repository:
vite-project
Commit Info:
Hash:
07824b0
Message:
update git_info.py
Author:
ditchbyers
Date:
Feb 18, 2025
Total Files Changed:
Insertions: 13
Deletions: 39
Lines: 52
Files: 1
Deps Info:
NPM Name:
vite-project
NPM Version:
0.0.0

New Dependencies
classnames
clsx
date-fns
express
framer-motion
hamburger-react
jest
lodash
react-icons
tailwind
Updated Dependencies
@types/react-dom
@types/react-dom: 18.3.5 -> 19.0.4
@types/react: 18.3.18 -> 19.0.10
@types/react

Figure 25. VR-Tablet: Git Dependencies Tab: Changes.



Repository:
vite-project
Commit Info:
Hash:
29cd1a0
Message:
update git_info.py
Author:
ditchbyers
Date:
Feb 18, 2025
Total Files Changed:
Insertions: 1
Deletions: 1
Lines: 2
Files: 1

New Files
docker-compose
Name: docker-compose
Lines of Code: 68
Extension: .yml
Updated Files
git_info
Difference in Loc: 26

Name: git_info
Lines of Code: 303
Extension: .py

Name: git_info
Lines of Code: 277
Extension: .py

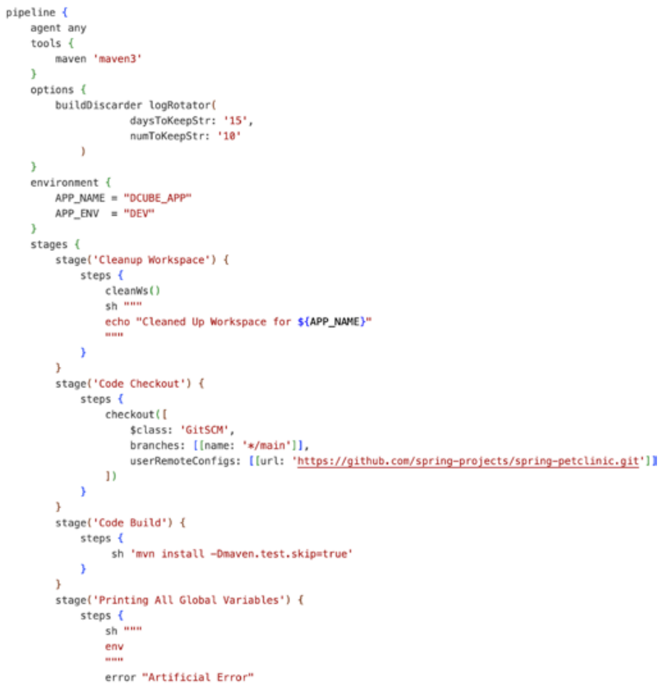
Figure 26. VR-Tablet: Git Files Tab: Changes.

B. *CI/CD Pipelines*

To demonstrate the heterogeneity of the solution, various screenshots of runs from Jenkins, Semaphore, or Drone are used interchangeably.

1) Evaluation Setup

For Jenkins, the SpringBoot PetClinic example pipeline [57] includes 39 Java files and 1335 Lines of Code (LOC). For pipeline error illustration purposes, an additional step with an artificial error was inserted into the SpringBoot example in a second version of the CI/CD pipeline, as shown in the listing in Figure 27. For Semaphore, the Android App example pipeline includes 13 Kotlin files and 287 LOC, as shown in the listing in Figure 28.



```
pipeline {
    agent any
    tools {
        maven 'maven3'
    }
    options {
        buildDiscarder logRotator(
                    daysToKeepStr: '15',
                    numToKeepStr: '10'
            )
    }
    environment {
        APP_NAME = "DCUBE_APP"
        APP_ENV  = "DEV"
    }
    stages {
        stage('Cleanup Workspace') {
            steps {
                cleanWs()
                sh """
                echo "Cleaned Up Workspace for ${APP_NAME}"
                """
            }
        }
        stage('Code Checkout') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/main']],
                    userRemoteConfigs: [[url: 'https://github.com/spring-projects/spring-petclinic.git']]
                ])
            }
        }
        stage('Code Build') {
            steps {
                 sh 'mvn install -Dmaven.test.skip=true'
            }
        }
        stage('Printing All Global Variables') {
            steps {
                sh """
                env
                """
                error "Artificial Error"
```

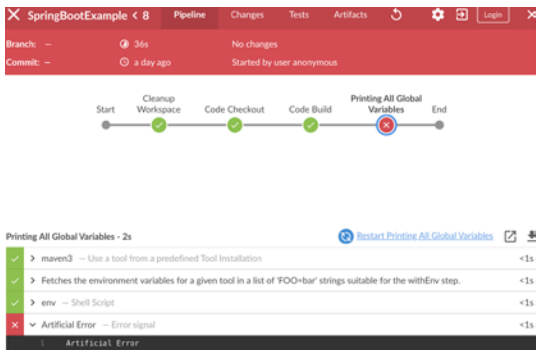
Figure 27. SpringBoot PetClinic Jenkins pipeline in Groovy (snippet).

2) CI/CD Pipeline Status Scenario

In 2D, dashboards are typically available for assessing a selected CI/CD pipeline instance state, yet each tool has its own unique interface, as exemplified for the Jenkins tool in Figure 29. The equivalent for Semaphore is shown in Figure 30. In our VR-GitEvo+CI/CD, a unified interface for heterogeneous CI/CD pipelines is provided, such that a stakeholder can readily comprehend and assess the current status and state of multiple pipeline runs. Any particular run may execute different steps and stages (e.g., due to an abort, error, option, etc.). Fully collapsed run planes provide a high-level overview, with black-lined cubes representing any pipeline instance and conveying details, as shown in Figure 34 and Figure 35.

```yaml
version: v1.0
name: Verification Pipeline
agent:
  machine:
    type: e1-standard-2
    os_image: ubuntu2004
  containers:
    - name: main
      image: 'registry.semaphoreci.com/android:29'
global_job_config:
  env_vars:
    - name: ADB_INSTALL_TIMEOUT
      value: '10'
  prologue:
    commands:
      - checkout
      - cache restore gradle-wrapper
      - cache restore gradle-cache
      - cache restore android-build
blocks:
  - name: Build
    task:
      jobs:
        - name: Build Project
          commands:
            - ./gradlew bundleDebug
      epilogue:
        on_pass:
          commands:
            - cache clear
            - cache store gradle-wrapper ~/.gradle/wrapper
            - cache store gradle-cache ~/.gradle/caches
            - cache store android-build ~/.android/build-cache
  - name: Verification
    task:
      jobs:
        - name: Analyze Code
          commands:
            - ./gradlew lint
        - name: Unit Tests
          commands:
            - ./gradlew testDebugUnitTest
      epilogue:
        always:
          commands:
            - artifact push job --expire-in 2w --destination reports/ app/build/reports/
promotions:
  - name: Start Integration tests
    pipeline_file: integration-tests.yml
```

Figure 28. Semaphore Android App pipeline in YAML format


SpringBootExample < 8 | Pipeline | Changes | Tests | Artifacts | Login

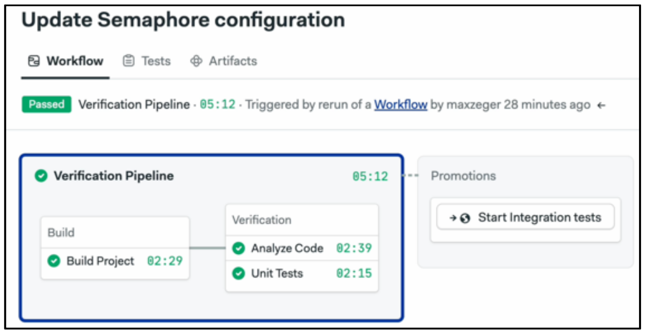
Branch: – | 36s | No changes
Commit: – | a day ago | Started by user anonymous

Start — Cleanup Workspace — Code Checkout — Code Build — Printing All Global Variables — End

Printing All Global Variables - 2s   Restart Printing All Global Variables

```
✓ > maven3 – Use a tool from a predefined Tool Installation                                                    <1s
✓ > Fetches the environment variables for a given tool in a list of 'FOO=bar' strings suitable for the withEnv step.   <1s
✓ > env – Shell Script                                                                                          <1s
× ˅ Artificial Error – Error signal                                                                            <1s
      Artificial Error
```

Figure 29. Jenkins tool web screenshot.


Update Semaphore configuration

Workflow | Tests | Artifacts

Passed Verification Pipeline · 05:12 · Triggered by rerun of a Workflow by maxzeger 28 minutes ago

Verification Pipeline 05:12
- Build
  - Build Project 02:29
- Verification
  - Analyze Code 02:39
  - Unit Tests 02:15
- Promotions
  - Start Integration tests

Figure 30. Semaphore tool web screenshot.

3) *CI/CD Pipeline Analysis Scenario*

VR-GitEvo+CI/CD supports analysis of issues via immersive visual patterns and contrasts, visually revealing differences in the detailed steps executions, as shown for the Android App in Figure 31. Here, each *run plane* represents a pipeline *run*, which is colored based on status (green=success, yellow=in progress, red=error, grey= aborted). The contrasts with a YAML (YAML Ain't Markup Language) pipeline definition, which contains many details that are difficult for certain stakeholders to grasp, while lacking status info, as in Figure 28. Alternatively, a web-based graphical status may be offered, yet lack comprehensive details for analysis, as seen in Figure 29 and Figure 30.

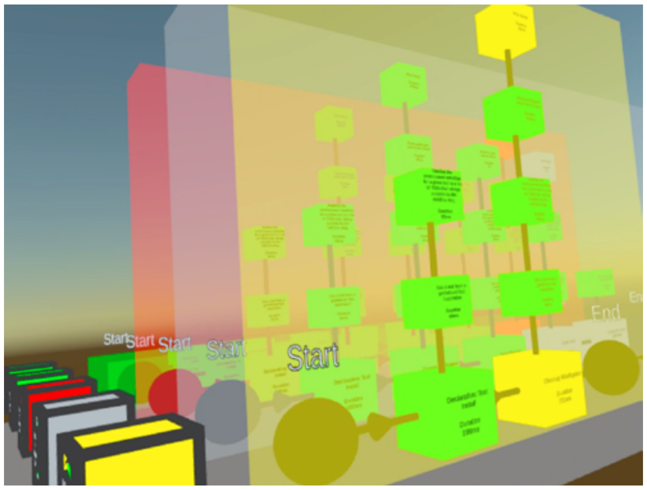
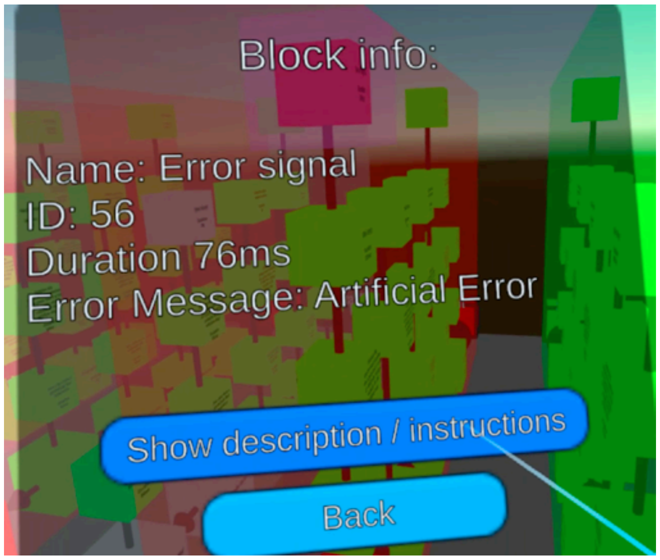


Figure 31. Immersive analysis via visual colored run comparison of stage/step status (for Semaphore pipeline).

Furthermore, to assist with analysis tasks, the VR-Tablet supports information retrieval, including additional context-specific *metadata* and *error messages* about a particular block as seen Figure 32.


Block info:

Name: Error signal
ID: 56
Duration 76ms
Error Message: Artificial Error

Figure 32. VR-Tablet shows contextual element details: metadata and instructions/description.

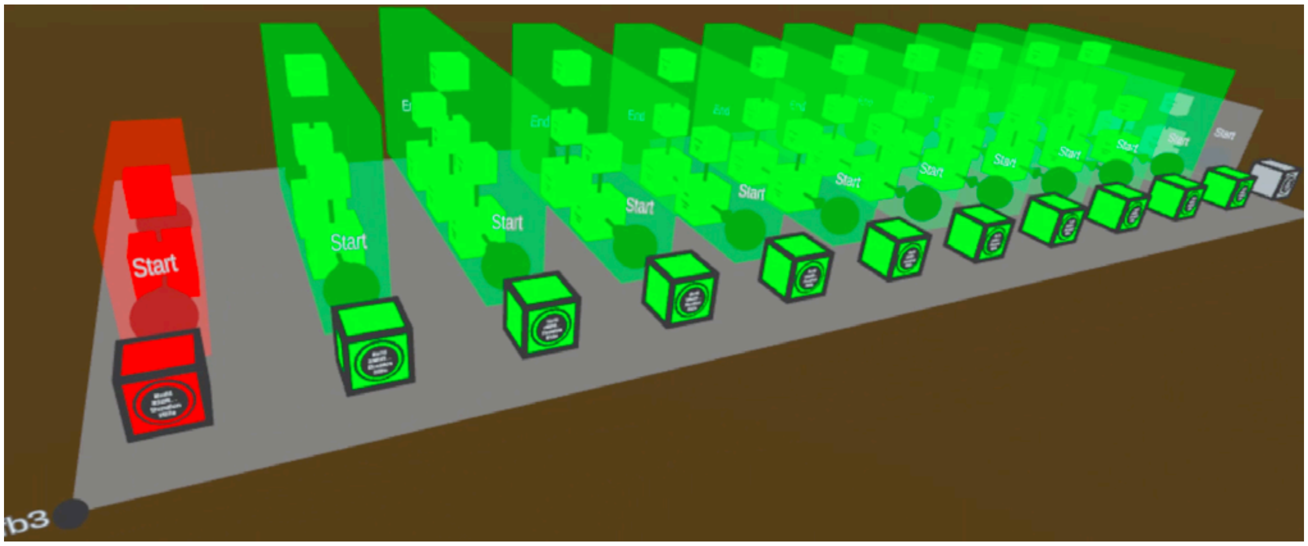


Figure 33. Pipeline run status for a set of Semaphore pipeline runs showing expanded step details and an aborted process in grey on the far right.

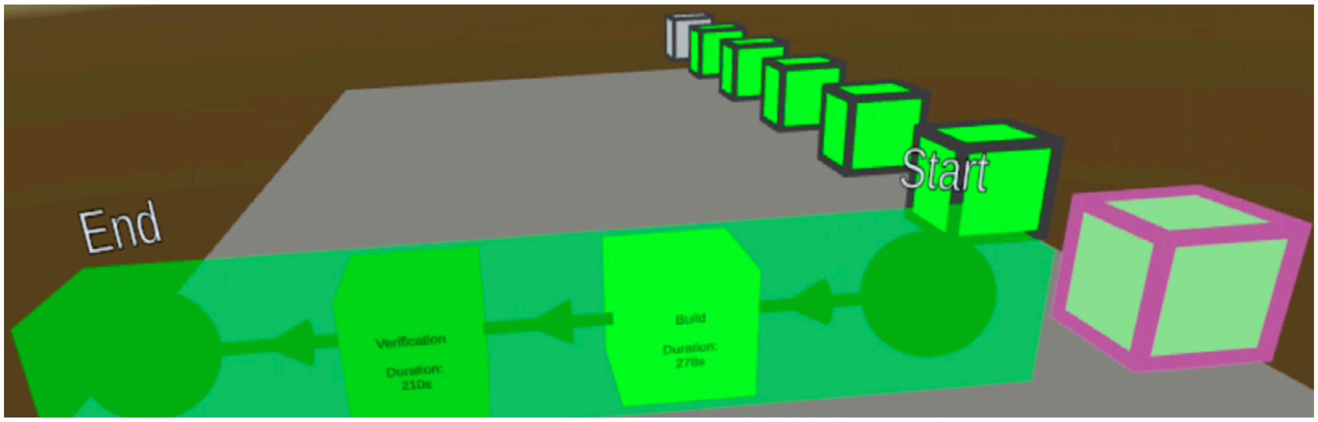
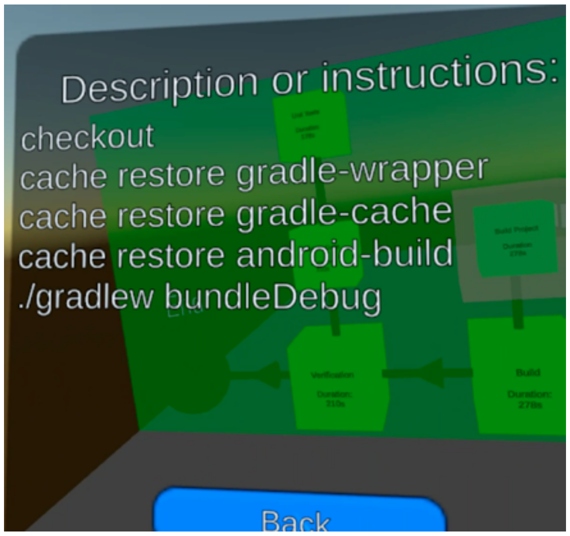


Figure 34. Collapsed Semaphore runs on a pipeline hyperplane with stages expanded (and steps collapsed) for a selected run.

The pipeline *stage* or *step task* instructions can be viewed, as shown in Figure 35.

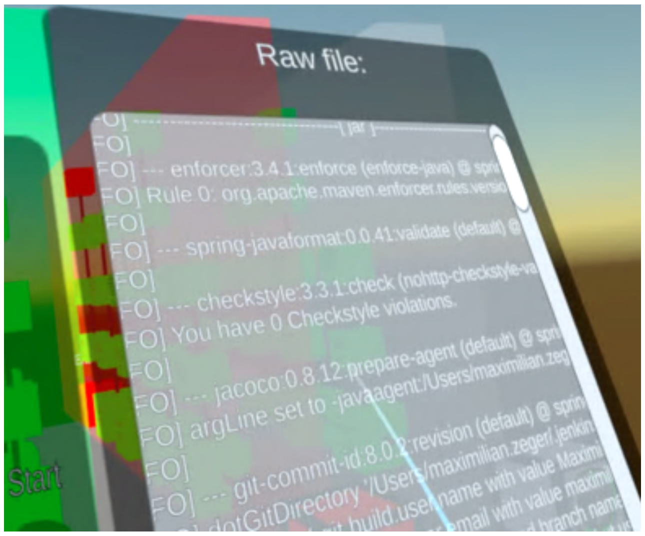


Description or instructions:

checkout
cache restore gradle-wrapper
cache restore gradle-cache
cache restore android-build
./gradlew bundleDebug

Figure 35. VR-Tablet shows contextual element details: instructions or description.

Raw pipeline *log* information is available, as shown in Figure 36.

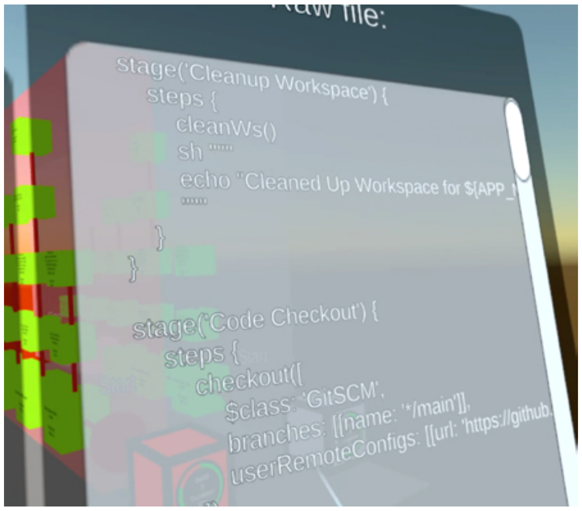


Raw file:

```
O] ----------------------------[ jar ]----------
FO]
FO] --- enforcer:3.4.1:enforce (enforce-java) @ spri
FO] Rule 0: org.apache.maven.enforcer.rules.versio
FO]
FO] --- spring-javaformat:0.0.41:validate (default) @
FO]
FO] --- checkstyle:3.3.1:check (nohttp-checkstyle-va
FO] You have 0 Checkstyle violations.
FO]
FO] --- jacoco:0.8.12:prepare-agent (default) @ spri
FO] argLine set to -javaagent:/Users/maximilian.zeg
FO]
FO] --- git-commit-id:8.0.2:revision (default) @ sprin
  ] dotGitDirectory '/Users/maximilian.zeger/.jenkin
     git.build.user.name with value Maximi
     email with value maximi
     branch name
```

Figure 36. VR-Tablet offers raw file access to log.

The raw *pipeline code specification* can also be viewed directly in the VR-Tablet, as seen in Figure 37.



```
stage('Cleanup Workspace') {
    steps {
        cleanWs()
        sh """
        echo "Cleaned Up Workspace for ${APP_
        """
    }
}

stage('Code Checkout') {
    steps {
        checkout([
            $class: 'GitSCM',
            branches: [[name: '*/main']],
            userRemoteConfigs: [[url: 'https://github.
```

Figure 37. VR-Tablet offers raw file access to pipeline definition.

This consolidation of information in the VR-Tablet, in conjunction with visual context in VR, could improve the utility and efficacy of analysis tasks, especially when considering increasing pipeline complexity, pipeline versions, and large scale-out of runs.

4) CI/CD Pipeline Evolution Scenario

To support pipeline evolution scenarios, changes and time are factors for stakeholder. To view versioning changes to stages or steps, an immersive visual differentiation of runs can be performed by choosing a perspective and alignment, as shown for the PetClinic pipeline in Figure 38. Comprehending the pipeline structure and any delta via its Groovy file would be more difficult for non-developer stakeholders. Visual depiction and differentiation can help support the inclusion of non-tech-savvy stakeholders, improving the speed of assessments and the quality of analysis tasks via the inclusion of diverse stakeholders.

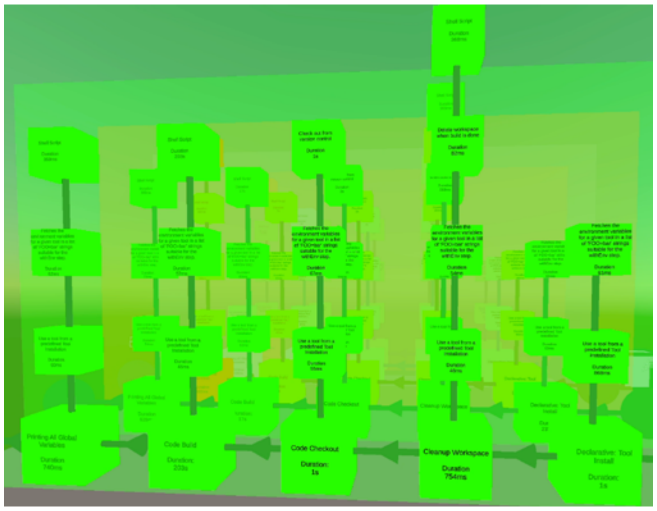


Figure 38. Immersive analysis of pipeline evolution via visual alignment of stage/steps (for Jenkins pipeline).

Moreover, any issues with pipelines over time can be readily viewed via the status of all instances. The status details of each run can be individually collapsed or expanded as desired for the analysis, shown in Figure 34 and Figure 35. Furthermore, the timeline slider on the VR-Tablet can be used to bring any specific pipeline instance to the forefront. This is shown for a specific failure case in Figure 40. The equivalent for a successful case is shown in Figure 41.

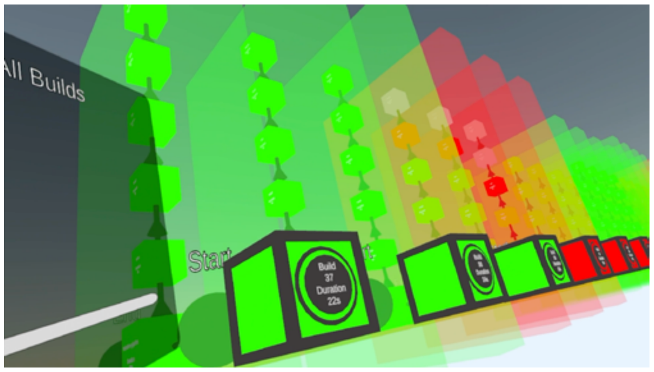


Figure 39. Timeline slider can be used to view history of all pipeline instances shown.

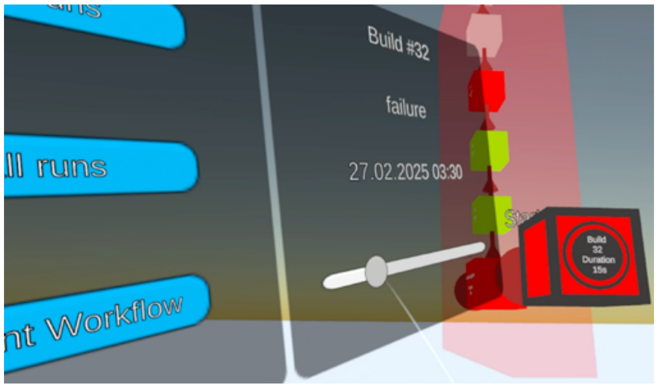


Figure 40. Timeline slider can be used to select a single execution.

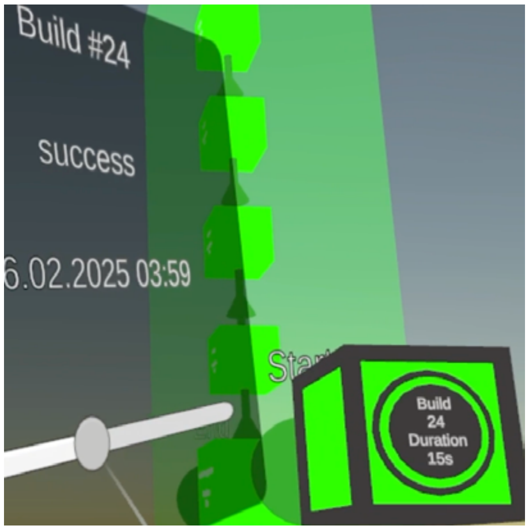


Figure 41. Timeline slider can be used to select a single.

C. Discussion

The scenario-based case study using our prototype realization showed the ability of the GitEvo+CI/CD solution concept to address DevOps stakeholders concerns regarding both Git codebase repositories and CI/CD pipelines. These scenarios included a status scenario, analysis scenario, and evolution scenario. The use of VR supports a collaborative, immersive experience without visual limitations, enabling it to scale across large projects and multiple projects concurrently with a relatively intuitive and simple homogeneous interface to diverse local and remote Git repository providers and CI/CD providers.

VI. Conclusion

Visualizing the evolution of both Git codebases and CI/CD pipelines remains a challenge and hinders comprehension, analysis, and collaboration among DevOps stakeholders. VR-GitEvo+CI/CD offers an immersive visualization solution concept for codebases, their dependencies, and CI/CD pipelines in VR. The realization prototype showed its feasibility, while the case-based evaluation showed its potential to support comprehension in typical scenarios such as status, analysis, and evolution. The solution concept is DevOps tool-independent, hiding the differences that the fragmented DevOps tool landscape might present to non-tech-savvy stakeholders. It thus provides a way towards broader inclusion of various DevOps stakeholders, and can thus support greater collaboration and communication to address a significant challenge facing DevOps.

For future work, we see the potential for more holistic DevOps insights via a deeper integration with our other existing VR solutions. Additional future work includes: support for GitOps, Infrastructure as Code, VR-native developer support, collaboration and annotation capabilities, and a comprehensive industrial empirical study.

Acknowledgment

The authors would like to thank Pascal Rene May, Maximilian Stricker, and Maximilian Zeger for his assistance with the design, implementation, and evaluation.

References

[1] R. Oberhauser, “VR-DevOps: Visualizing and Interacting with DevOps Pipelines in Virtual Reality,” In: Proc. 19th International Conference on Software Engineering Advances, pp. 43-48, 2024.

[2] Zampetti, F., Geremia, S., Bavota, G., and Di Penta, M., “CI/CD pipelines evolution and restructuring: A qualitative and quantitative study,” In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME) (pp. 471-482). IEEE, 2021.

[3] J. M. Robinson, “An introduction to early Greek philosophy: The chief fragments and ancient testimony, with connecting commentary,” Advanced Reasoning Forum, p. 90, Fragment 5.14, 2021.

[4] F. P. Brooks, Jr., *The Mythical Man-Month*, Boston, MA: Addison-Wesley Longman Publ. Co., Inc., 1995.

[5] Sonatype, “State of the Software Supply Chain,” 2024, https://sonatype.com/hubfs/SSCR-2024/SSCR_2024-FINAL-optimized.pdf 2025.05.10

[6] IT Revolution, “DevOps Guide: Selected Resources to Start Your Journey,” The IT Revolution, 2015. [Online]. Available from: https://web.archive.org/web/20211010072856/http://images.itrevolution.com/documents/ITRev_DevOps_Guide_5_2015.pdf 2025.05.10

[7] J. Micco, “Tools for continuous integration at google scale,” Google Tech Talk, Google Inc., 2012.

[8] DevOps Research and Assessment (DORA) Team, “Accelerate State of DevOps report,” 2021. [Online]. Available from: https://services.google.com/fh/files/misc/state-of-devops-2021.pdf 2025.05.10

[9] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "DevOps," in IEEE Software, vol. 33, no. 3, pp. 94-100, May-June 2016.

[10] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect’s Perspective*, Addison-Wesley Professional, 2015.

[11] L. Dodd and B. Noll, “State of CI/CD Report 2024: The Evolution of Software Delivery Performance,” SlashData and the Continuous Delivery Foundation, 2024.

[12] GitLab, “A Maturing DevSecOps Landscape,” 2021. [Online]. Available from: https://about.gitlab.com/images/developer-survey/gitlab-devsecops-2021-survey-results.pdf 2025.05.10

[13] J. D'Souza, “GitHub Statistics By Developers, Git Pushes and Facts” [Online]. Available from: https://web.archive.org/web/20250226132029/https://www.coolest-gadgets.com/github-statistics/ 2025.05.10

[14] R. Potvin and J. Levenberg, “Why Google stores billions of lines of code in a single repository,” In: Communications of the ACM, 59(7), pp.78-87, 2016. https://doi.org/10.1145/2854146

[15] M. Tyson, “Linux kernel source expands beyond 40 million lines – it has doubled in size in a decade,” Tom's Hardware, January 26, 2025. [Online]. Available from: https://www.tomshardware.com/software/linux/linux-kernel-source-expands-beyond-40-million-lines-it-has-doubled-in-size-in-a-decade 2025.05.10

[16] M. S. Khan, A. W. Khan, F. Khan, M. A. Khan, and T. K. Whangbo, "Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review," in IEEE Access, vol. 10, pp. 14339-14349, 2022.

[17] L. Giamattei et al., “Monitoring tools for DevOps and microservices: A systematic grey literature review,” Journal of Systems and Software, vol. 208, 2024, p.111906.

[18] R. Oberhauser, "VR-Git: Git Repository Visualization and Immersion in Virtual Reality," 17th International Conference on Software Engineering Advances (ICSEA 2022), IARIA, 2022, pp. 9-14.

[19] R. Oberhauser, “VR-GitCity: Immersively Visualizing Git Repository Evolution Using a City Metaphor in Virtual Reality,” International Journal on Advances in Software, 16 (3 & 4), 2023, pp. 141-150.

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

[21] GitHub Skyline [Online]. Available from: https://skyline.github.com 2025.05.10

[22] L. Zhang, A. Agrawal, S. Oney, and A. Guo, “VRGit: A Version Control System for Collaborative Content Creation in Virtual Reality,” In: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23). Association for Computing Machinery, New York, NY, USA, Article 36, pp. 1–14, 2023. https://doi.org/10.1145/3544548.3581136

[23] K. Højelse, T. Kilbak, J. Røssum, E. Jäpelt, L. Merino, and M. Lungu, "Git-Truck: Hierarchy-Oriented Visualization of Git Repository Evolution," 2022 Working Conference on Software Visualization (VISSOFT), Limassol, Cyprus, 2022, pp. 131-140, doi: 10.1109/VISSOFT55257.2022.00021.

[24] A. Hoff, T. H. Kilbak, L. Merino, and M. Lungu, "GitTruck@Duck - Interactive Time Range Selection in

Hierarchy-Oriented Polymetric Visualization of Git Repository Evolution," 2024 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2024, pp. 853-857, doi: 10.1109/ICSME58944.2024.00090.

[25] https://gource.io, last accessed 2025.05.10

[26] CodeFlower [Online]. Available from: https://github.com/fzaninotto/CodeFlower 2025.05.10

[27] Y. Kim et al., "Githru: Visual Analytics for Understanding Software Development History Through Git Metadata Analysis," IEEE Transactions on Visualization and Computer Graphics, vol. 27, 2021.

[28] C. V. Alexandru, S. Proksch, P. Behnamghader, and H. C. Gall, "Evo-Clocks: Software Evolution at a Glance," in 2019 Working Conference on Software Visualization (VISSOFT). IEEE, 2019.

[29] 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.

[30] 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.

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

[32] 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.

[33] B. Hensen and R. Klamma, "VIAProMa: An Agile Project Management Framework for Mixed Reality," In: Augmented Reality, Virtual Reality, and Computer Graphics (AVR 2021), LNCS, vol 12980, Springer, Cham, 2021, pp. 254-272.

[34] A. Colantoni, L. Berardinelli, and M. Wimmer, "DevopsML: Towards modeling devops processes and platforms," In: 23rd ACM/IEEE International Conference Model Driven Engineering Languages and Systems: Companion Proc., ACM, 2020, pp. 1-10.

[35] I. Koren, "DevOpsUse: A Community-Oriented Methodology for Societal Software Engineering," In: Ernst Denert Award for Software Engineering 2020, Springer, 2022, pp. 143-165.

[36] I. J. Akpan and M. Shanker, "The confirmed realities and myths about the benefits and costs of 3D visualization and virtual reality in discrete event modeling and simulation: A descriptive meta-analysis of evidence from research and practice," Computers & Industrial Engineering, vol. 112, pp. 197-211, 2017.

[37] S. Narasimha, E. Dixon, J. W. Bertrand, and K. C. Madathil, "An empirical study to investigate the efficacy of collaborative immersive virtual reality systems for designing information architecture of software systems," Applied Ergonomics, vol. 80, pp. 175-186, 2019.

[38] A. Fonnet and Y. Prie, "Survey of immersive analytics," IEEE Transactions on Visualization and Computer Graphics, vol. 27, no. 3, pp. 2101-2122, 2019.

[39] 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.

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

[41] R. Oberhauser, "VR-SDLC: A Context-Enhanced Life Cycle Visualization of Software-or-Systems Development in Virtual Reality," In: Business Modeling and Software Design (BMSD 2024), LNBIP, vol 523, Springer, Cham, 2024, pp. 112-129, https://doi.org/10.1007/978-3-031-64073-5_8.

[42] R. Oberhauser, "VR-SBOM: Visualization of Software Bill of Materials and Software Supply Chains in Virtual Reality," In: Business Modeling and Software Design (BMSD 2025), LNBIP, Springer, Cham, 2025.

[43] R. Oberhauser, "VR-ISA: Immersively Visualizing Informed Software Architectures Using Viewpoints Based on Virtual Reality," In: International Journal on Advances in Software, Vol. 17, No. 3 & 4, pp. 282-300, IARIA, ISSN: 1942-2628.

[44] R. Oberhauser, "VR-V&V: Immersive Verification and Validation Support for Traceability Exemplified with ReqIF, ArchiMate, and Test Coverage," International Journal on Advances in Systems and Measurements, 16 (3 & 4), 2023, pp. 103-115.

[45] R. Oberhauser, "VR-TestCoverage: Test Coverage Visualization and Immersion in Virtual Reality," In: Proc. Fourteenth International Conference on Advances in System Testing and Validation Lifecycle (VALID 2022), IARIA, 2022, pp. 1-6.

[46] R. Oberhauser, "VR-UML: The unified modeling language in virtual reality – an immersive modeling experience," International Symposium on Business Modeling and Software Design (BMSD 2021), Springer, Cham, 2021, pp. 40-58, doi.org/10.1007/978-3-030-79976-2_3

[47] 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.

[48] R. Oberhauser and C. Pogolski, "VR-EA: Virtual Reality Visualization of Enterprise Architecture Models with ArchiMate and BPMN," In: Business Modeling and Software Design (BMSD 2019), LNBIP, vol. 356, Springer, Cham, 2019, pp. 170-187, https://doi.org/10.1007/978-3-030-24854-3_11.

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

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

[51] R. Oberhauser, M. Baehre, and P. Sousa, "VR-EvoEA+BP: Using Virtual Reality to Visualize Enterprise Context Dynamics Related to Enterprise Evolution and Business Processes," In: Business Modeling and Software Design (BMSD 2023), LNBIP, vol 483, Springer, 2023, pp. 110-128, https://doi.org/10.1007/978-3-031-36757-1_7.

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

[53] https://www.jenkins.io, last accessed 2025.05.10

[54] https://semaphore.io, last accessed 2025.05.10

[55] https://www.drone.io, last accessed 2025.05.10

[56] 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.

[57] B. Wilson. *Jenkins Pipeline Tutorial For Beginners*. [Online]. Available from: https://devopscube.com/jenkins-pipeline-as-code/ 2025.05.10

[58] GitHub. *Semaphore demo CI/CD pipeline for Android*. [Online]. Available from: https://github.com/Semaphore-demos/semaphore-demo-android/ 2025.05.