VR-V&V: Immersive Verification and Validation Support for Traceability Exemplified with ReqIF, ArchiMate, and Test Coverage

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

Abstract – **To build quality into a software (SW) system necessitates supporting quality-related lifecycle activities during the software development. In software engineering, software Verification and Validation (V&V) processes constitute an inherent part of Software Quality Assurance (SQA) processes. A subset of the V&V activities involved are: 1) bidirectional traceability analysis of requirements to design model elements, and 2) software testing. Yet the complex nature of large SW systems and the dependencies involved in both design models and testing present a challenge to current V&V tools and methods regarding support for trace analysis. One of software's essential challenges remains its invisibility, which also affects V&V activities. This paper contributes VR-V&V, a Virtual Reality (VR) solution concept towards supporting immersive V&V activities. By visualizing requirements, models, and testing artifacts with dependencies and trace relations immersively, they are intuitively accessible to a larger stakeholder audience such as SQA personnel while supporting digital cognition. Our prototype realization shows the feasibility of supporting immersive bidirectional traceability as well as immersive software test coverage and analysis. The evaluation results are based on a case study demonstrating its capabilities, in particular traceability support was performed with ReqIF, ArchiMate models, test results, test coverage, and test source to test target dependencies.**

Keywords – Virtual reality; visualization; software verification and validation; software requirements traceability; software test traceability; software test coverage; code coverage; software testing; ReqIF; ArchiMate.

I. INTRODUCTION

This paper extends out Virtual Reality (VR)-based immersive test coverage capability presented in VR-TestCoverage [1], extending its scope to support software (SW) Verification and Validation (V&V) activities in VR, in particular support for immersive trace analysis of dependencies between requirements, design models such as ArchiMate, test results, test coverage, and test source to test target dependencies.

The IEEE 730-2014 Standard for Software Quality Assurance Processes [2] includes evaluation tasks that specifically include the terms *verify* and *validate*, otherwise known as V&V. During the development lifecycle, software validation is the technical process that evaluates and provides evidence about software satisfying requirements, intended usage, and user needs [3]. During the software development lifecycle, software verification is the technical process of evaluating the software or component and associated artifacts for objective evidence that activities performed during each development process satisfy the criteria for that lifecycle activity [3]. The stated intention of V&V is to support an organization in building quality into the software during its development life cycle [4]. V&V does so by ensuring that requirements meet certain quality criteria (e.g., complete, correct, consistent, accurate). Conformance with an activity's requirements and the product's requirements is determined by assessing, analyzing, reviewing, inspecting, and *testing* products and processes [4]. Depending on the required integrity level, SW testing varies in the types, degree, and scope performed to support V&V at various levels, e.g., construction verification via unit testing, integration testing, or system testing. Furthermore, in Annex E of [2] for Industry-specific guidance for applying IEEE 730-2014, the definition of software verification for the medical device industry includes: "Software testing is one of many verification activities intended to confirm that software development output meets its input requirements." Indeed, without executing SW dynamically via SW testing, one would be hard pressed to confirm its requirements are satisfied. Traceability analysis involving bidirectional tracing between elements is a common task specified in many V&V activities. tracing between (product/system or process) requirements, design, construction, test, and other elements to check of correctness, completeness, and consistency [4]. Thus, tracing (dependent on traceability) and testing (either reviewing thereof or performing) are inherent tasks accompanying V&V.

Despite the apparent importance of traceability, the software industry lacks explicit support for bidirectional traceability across software artefacts, e.g., via international specifications, formats, automation, or non-proprietary popular tools. This situation often results in traceability being a manual effort documented utilizing spreadsheets or text documents as exemplified in [5] and [6]. Confounding the traceability issues for V&V are the inherent properties and essential difficulties of software according to Brooks [7]: its complexity, conformity, changeability, and invisibility. Brooks stated that the invisibility of software is an essential difficulty of software construction because the reality of software is not embedded in space. With regard to V&V and traceability support for larger SW systems, comprehension challenges emerge for stakeholders due to two main aspects: as the quantity and granularity of elements and related artifacts increase, the inter- and intra-dependencies that traceability considers exacerbate the *complexity*. Furthermore, the *invisibility* of these "implicit" relations in current tooling diminishes comprehension due to a lack of visualization capability that can extend across artifacts, model, and heterogeneous tool elements.

As a powerful visualization capability, Virtual Reality (VR) could potentially address aspects of both: 1) invisibility, due to its digital nature and ability to portray artificial constructs, and 2) complexity, due to its unlimited immersive space. VR thus provides an unlimited immersive space for visualizing and analyzing 3D spatial structures viewable from different perspectives. Müller et al. [8] 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. Via its unique visualization and immersive capability, VR can support V&V trace visualization and analysis while providing a motivational benefit.

As to our prior work, with regard to modeling in VR, VR-UML [9] and VR-SysML [10] provide VR-based visualization of Unified Modeling Language (UML) [11] and System Modeling Language (SysML) [12] diagrams respectively, with VR-EA [13] supporting immersive ArchiMate [14] EA models. VR-SysML+Traceability [15] investigated SysML-centric traceability support in VR via automated extraction of manually placed requirement ID annotations in code and test files referencing requirements modeled in SysML and depicting test pass rates; yet it did not address ReqIF [16] sources, ArchiMate, nor automated test coverage nor test dependency aspects.

Extending the immersive test coverage and tracing capability of VR-TestCoverage [1], this paper contributes the solution concept VR-V&V towards supporting immersive V&V activities. It visualizes requirements extracted from ReqIF together with design models and testing artifacts, showing dependencies and trace relations immersively to address invisibility and complexity issues. Thus, comprehension for V&V tracing can be improved while being intuitively accessible to a larger stakeholder audience (such as SQA personnel). Our prototype realization shows its feasibility. The case-based evaluation provides insights into its capabilities, in particular traceability support with ReqIF, ArchiMate models, test results, test coverage, and test source to test target dependencies

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

II. Related Work

In work related to requirements traceability visualization, Li & Maalej [17] found traceability matrices and graphs preferrable for management tasks. Graphs were preferred for navigating linked artifacts, while matrices were preferred for an overview. Users were not always capable of choosing the most suitable visualization. Abad et al. [18] performed a systematic literature review on requirements engineering visualization. Madaki & Zainon [19] performed a review on tools and techniques for visualizing SW requirement traceability. In none of the above literature were immersive or VR techniques mentioned, nor was our own literature search able to find similar work. Some software tool vendors provide proprietary product solutions to support some aspects of traceability, e.g., IBM Engineering Requirements Management DOORS Next [20], Perforce Helix ALM [21], Sparx Systems Enterprise Architect [22], etc. Yet these typically do not address heterogenous design models, cross-diagram dependencies, integration with ReqIF requirements, and test coverage and test target dependencies. In any case, they do not support the display of such trace dependencies in 3D or VR.

Furthermore, our literature search found no other VR work directly addressing test coverage (or code coverage). VR-related work regarding software analysis includes VR City [23], which applies a 3D city metaphor. While it briefly mentions that its work might be used for test coverage, it shows no actual results in this regard and in this regard only a trace mode visualization is depicted.

Non-VR work on structural testing or code coverage includes Dreef et al. [24], which applies a global overview test-matrix visualization. Rahmani et al. [25] incorporates JaCoCo to process coverage metrics and TRGeneration to visualize a control flow graph and assist the tester in determining the test input requirements to increase coverage. VIRTuM [26] is an IntelliJ JetBrains plugin that provides static and dynamic test-related metrics. Alemerien and Magel [27] list various coverage tools they assess in their study, determining that there is a wide range of differences in how the metrics are calculated. Open Code Coverage Framework (OCCF) [28] proposes a framework to unify code coverage across many programming languages.

In contrast, our solution is VR-based and thus immersive and 3D, leverages requirements in text form via ReqIF, yet supports additional requirements modeling in ArchiMate (which provides broad modeling support), supports cross-diagram traceability, and integrates test dependency and test coverage for enhanced V&V traceability. In utilizing available standardized formats such as ReqIF and ArchiMate, to support a non-proprietary and tool-independent integration platform. As they are non-standardized, any tool-generated code coverage or test report format can be converted into our import format and utilized.

III. Solution Concept

A. Solution Positioning

While the solution concept in this paper is focused on V&V and specifically traceability support, our other solution concepts that address other aspects in the Software Engineering (SE) and Enterprise Architecture (EA) area are shown in Figure 1. VR-V&V utilizes our generalized VR Modeling Framework (VR-MF) (detailed in [11]). 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-based solutions specific to SE (VR-SE) include: VR-V&V (the focus of this paper, shown in black), which extends VR-TestCoverage [1], and VR-Git [29]. In the modeling area, VR-UML [9] and VR-SysML [10] and VR-SysML+Traceability [15]. Modeling support extending into the EA area includes VR-EA [11], which visualizes EA ArchiMate models in VR; VR-ProcessMine [30] supports process mining and analysis in VR; and VR-BPMN [31] visualizes Business Process Modeling Notation (BPMN) models in VR. VR-EAT [32] integrates the EA Tool (EAT) Atlas to provide dynamically-generated EA diagrams in VR, while VR-EA+TCK [33] integrates Knowledge Management Systems (KMS) and/or Enterprise Content Management Systems (ECMS), and VR-EvoEA+BP [34].

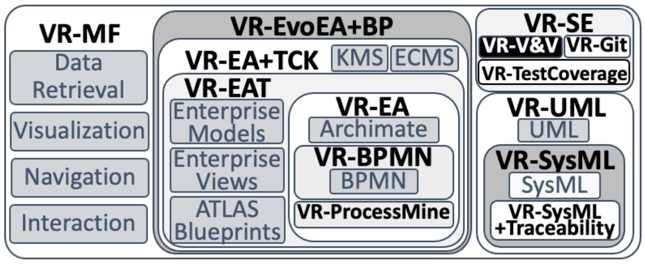


Figure 1. Conceptual map of our various VR solution concepts.

B. *V&V Considerations*

1) *Stakeholders:* While V&V typically involves a broad set of artifacts and activities, SW validation inherently involves and references requirements, while SW verification regarding realized SW elements will also typically involve or assess SW testing. Thus, V&V stakeholders are likely to require knowledge of requirements and the ability to assess SW testing coverage, as they are an essential part of V&V assessments. V&V activities may be performed by independent personnel, known as Independent V&V (IV&V), and these stakeholders may not be as familiar with the requirements, various internals of the SW architecture, and associated tests. Thus, an intuitive visualization and accessibility of relevant information and tracing can be supportive for such stakeholders.

2) *Testing:* Software testing is also a Knowledge Area (KA) within the SWEBOK [35]. Both the SWEBOK and the international software testing standard ISO/IEC/IEEE 29119 [36] include test coverage measures within their test technique descriptions. Test effectiveness is always a challenging factor to measure. While test coverage (a.k.a. code coverage, in this paper we assume statement coverage) as a single factor may not be strongly correlated with test effectiveness [37], it nevertheless is still low to moderately correlated, and this can be helpful and supportive data for the test effort and verification. Considering the adoption rate of test coverage by software developers, for an insight into the industrial popularity of test coverage, of 512 developers randomly surveyed at Google in a 2019 survey [38], 45% indicated they use it (very) often when authoring a changelist and 25% sometimes. When reviewing a changelist, 40% use coverage (very) often and 28% sometimes. Only 10% of respondents never use coverage, which conversely means 90% do. So overall, a substantial number of developers apply code coverage regularly and find value in it. Voluntary adoption at the project level went from 20% in 2015 to over 90% by 2019. As to modeling tests (or test modeling), while a UML Test Profile is available to extend UML, its industrial usage is relatively rare, since the expense of modeling and realizing the solution often exact the project effort and budget, and typically the preference is for utilizing the testing budget for writing and executing tests, rather than expending effort on the modeling of tests, which don't actually expand the testing coverage. The first form of traceability to support verification is to determine which test actually tested which test target. This type of verification is often not performed nor supported by test tooling. Usually, if code coverage is utilized, it usually does not offer a detailed assessment of exactly which test reached which test target line, but rather a summary of which lines or branches were reached via some test suite. Thus, the bidirectional traceability data is typically missing between unit test and test target, and is usually assumed using the test names.

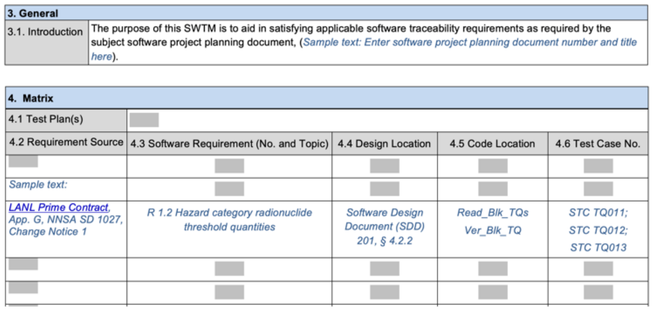
3) *Complexity:* A V&V visual scalability challenge is that with increasing digitalization, the software scope, capabilities, and features often increase, resulting in increases to requirements, code size, and complexity. Codebases can grow and become very large for software projects, be they open-source, commercial, or other organizations, as exemplified with the over 2 billion Lines of Code (LOC) across 9 million source files in a single monolithic repository accessed by 25k developers at Google [39]. There are estimated to be over 25m professional software developers worldwide [40] who continue to add source code to private and public repositories. One quality aspect to consider is how well this code is tested, and if any codebase changes have been covered by tests. With large code bases, visualization of test coverage can provide helpful insights, especially into what is not covered. As software projects grow in size and complexity, an immersive digital environment can provide an additional visualization capability to comprehend and analyze both the software production code (i.e., test target) and the software test suite and how they relate, as well as determine areas where the code coverage achieved by a test suite is below expectations.

4) *Modeling:* With regard to the choice of modeling notations, ArchiMate has a much broader modeling scope than UML (and SysML, which extends UML via a profile), overlapping many modeling notations and thus is able to act as a bridge across modeling notations. Besides requirements, ArchiMate also supports modeling externally relevant aspects such as behavior, interfaces, deployment, and infrastructure and how it may interact with other external systems. While UML entails approximately 150 modeling concepts, compared to the approximately 50 in ArchiMate, ArchiMate is relative lightweight, and its simplicity and broad support for enterprise and business modeling suggests it can perhaps more flexibly support the use of requirements with design models. Whereas UML is constrained to fixed

diagram types, ArchiMate permits custom stakeholder-oriented views. UML is object-oriented (OO), whereas ArchiMate is not constrained in this way, and has, e.g., separate service and interface concepts. As to requirements, UML offers primarily use cases. In contrast, The Open Group's Agile Guide for using ArchiMate [41] explicitly mentions modeling user stories, where "epics" (modeled as outcomes), which in turn are realized by "features" (modeled as requirements), which are themselves aggregated from individual "stories" (also modeled as requirements). However, both modeling notations can be used together, as described in by an Open Group whitepaper [42]. Thus, while our solution concept is design model notation agnostic, for demonstration purposes, our prototype realization will focus on ArchiMate.

5) Requirements: Software requirements is a KA within the Software Engineering Body Of Knowledge (SWEBOK) [35]. Both the SWEBOK and the requirements engineering process ISO/IEC/IEEE 29148 [43] mention requirements tracing and traceability, also in conjunction with requirements validation. We selected ArchiMate for requirements traceability modeling, among other reasons for its ability to model actors, system goals, and associated requirements independent of the narrow concept of Use Cases, the only direct form of requirement support that UML offers. Furthermore, ArchiMate offers various motivation elements such as Principles, Constraints, Value, Meaning, Outcome, Driver, Assessment in addition to Goal and Requirement, and these offer broad support for the typical concepts involved during requirements elicitation and related activities. While ArchiMate models support the modeling of such requirements concepts, typically they are nevertheless not the starting point for requirements. Rather, these are often formulated in text form, either more formally in a Software Requirements Specification (SRS) or System Requirements Specification (SyRS) that may be compliant with ISO/IEC/IEEE 29148, or these may come from more agile user stories or use cases. These requirements sources are thus not directly included or mapped in the design model such as ArchiMate. Since the modeling of requirements could incur errors, for V&V we thus consider the requirements source to be of principal character, and wish to have access to these sources in VR. Since ReqIF is a specified exchange format supported by requirements engineering tools, we chose to support importing ReqIF requirements into VR. This ensures that requirement information is complete and nothing is overlooked. This does not preclude the powerful requirements modeling support in ArchiMate, but rather supports V&V of such requirements modeling to the original source while remaining contextually immersed in VR.

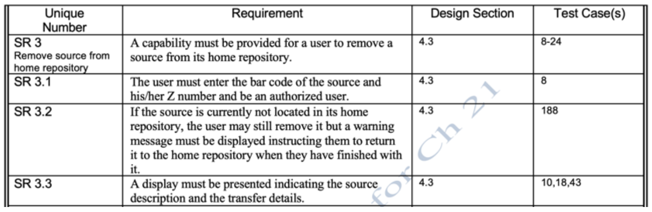
6) Traceability: The lack of a traceability standard or exchange format limits automation and tool accessibility. While vendors may have a proprietary solution, typically the inclusion of tracing information is a manual documentation effort utilizing spreadsheets or text documents, as exemplified in Figure 2 as a typical form template, and with a filled-in example in Figure 3.



3. General	
3.1. Introduction	The purpose of this SWTM is to aid in satisfying applicable software traceability requirements as required by the subject software project planning document, (*Sample text: Enter software project planning document number and title here*).

4. Matrix				
4.1 Test Plan(s)				
4.2 Requirement Source	4.3 Software Requirement (No. and Topic)	4.4 Design Location	4.5 Code Location	4.6 Test Case No.
Sample text:				
LANL Prime Contract, App. G, NNSA SD 1027, Change Notice 1	*R 1.2 Hazard category radionuclide threshold quantities*	*Software Design Document (SDD) 201, § 4.2.2*	*Read_Blk_TQs Ver_Blk_TQ*	*STC TQ011; STC TQ012; STC TQ013*

Figure 2. Screenshot from a V&V traceability matrix form [5].



Unique Number	Requirement	Design Section	Test Case(s)
SR 3 Remove source from home repository	A capability must be provided for a user to remove a source from its home repository.	4.3	8-24
SR 3.1	The user must enter the bar code of the source and his/her Z number and be an authorized user.	4.3	8
SR 3.2	If the source is currently not located in its home repository, the user may still remove it but a warning message must be displayed instructing them to return it to the home repository when they have finished with it.	4.3	188
SR 3.3	A display must be presented indicating the source description and the transfer details.	4.3	10,18,43

Figure 3. Screenshot of an example filled-in V&V traceability matrix [6].

We thus inserted tracing information manually.

C. Data Retrieval

Our solution concept includes a data hub, which is used to handle the importing, adapting, and storing of data for internal VR access. It supports the import of XML-based ArchiMate Model Exchange File Format [44] and ReqIF files to an internal JSON format stored in a local database accessible to the VR implementation.

D. Visualization in VR

A plane is used to group the production code (test suite target) as well as the test suite. A tree map using a step pyramid paradigm (or mountain range) is used to stack containers (i.e., groups, collections, folders, directories, packages) in the third dimension (height) on the plane.

For modeling test target to test source dependencies, a visualization challenge was that we initially thought we could depict the test target code by simply overlaying a layer on the production code and indicating which test "covered" what production code. However, once we completed the dependency analysis of large projects, we found that while one test may have a test target focus, it nevertheless may indirectly invoke various other dependent portions of the test target, resulting in n-m relations between tests and the test targets. This quickly becomes visually cluttered. Thus, we chose to keep the visual depiction of the test suite separated from the test target (since it can have its own hierarchical organization), yet apply the same visualization paradigm to depict "containers" or collections as packages or folders.

E. Navigation in VR

The space that can be traversed in VR can become quite large, whereas the physical space of the VR user may be constrained, e.g., to a desk. Thus, the left controller is used for controlling flight (moving the VR camera), while the right controller is used for interaction.

F. *Interaction in VR*

Since interaction in VR is not yet standardized, in our concept, user-element interaction is supported primarily through the VR controllers and our *VR-Tablet*. The VR-Tablet is used to provide context-specific detailed element information, supporting an internet browser for access to any documentation. It provides a *virtual keyboard* for text entry via laser pointer key selection. While it may be potentially cumbersome to enter text via a virtual keyboard in VR compared to a real keyboard, most V&V traceability analysis will likely be focused on confirming or marking or noting issues. A potential workaround would be to record the audio during the immersion and then transcribe the notes outside of VR. Our solution could be readily extended to add annotation capabilities to elements directly.

IV. Realization

To avoid redundancy, only realization aspects not explicitly mentioned in the concept or in the evaluation sections are described in this section.

The logical architecture of our VR implementation is shown in Figure 4. VR was realized with Unity and tested with HTC Vive. Internally, besides any localized intra-model graphs, a MetaGraph script is used to determine and model both inter- and intra-relations (edges) between elements (nodes) to support bidirectional tracing across any elements or models. All exported data is stored in the data hub and accessed via scripts from Unity. The JSONUtility library was used for JSON processing.

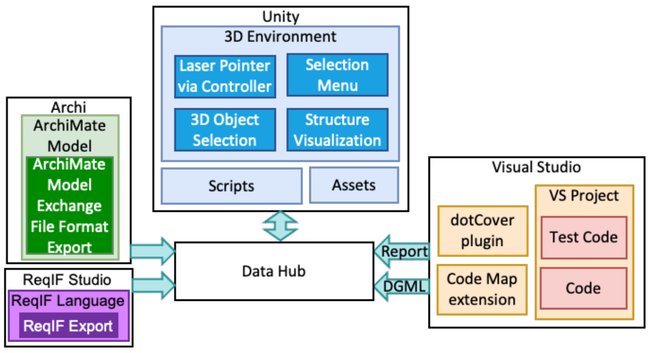


Figure 4. VR-V&V logical architecture.

A. *Requirements Traceability with ReqIF and ArchiMate*

While our VR-V&V requirements traceability solution concept is generic, for the prototype demonstration we focused on supporting ReqIF and ArchiMate. As shown in Figure 5, the content of an XML-based ReqIF file consists of a ReqIF Header, ReqIF ToolExtensions, and ReqIF CoreContent. CoreContent consists of primitive strongly-typed Datatypes definitions (String, Boolean, Integer, Real, Enumeration, Date, and XHTML that can include an image). SpecTypes are used to define requirement types, such as functional, quality, performance requirements, including their attributes and possible relationship types. A SpecObject is an actual requirement, and acts as container for a requirement and holds user-defined attributes, each of a specific SpecType and/or Datatype. SpecRelations represent relationships between SpecObjects and can have attributes. Specifications are a structured view of SpecObjects using hierarchical trees. A RelationGroup can be used to group relationships. The ReqIFSharp [45] library was used for importing ReqIF files.

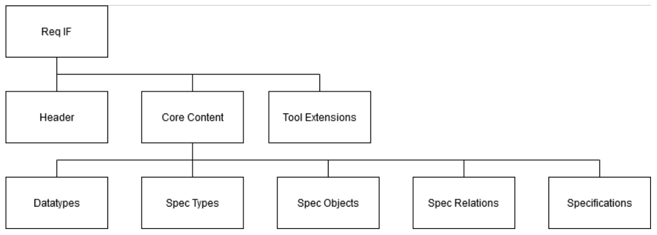
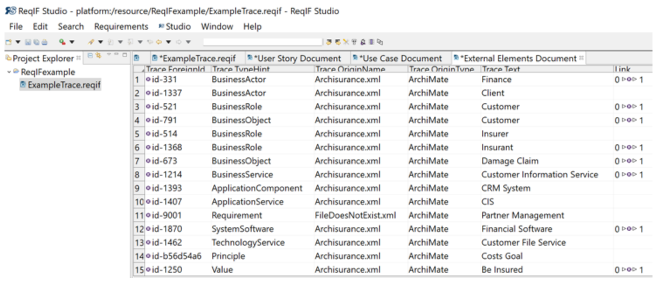


Figure 5. ReqIF structure.

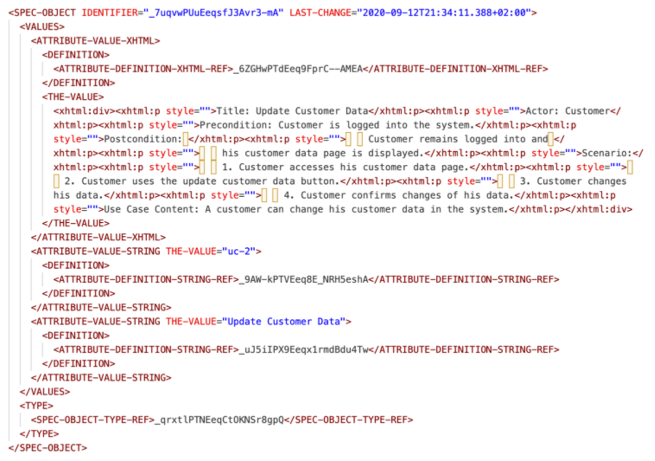
The ArchiMate Exchange File (XF) format consists of three XML Schema Definitions (XSDs) that build on or include the prior: first model, then view, and then diagram exchange. Since ArchiMate is much more involved and is a full enterprise modeling language, the EF is also much more involved and won't be described here, we refer to [44] for more information.

To demonstrate traceability, in ReqIF Studio [46] ArchiMate element ID strings were manually added to ReqIF SpecObjects as External Elements: attribute TraceForeignId (e.g., id-791), TraceTypeHint (e.g., BusinessActor), TraceOriginName (filename), TraceOriginType (e.g., ArchiMate), and Trace Text to optionally name the trace, as shown in Figure 6.



	Trace ForeignId	Trace TypeHint	Trace OriginName	Trace OriginType	Trace Text	Link
1	id-331	BusinessActor	Archisurance.xml	ArchiMate	Finance	0 ▷●▷ 1
2	id-1337	BusinessActor	Archisurance.xml	ArchiMate	Client	
3	id-521	BusinessRole	Archisurance.xml	ArchiMate	Customer	0 ▷●▷ 1
4	id-791	BusinessObject	Archisurance.xml	ArchiMate	Customer	0 ▷●▷ 1
5	id-514	BusinessRole	Archisurance.xml	ArchiMate	Insurer	
6	id-1368	BusinessRole	Archisurance.xml	ArchiMate	Insurant	0 ▷●▷ 1
7	id-673	BusinessObject	Archisurance.xml	ArchiMate	Damage Claim	0 ▷●▷ 1
8	id-1214	BusinessService	Archisurance.xml	ArchiMate	Customer Information Service	0 ▷●▷ 1
9	id-1393	ApplicationComponent	Archisurance.xml	ArchiMate	CRM System	
10	id-1407	ApplicationService	Archisurance.xml	ArchiMate	CIS	
11	id-9001	Requirement	FileDoesNotExist.xml	ArchiMate	Partner Management	
12	id-1870	SystemSoftware	Archisurance.xml	ArchiMate	Financial Software	0 ▷●▷ 1
13	id-1462	TechnologyService	Archisurance.xml	ArchiMate	Customer File Service	
14	id-b56d54a6	Principle	Archisurance.xml	ArchiMate	Costs Goal	
15	id-1250	Value	Archisurance.xml	ArchiMate	Be Insured	0 ▷●▷ 1

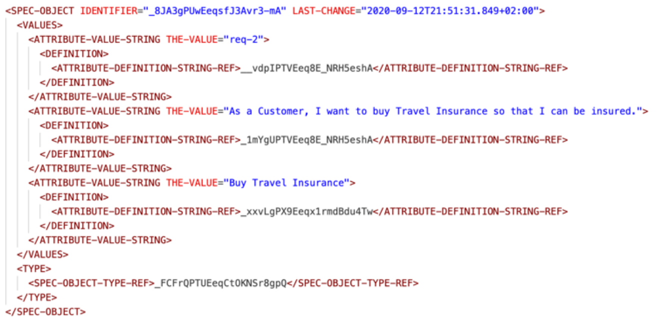
Figure 6. ReqIF Studio External Elements used for ArchiMate tracing.



```
<SPEC-OBJECT IDENTIFIER="_7uqvwPUuEeqsfJ3Avr3-mA" LAST-CHANGE="2020-09-12T21:34:11.388+02:00">
  <VALUES>
    <ATTRIBUTE-VALUE-XHTML>
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-XHTML-REF>_6ZGHwPTdEeq9FprC--AMEA</ATTRIBUTE-DEFINITION-XHTML-REF>
      </DEFINITION>
      <THE-VALUE>
        <xhtml:div><xhtml:p style="">Title: Update Customer Data</xhtml:p><xhtml:p style="">Actor: Customer</
        xhtml:p><xhtml:p style="">Precondition: Customer is logged into the system.</xhtml:p><xhtml:p
        style="">Postcondition:▯</xhtml:p><xhtml:p style="">▯ ▯ Customer remains logged into and▯</
        xhtml:p><xhtml:p style="">▯ ▯ his customer data page is displayed.</xhtml:p><xhtml:p style="">Scenario:</
        xhtml:p><xhtml:p style="">▯ ▯ 1. Customer accesses his customer data page.</xhtml:p><xhtml:p style="">▯
        ▯ 2. Customer uses the update customer data button.</xhtml:p><xhtml:p style="">▯ ▯ 3. Customer changes
        his data.</xhtml:p><xhtml:p style="">▯ ▯ 4. Customer confirms changes of his data.</xhtml:p><xhtml:p
        style="">Use Case Content: A customer can change his customer data in the system.</xhtml:p></xhtml:div>
      </THE-VALUE>
    </ATTRIBUTE-VALUE-XHTML>
    <ATTRIBUTE-VALUE-STRING THE-VALUE="uc-2">
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-STRING-REF>_9AW-kPTVEeq8E_NRH5eshA</ATTRIBUTE-DEFINITION-STRING-REF>
      </DEFINITION>
    </ATTRIBUTE-VALUE-STRING>
    <ATTRIBUTE-VALUE-STRING THE-VALUE="Update Customer Data">
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-STRING-REF>_uJ5iIPX9Eeqx1rmdBdu4Tw</ATTRIBUTE-DEFINITION-STRING-REF>
      </DEFINITION>
    </ATTRIBUTE-VALUE-STRING>
  </VALUES>
  <TYPE>
    <SPEC-OBJECT-TYPE-REF>_qrxtlPTNEeqCtOKNSr8gpQ</SPEC-OBJECT-TYPE-REF>
  </TYPE>
</SPEC-OBJECT>
```

Figure 7. ReqIF file snippet example of use case.

Alternatively, Requirement IDs could also be added to ArchiMate element properties to refer to the requirement. These ReqIF attributes were utilized by the MetaGraph to determine the traces. An example use case snippet from a ReqIF file is shown in Figure 7. A user story example ReqIF file snippet is shown in Figure 8.



```
<SPEC-OBJECT IDENTIFIER="_8JA3gPUwEeqsfJ3Avr3-mA" LAST-CHANGE="2020-09-12T21:51:31.849+02:00">
  <VALUES>
    <ATTRIBUTE-VALUE-STRING THE-VALUE="req-2">
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-STRING-REF>__vdpIPTVEeq8E_NRH5eshA</ATTRIBUTE-DEFINITION-STRING-REF>
      </DEFINITION>
    </ATTRIBUTE-VALUE-STRING>
    <ATTRIBUTE-VALUE-STRING THE-VALUE="As a Customer, I want to buy Travel Insurance so that I can be insured.">
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-STRING-REF>_1mYgUPTVEeq8E_NRH5eshA</ATTRIBUTE-DEFINITION-STRING-REF>
      </DEFINITION>
    </ATTRIBUTE-VALUE-STRING>
    <ATTRIBUTE-VALUE-STRING THE-VALUE="Buy Travel Insurance">
      <DEFINITION>
        <ATTRIBUTE-DEFINITION-STRING-REF>_xxvLgPX9Eeqx1rmdBdu4Tw</ATTRIBUTE-DEFINITION-STRING-REF>
      </DEFINITION>
    </ATTRIBUTE-VALUE-STRING>
  </VALUES>
  <TYPE>
    <SPEC-OBJECT-TYPE-REF>_FCFrQPTUEeqCtOKNSr8gpQ</SPEC-OBJECT-TYPE-REF>
  </TYPE>
</SPEC-OBJECT>
```

Figure 8. ReqIF file snippet example of user story.

To compare the realization of our VR visualization of an ArchiMate model, we use the ArchiSurance [47] example. The 2D view available in Archi [48] is shown in Figure 9. The equivalent ArchiMate diagram in VR-V&V is shown in Figure 10. Our backplane concept, with randomly colored traces of elements that exist on other diagram planes, is depicted in Figure 11. To reduce visual clutter across the diagrams, these leave from below the element and go to a backplane on which all ArchiMate diagrams are aligned, allowing one to follow a trace between diagram planes. This capability is not available in typical ArchiMate tools that offer 2D views.

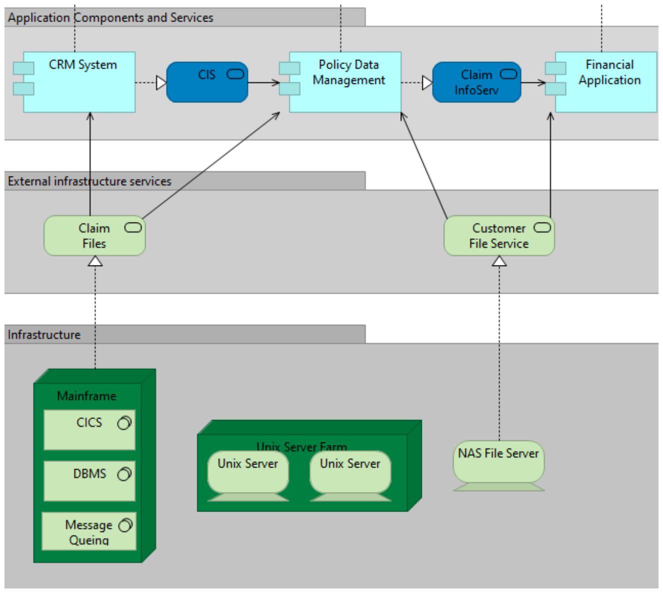


Figure 9. Screenshot of partial model of ArchiMate ArchiSurance example in the desktop Archi tool.

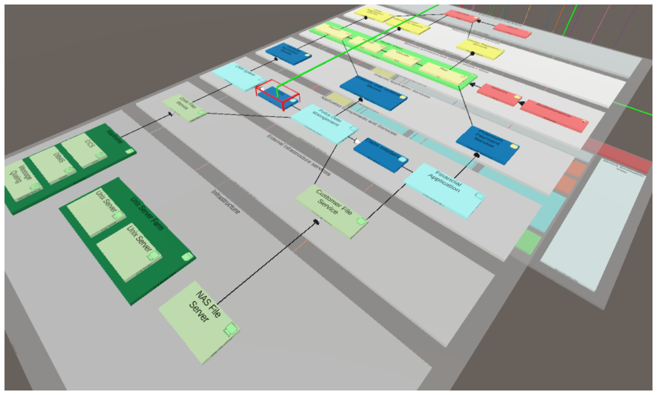


Figure 10. ArchiMate ArchiSurance example model in VR-V&V.

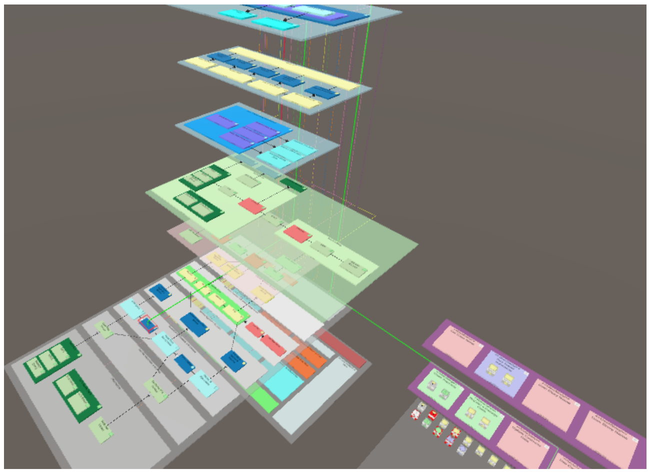
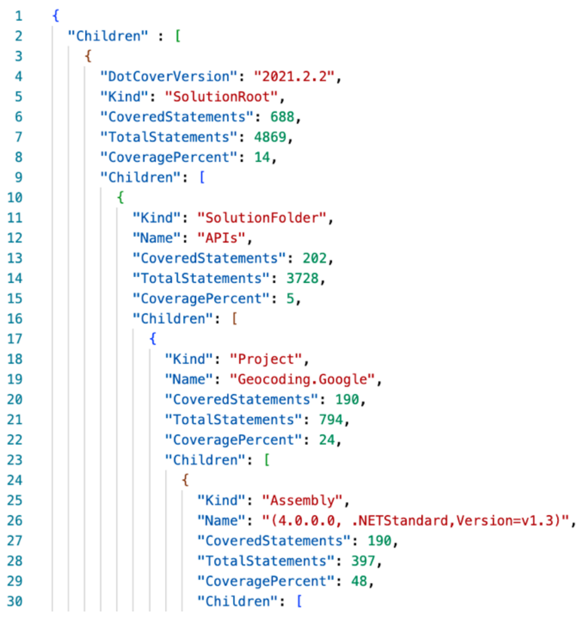


Figure 11. ArchiMate ArchiSurance example model in VR-V&V.

B. Test Tracing Realization

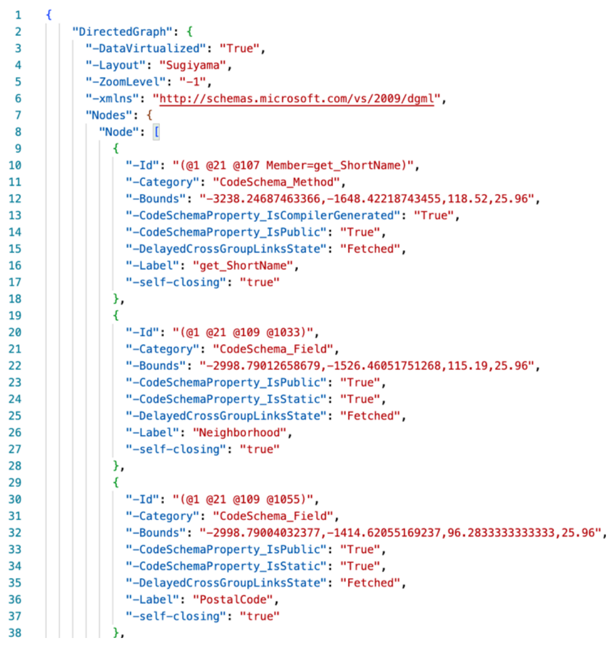
While our VR-V&V test coverage solution concept is generic, for the prototype demonstration we focused on the .NET platform. As a test coverage tool, we utilized JetBrains dotCover. This Microsoft Visual Studio plugin is a .NET Unit test runner and code coverage tool that can generate a statement coverage report in JSON, XML, etc. as shown in Figure 12. While it is a static analysis tool, it can also import coverage reports. A challenge we faced is that among the coverage tools we considered, they only report on dependencies between test targets, and do not explicitly indicate or name direct dependencies to the invoking test.

Thus, to determine C# code dependencies, Visual Studio 2022 Enterprise Edition (EE) was used, which provides a Code Map that is stored as a Directed Graph Markup Language (DGML) file. Its XML-like format is converted to JSON as shown in Figure 13. This dependency report is then partitioned into a node report and a link report. Only direct dependencies between test and test target are considered, otherwise the dependency structure could readily become very complex with large sets of intermediate nodes and their interdependencies.



```
{
  "Children" : [
    {
      "DotCoverVersion": "2021.2.2",
      "Kind": "SolutionRoot",
      "CoveredStatements": 688,
      "TotalStatements": 4869,
      "CoveragePercent": 14,
      "Children": [
        {
          "Kind": "SolutionFolder",
          "Name": "APIs",
          "CoveredStatements": 202,
          "TotalStatements": 3728,
          "CoveragePercent": 5,
          "Children": [
            {
              "Kind": "Project",
              "Name": "Geocoding.Google",
              "CoveredStatements": 190,
              "TotalStatements": 794,
              "CoveragePercent": 24,
              "Children": [
                {
                  "Kind": "Assembly",
                  "Name": "(4.0.0.0, .NETStandard,Version=v1.3)",
                  "CoveredStatements": 190,
                  "TotalStatements": 397,
                  "CoveragePercent": 48,
                  "Children": [
```

Figure 12. DotCover coverage report snippet for the Geocoding.net project.



```
{
  "DirectedGraph": {
    "-DataVirtualized": "True",
    "-Layout": "Sugiyama",
    "-ZoomLevel": "-1",
    "-xmlns": "http://schemas.microsoft.com/vs/2009/dgml",
    "Nodes": {
      "Node": [
        {
          "-Id": "(@1 @21 @107 Member=get_ShortName)",
          "-Category": "CodeSchema_Method",
          "-Bounds": "-3238.24687463366,-1648.42218743455,118.52,25.96",
          "-CodeSchemaProperty_IsCompilerGenerated": "True",
          "-CodeSchemaProperty_IsPublic": "True",
          "-DelayedCrossGroupLinksState": "Fetched",
          "-Label": "get_ShortName",
          "-self-closing": "true"
        },
        {
          "-Id": "(@1 @21 @109 @1033)",
          "-Category": "CodeSchema_Field",
          "-Bounds": "-2998.79012658679,-1526.46051751268,115.19,25.96",
          "-CodeSchemaProperty_IsPublic": "True",
          "-CodeSchemaProperty_IsStatic": "True",
          "-DelayedCrossGroupLinksState": "Fetched",
          "-Label": "Neighborhood",
          "-self-closing": "true"
        },
        {
          "-Id": "(@1 @21 @109 @1055)",
          "-Category": "CodeSchema_Field",
          "-Bounds": "-2998.79004032377,-1414.62055169237,96.2833333333333,25.96",
          "-CodeSchemaProperty_IsPublic": "True",
          "-CodeSchemaProperty_IsStatic": "True",
          "-DelayedCrossGroupLinksState": "Fetched",
          "-Label": "PostalCode",
          "-self-closing": "true"
        },
```

Figure 13. Code Map snippet (in JSON) for the Geocoding.net project for determining dependencies.

With regard to VR visualization, to attempt to retain the intuitive paradigm of test "coverage," we elected to place the test suite visualization directly above the test target, rather than on the sides as shown in Figure 14. That way, dependencies can be followed from top to bottom during VR navigation. Since the most concrete tests are typically the smallest (greatest depth being the structural leaves), the test suite uses depth to bring these closer to its target. Dependencies are then shown as lines between the test and test target, analogous to puppet strings. A selected line can be either highlighted or alternatively configured to ghost all others.

Testers focused on test coverage are typically concerned about the overall coverage (e.g., to compare its level against some high-level test goal), while also concerned about assessing details and risks as to which areas were not covered by tests. Thus, in VR our visualization of the System Under Test (SUT) or test target is shown on a plane using stepped pyramids for a 3D effect, with the coverage percentages for a container (folder, directory, package) shown on each side.

The lowest level container is on the bottom and represents the entire project. The test suite is projected above this onto a separate plane and upside-down, also using stepped pyramids for containers.

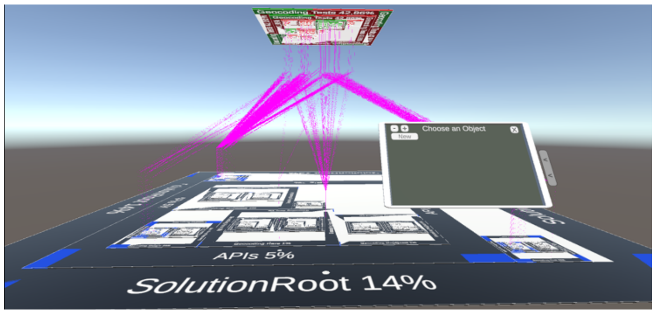


Figure 14. VR-V&V: test suite and test results visible on top, test target and code coverage shown on bottom; the VR-Tablet is visible on the right as are dependencies (magenta lines).

The test coverage of the test targets is indicated via a bar on all four sides so that from any perspective the coverage is visually indicated as seen in Figure 15. A bar graph is used on all sides, with blue visually indicating the percentage of coverage and black used for the rest (the exact coverage percentage is also shown numerically). A stepped pyramid paradigm is used to portray the granularity, with the highest cubes having the finest granularity or depth, and the lowest being the least granular. For instance, a user can quickly hone in on overall areas with little to no blue, meaning that coverage there was scarce, and one can quickly find and focus on details (without losing the overview) by focusing on the higher elevations.

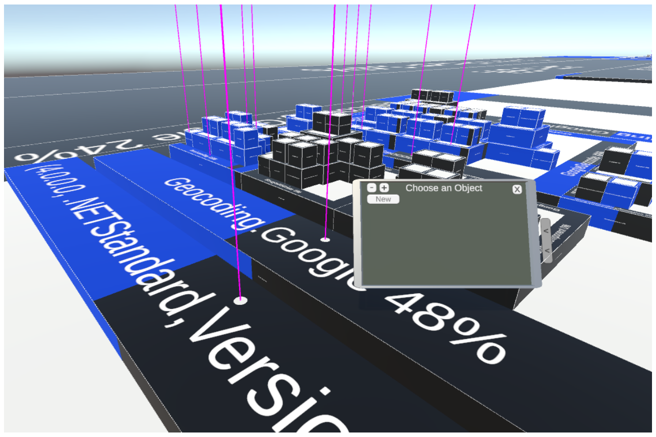


Figure 15. VR-V&V: test coverage showing stepped pyramid with highest points being finest granularity.

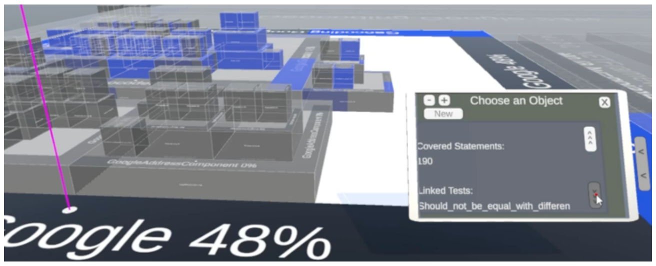


Figure 16. VR-Tablet showing coverage report details for the selected element (non-selected elements become transparent).

Selecting a test target element causes all other target elements and unassociated dependency links to become transparent, while element-relevant details from the coverage report can be inspected in the VR-Tablet as shown in Figure 16.

V. Evaluation

We base the evaluation of our solution concept on design science method and principles [49], in particular, a viable artifact, problem relevance, and design evaluation (utility, quality, efficacy). To evaluate our prototype realization of our solution concept, a case study is used based on two main scenarios supporting V&V: 1) requirements and design model tracing, and 2) test coverage and test source tracing.

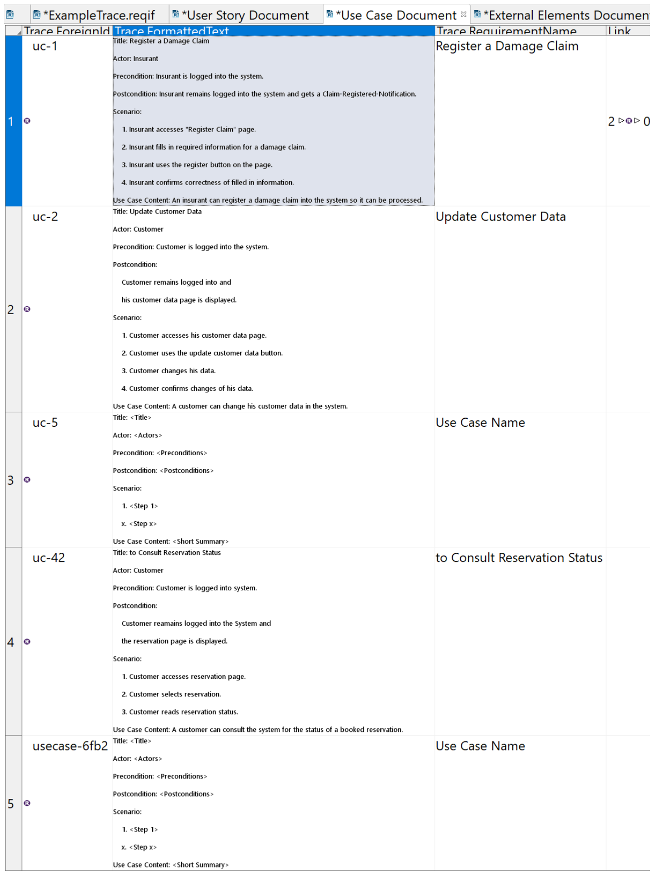
A. Requirements Traceability with ArchiMate Scenario

For an example ArchiMate design model we used the ArchiSurance [47] to demonstrate requirements traceability between ReqIF-based requirements and an ArchiMate model in VR. The requirements examples used are shown in ReqIF Studio for use cases in Figure 17 and user stories in Figure 18. User stories and use cases are written as plain text.

For visualization in VR, requirements specified in a ReqIF file are placed on purple planes, with either all on a single plane or split across multiple planes if desired. A requirement can be a use case (denoted with the stereotype <<UseCase>>), a user story (denoted by the stereotype <<UserStory>>), or any other type of requirement. The color of the requirement indicates the number of relations with external elements:

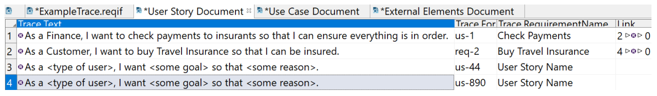
- if none, it is colored red (perceived as peach here, as a warning the requirement has no trace and may be unaddressed);
- if it has at least one relation and all elements are shown, it is green (see Figure 19);
- and if it relates to more elements than can be shown on the plane, it is blue.

This can be seen in Figure 20, where only two elements are seen in the requirement, but once selected, actually four elements are traced to the external elements plane below. A selected element is outlined with red.


*ExampleTrace.reqif | *User Story Document | *Use Case Document | *External Elements Document

	Trace ForeignId	Trace FormattedText	Trace RequirementName	Link
1	uc-1	Title: Register a Damage Claim Actor: Insurant Precondition: Insurant is logged into the system. Postcondition: Insurant remains logged into the system and gets a Claim-Registered-Notification. Scenario: 1. Insurant accesses "Register Claim" page. 2. Insurant fills in required information for a damage claim. 3. Insurant uses the register button on the page. 4. Insurant confirms correctness of filled in information. Use Case Content: An insurant can register a damage claim into the system so it can be processed.	Register a Damage Claim	2 ▷ ▷ 0
2	uc-2	Title: Update Customer Data Actor: Customer Precondition: Customer is logged into the system. Postcondition: Customer remains logged into and his customer data page is displayed. Scenario: 1. Customer accesses his customer data page. 2. Customer uses the update customer data button. 3. Customer changes his data. 4. Customer confirms changes of his data. Use Case Content: A customer can change his customer data in the system.	Update Customer Data	
3	uc-5	Title: <Title> Actor: <Actors> Precondition: <Preconditions> Postcondition: <Postconditions> Scenario: 1. <Step 1> x. <Step x> Use Case Content: <Short Summary>	Use Case Name	
4	uc-42	Title: to Consult Reservation Status Actor: Customer Precondition: Customer is logged into system. Postcondition: Customer reamains logged into the System and the reservation page is displayed. Scenario: 1. Customer accesses reservation page. 2. Customer selects reservation. 3. Customer reads reservation status. Use Case Content: A customer can consult the system for the status of a booked reservation.	to Consult Reservation Status	
5	usecase-6fb2	Title: <Title> Actor: <Actors> Precondition: <Preconditions> Postcondition: <Postconditions> Scenario: 1. <Step 1> x. <Step x> Use Case Content: <Short Summary>	Use Case Name	

Figure 17. Screenshot of a use case document in ReqIF Studio.


*ExampleTrace.reqif | *User Story Document | *Use Case Document | *External Elements Document

	Trace Text	Trace For	Trace RequirementName	Link
1	As a Finance, I want to check payments to insurants so that I can ensure everything is in order.	us-1	Check Payments	2 ▷ ▷ 0
2	As a Customer, I want to buy Travel Insurance so that I can be insured.	req-2	Buy Travel Insurance	4 ▷ ▷ 0
3	As a <type of user>, I want <some goal> so that <some reason>.	us-44	User Story Name	
4	As a <type of user>, I want <some goal> so that <some reason>.	us-890	User Story Name	

Figure 18. Screenshot of a user story document in ReqIF Studio.

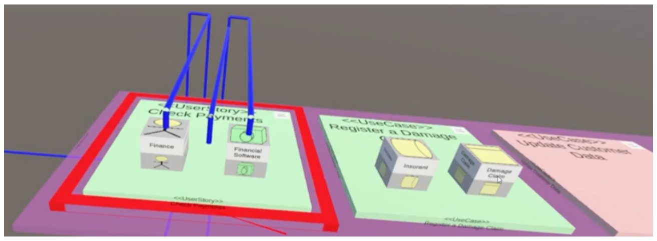


Figure 19. Selected user story (left) with requirements traces (blue) showing referenced ArchiMate elements; further use cases (to its right) depicted on requirements plane.

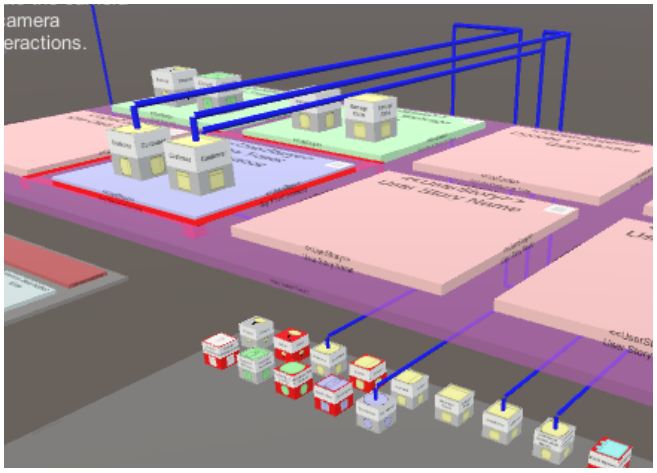


Figure 20. Selecting a blue requirement highlights additional external elements.

As shown in Figure 21, external elements specified in the ReqIF file are placed on a gray external elements plane shown below the requirements plane, and used to relate requirements elements to ArchiMate elements. Any external elements that are missing links or relations to a requirement (SpecObject, e.g., Use Case / User Story) have red colored cubes to draw attention a potential issue, while elements with satisfied associations remain gray cubes to not draw attention. Since an ArchiMate model is present and these elements are linked to it, those icons are used to indicate the type. If an element is non-existent in the ArchiMate model, then a question mark is used as its icon (the VR-tablet will still provide access to its details, the available text as provided in the ReqIF file).

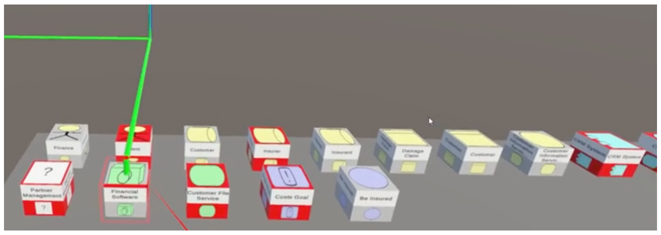


Figure 21. External elements plane: Financial Software element selected (outlined in red), green trace to location on other planes; bottom left shows an unassociated element (red) of unknown type (question mark icon).

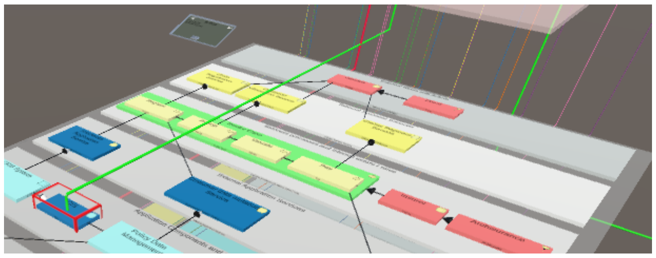


Figure 22. Highlighted ArchiMate element (red box) and green trace above.

On ArchiMate diagrams in VR, any connectors on the ArchiMate diagrams (that lie flat) have the same layout and meaning as in ArchiMate. We introduced backplane traces (see Figure 22 and Figure 23) to link and make apparent identical elements on different diagram planes. These backplane traces are randomly multi-colored and trace to the same identity elements on other diagrams and depart below the element and follow along a backplane to reduce clutter.

Selecting an external item (Figure 21) or an ArchiMate element referenced within a requirement (Figure 23) will highlight the element itself and all other elements representing the underlying artefact in the MetaGraph with an outlined red box (Figure 24). This will also produce a green trace line departing above the element linking these same elements across diagrams. A blue trace line also links the containing requirement to the external element plane.

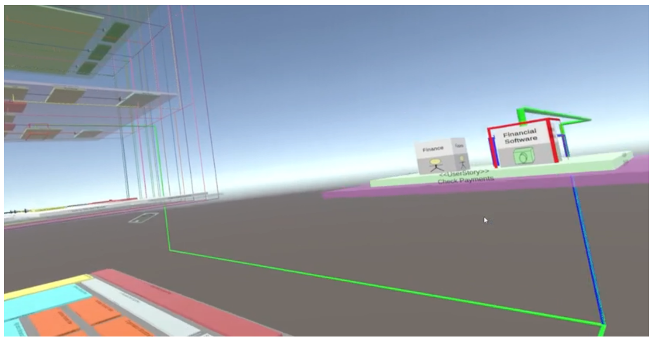


Figure 23. Financial Software element in requirement user story outlined in red with green trace to its location on other planes.

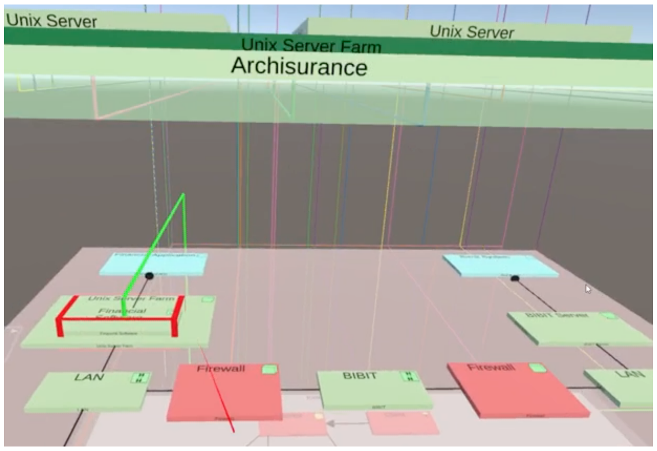
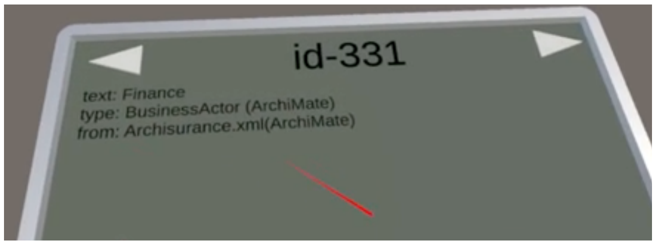


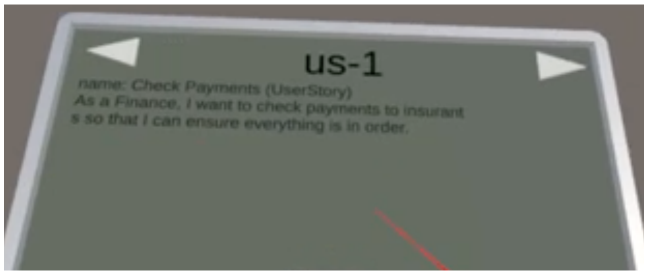
Figure 24. Financial Software element in ArchiMate diagram outlined in red with green trace its location on other planes.


id-331

text: Finance
type: BusinessActor (ArchiMate)
from: Archisurance.xml(ArchiMate)

Figure 25. VR-Tablet showing details for a selected external element (not associated here with a diagram).

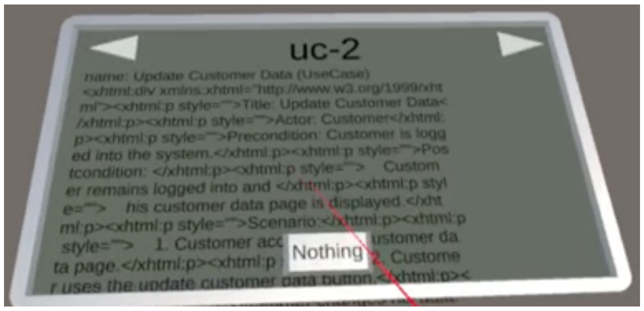
The VR-Tablet enables access to selected element details while in VR, such as an external element (see Figure 25), a user story (see Figure 26), a use case (see Figure 27), or an Archimate diagram element (see Figure 28).



us-1

name: Check Payments (UserStory)
As a Finance, I want to check payments to insurant
s so that I can ensure everything is in order.

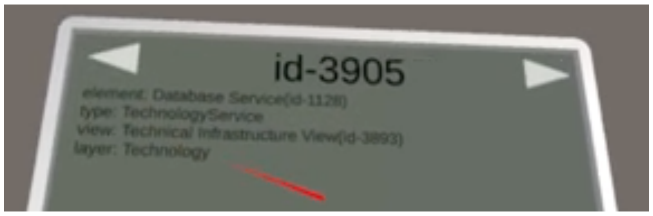
Figure 26. VR-Tablet showing details of a selected user story.



uc-2

```
name: Update Customer Data (UseCase)
<xhtml:div xmlns:xhtml="http://www.w3.org/1999/xht
ml"><xhtml:p style="">Title: Update Customer Data<
/xhtml:p><xhtml:p style="">Actor: Customer</xhtml:
p><xhtml:p style="">Precondition: Customer is logg
ed into the system.</xhtml:p><xhtml:p style="">Pos
tcondition: </xhtml:p><xhtml:p style="">    Custom
er remains logged into and </xhtml:p><xhtml:p styl
e="">    his customer data page is displayed.</xht
ml:p><xhtml:p style="">Scenario:</xhtml:p><xhtml:p
style="">    1. Customer acc Nothing ustomer da
ta page.</xhtml:p><xhtml:p      2. Custome
r uses the update customer data button.</xhtml:p><
```

Figure 27. VR-Tablet showing details of a selected use case.



id-3905

element: Database Service(id-1128)
type: TechnologyService
view: Technical Infrastructure View(id-3893)
layer: Technology

Figure 28. VR-Tablet showing details of an element selected on an ArchiMate diagram.

Thus, VR-V&V helps support trace analysis of requirements and associated elements to design model elements such as those in ArchiMate. It shows the feasibility of supporting V&V in VR, and that analogously other design modeling notations could be similarly supported. Hence, VR-V&V can support software comprehension for V&V traceability tasks by partially addressing invisibility, making links and traces visibile in the digital reality of VR, and addressing complexity limitations via the unlimited space available to present and visualize all design diagrams and requirements comprehensively.

B. *Test Tracing Scenario*

Our test coverage scenario considers V&V support for analyzing: 1) test results, 2) test coverage, and 3) test dependencies. For demonstration purposes, Geocoding.net [50] was used as an example C# project. However, any C# project could be used by the prototype, and currently any coverage tool could be used by mapping and transforming the report format to the DotCover JSON format.

1) *Test Result Visualization*

In VR, tests (and their containers) in the test suite (a.k.a. test source) are colored based on the test result status: green for successful, red if any test failed, and yellow for other (such as ignored). To depict test results and overall pass (or success) rate, the test suite is visualized as a tree map of all tests using a step pyramid for the third dimension to indicate granularity via depth. Analogously to how coverage was shown as a colored bar on all four sides of a container, on the test suite green is proportionally shown for success rate and red for failure (yellow for other), with its numerical value also given as shown in Figure 29.

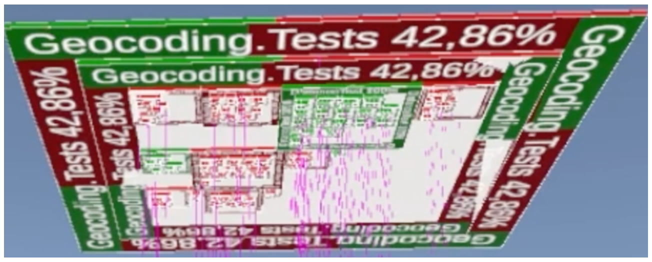


Figure 29. Test suite overview; bar indicates pass percentage for a collection (green for pass, red for failed).

A closeup view showing how test case and unit test information is provided, showing the test cases (lowest and closest to the test target), the test unit (showing name and percentage), and a test container (folder or directory) is provided in Figure 30. The VR-Tablet can be used to inspect the test results for a selected test.

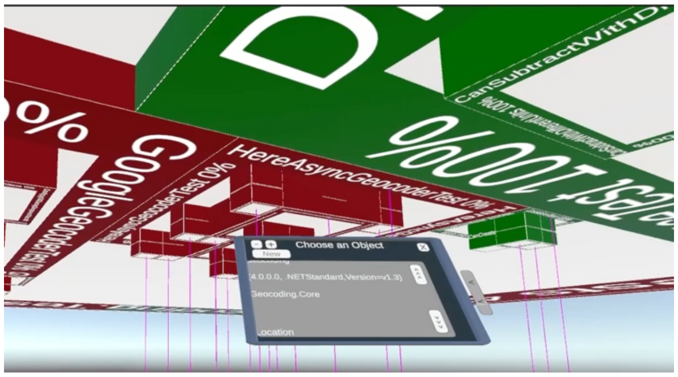


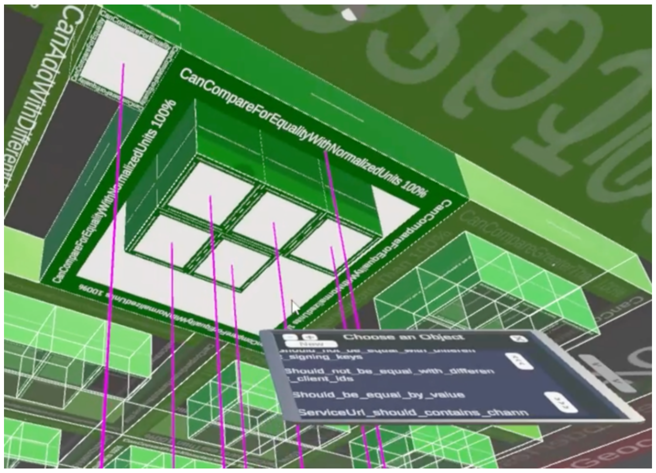
Figure 30. Test suite success shown by test case, test unit, and test container (folder or directory).

2) *Test Coverage Visualization*

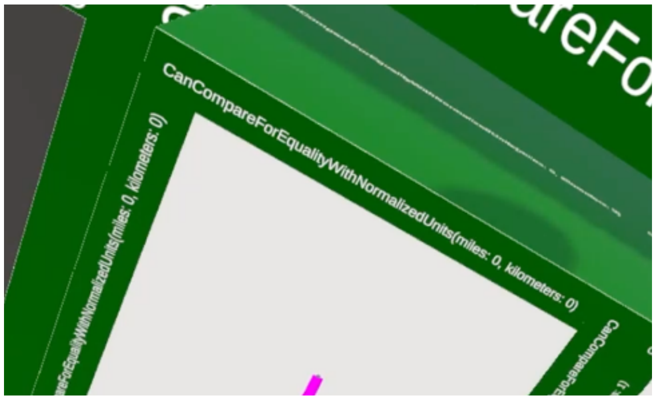
Coverage of the test targets is shown as a bar on all four sides and on the elevation, with the blue area visually indicating the percentage of coverage, and black used for the rest, with the percentage also shown numerically, as shown in Figure 15. Details on a coverage target can also be retrieved via the VR Tablet as shown in Figure 16.

3) *Test Dependency Trace Visualization*

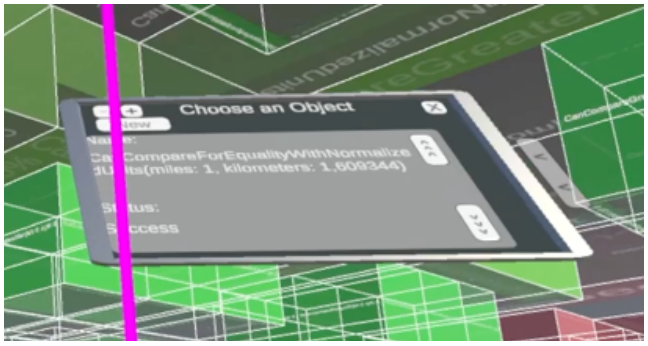
V&V support for test dependencies is typically not supported by test tools. Thus, VR-V&V supports a test dependency view, with which stakeholders can view which tests are directly invoking or reaching which target code. Typically, by convention tests are named in such a way to express the test target, yet the actual dependencies could nevertheless differ from expectations. This is especially true if the test suite consists not only of unit tests but also integration or system tests. By eliminating the guess work, dependencies could be used to determine which tests are primarily reaching a target, and then focus on extending that test in order to increase the target coverage. One challenge is that there is not necessarily a 1-1 match of a test to its test target, thus dependency links provide a way to visualize these hitherto hidden dependencies.


CanCompareForEqualityWithNormalizedUnits 100%
Choose an Object
Should_not_be_equal_with_differen t_client_ids
Should_be_equal_by_value
ServiceUrl_should_contains_chann
Figure 31. Magenta traces can be followed to dependent test cases.

VR-V&V depicts the test dependencies of a selected target as a magenta-colored trace line. When a selected test target or trace is followed, the associated test cases in the test suite are opaque, perceived as dark green in Figure 31. And tests can be followed to the most granular level of the test case which remain opaque as seen in Figure 32. Unassociated tests are made partially transparent (perceived as bright green, bright red, or grey).


CanCompareForEqualityWithNormalizedUnits(miles: 0, kilometers: 0)
Figure 32. Bottom view showing dependent test cases and pass rate.

The VR-Tablet can be used to inspect test report details about a selected test object, with the test method (CanCompareForEqualityWithNormalizedUnits), test data input values (miles: 1, kilometers: 1.609344), and test status (success) shown in Figure 33.


Choose an Object
New
CanCompareForEqualityWithNormalize dUnits(miles: 1, kilometers: 1.609344)
Status:
Success
Figure 33. VR-Tablet showing test case details.

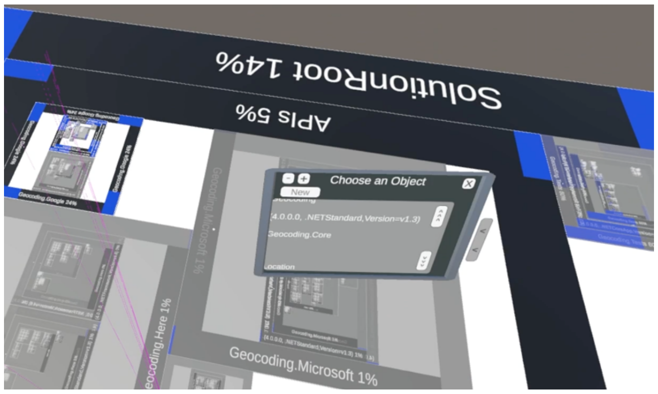


Figure 34. When a test element is selected, non-applicable target areas are greyed out.

These traces can be followed to the test target plane to determine what (sub)target(s) a selected test is actually reaching, as can be seen in Figure 34. The non-relevant test target areas are then partially transparent.

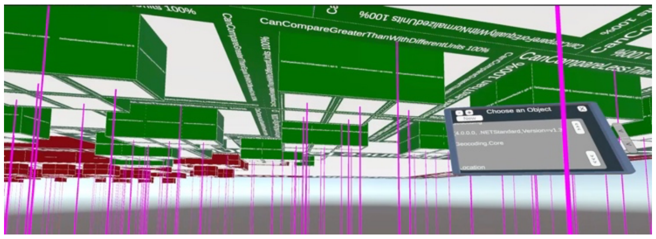


Figure 35. All test dependencies shown (by toggling selection).

By unselecting a test element, all dependency traces are restored and all test elements opaque, as can be seen in Figure 35. Thus, one can trace overall test groups or determine that certain tests are perhaps in preparation, or not (as yet) traceable or related to the test target if trace dependencies are missing. E.g., this might occur if tests were written before the production code has been implemented (e.g., in the case of acceptance test-driven techniques). Alternatively, this could be an indicator of a test suite and test target mismatch, perhaps if the production code was significantly changed without making associated changes to the test suite.

C. Discussion

The test tracing scenario shows the ability of VR-V&V to support test result tracing, test coverage, and test dependency tracing. To reduce test redundancy, measuring test target coverage can help focus test development on those areas that are not yet sufficiently tested or have the most risk. As software projects grow, it can be difficult to visualize both the software product and the software testing area and their dependencies. While direct tracing of requirements to tests was not shown in the evaluation, its feasibility is apparent via inclusion of test associations in ReqIF SpecObject attributes using a test-based TraceOriginType and associated attributes, analogous to the ReqIF scenario.

VI. Conclusion

As software size and its quality expectations grow, its invisibility and complexity affect software comprehension for V&V stakeholders. VR-V&V contributes an immersive solution concept for supporting bidirectional traceability of requirements to design elements and analysis of software testing dependencies and coverage. The prototype realization showed the feasibility of supporting immersive bidirectional traceability as well as immersive software test coverage and analysis. The evaluation results based on a case study demonstrated its capabilities, in particular traceability support involving ReqIF, ArchiMate models, test results, test coverage, and test source to test target dependency tracing. Performing analysis tasks in VR provides a unique immersive experience that can enhance and make visible often "invisible" traces between various digital artifacts, while providing a potential motivational aspect to V&V tasks in general.

Future work includes: integration with VR-SysML, VR-UML, and VR-Git with expanded support for traceability across of all lifecycle artifacts; support for V&V collaboration and annotations; and conducting a comprehensive empirical study. An automated approach for detecting and associating test and code artifacts with requirements is described in our prior work VR-SysML+Traceability. We thus intend to explore automated code-level traceability support and integration of VR-Git to support commit traceability.

Acknowledgment

The author would like to thank Jakob Loskan and Lukas Tobias Westhäußer and for their assistance with the design, implementation, figures, and evaluation.

References

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

[2] IEEE, "IEEE Standard for Software Quality Assurance Processes," IEEE Std 730-2014, IEEE, 2014.

[3] IEEE, "Systems and software engineering—Vocabulary," ISO/IEC/IEEE 24765:2010, IEEE, 2010.

[4] IEEE, "IEEE Standard for System, Software, and Hardware Verification and Validation," IEEE Std 1012-2016, IEEE, 2017.

[5] Los Alamos National Laboratory, "Form 3056 - Software Requirements Traceability Matrix (SWTM)," 2018. [Online]. Available from: https://engstandards.lanl.gov/esm/software/Form-3056.docx 2023.11.13

[6] Los Alamos National Laboratory, "Source Tracker Software Requirements Traceability Matrix," 2016. [Online]. Available from: https://engstandards.lanl.gov/esm/software/Form-3056.docx 2023.11.13

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

[8] 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.

[9] 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.

[10] 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.

[11] OMG, "Unified modeling language version 2.5.1," Object Management Group, 2019

[12] OMG, "OMG Systems Modeling Language Version 1.6," Object Management Group, 2019.

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

[14] Open Group, "ArchiMate 3.2 Specification," The Open Group, 2022.

[15] R. Oberhauser, "VR-SysML+Traceability: Immersive Requirements Traceability and Test Traceability with SysML to Support Verification and Validation in Virtual Reality," International Journal on Advances in Software, Volume 16, Numbers 1 & 2, 2023, pp. 23-35. ISSN: 1942-2679

[16] OMG, "Requirements Interchange Format (ReqIF) Version 1.2", OMG, 2016

[17] Y. Li and W. Maalej, "Which Traceability Visualization Is Suitable in This Context? A Comparative Study," In: Regnell, B., Damian, D. (eds) Requirements Engineering: Foundation for Software Quality (REFSQ 2012), LNCS, vol 7195. Springer, Berlin, Heidelberg, 2012. https://doi.org/10.1007/978-3-642-28714-5_17

[18] Z.S.H. Abad, M. Noaeen, and G. Ruhe, "Requirements Engineering Visualization: A Systematic Literature Review," 2016 IEEE 24th International Requirements Engineering Conference (RE), Beijing, China, 2016, pp. 6-15, doi: 10.1109/RE.2016.61.

[19] A.A. Madaki and W.M.N.W. Zainon, "A Review on Tools and Techniques for Visualizing Software Requirement Traceability," In: Mahyuddin, N.M., Mat Noor, N.R., Mat Sakim, H.A. (eds) Proceedings of the 11th International Conference on Robotics, Vision, Signal Processing and Power Applications, Lecture Notes in Electrical Engineering, vol 829, Springer, Singapore, 2022. https://doi.org/10.1007/978-981-16-8129-5_7.

[20] IBM Engineering Requirements Management DOORS Next. [Online]. Available from: https://www.ibm.com/products/requirements-management-doors-next 2023.11.13

[21] Perforce Helix ALM. [Online]. Available from: https://www.perforce.com/products/helix-alm 2023.11.13

[22] Sparx Systems Enterprise Architect. [Online]. Available from: https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_navigation/elementrelationshipmatrix.html 2023.11.13

[23] J. Vincur, P. Navrat, and I. Polasek, "VR City: Software analysis in virtual reality environment," In 2017 IEEE international conference on software quality, reliability and security companion (QRS-C), IEEE, 2017, pp. 509-516.

[24] K. Dreef, V.K. Palepu, and J.A. Jones, "Global Overviews of Granular Test Coverage with Matrix Visualizations," 2021 Working Conference on Software Visualization (VISSOFT), 2021, pp. 44-54, doi: 10.1109/VISSOFT52517.2021.00014

[25] A. Rahmani, J.L. Min, and A. Maspupah, "An evaluation of code coverage adequacy in automatic testing using control flow graph visualization," In 2020 IEEE 10th Symposium on Computer Applications & Industrial Electronics (ISCAIE), IEEE, 2020, pp. 239-244.

[26] F. Pecorelli, G. Di Lillo, F. Palomba, and A. De Lucia, "VITRuM: A plug-in for the visualization of test-related metrics," Proc. Int'l Conf. on Adv. Visual Interfaces (AVI '20), ACM, 2020, pp. 1-3, doi: 10.1145/3399715.3399954

[27] K. Alemerien and K. Magel, "Examining the effectiveness of testing coverage tools: An empirical study," Int'l J of Software Engineering and its Applications, 8(5), 2014, pp.139-162.

[28] K. Sakamoto, K. Shimojo, R. Takasawa, H. Washizaki, and Y. Fukazawa, "OCCF: A Framework for Developing Test Coverage Measurement Tools Supporting Multiple Programming Languages," 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013, pp. 422-430, doi: 10.1109/ICST.2013.59

[29] 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.

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

[31] 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

[32] 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. doi: 10.1007/978-3-030-52306-0_14

[33] 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. LNBIP, vol 453, pp. 122-140. Springer, Cham. doi:10.1007/978-3-031-11510-3_8

[34] 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: Shishkov, B. (eds) Business Modeling and Software Design. BMSD 2023. LNBIP, vol 483. Springer, Cham, 2023. https://doi.org/10.1007/978-3-031-36757-1_7

[35] ISO/IEC, "Software Engineering — Guide to the software engineering body of knowledge (SWEBOK)," ISO/IEC TR 19759:2015, 2015.

[36] ISO/IEC/IEEE, "International Standard - Software and systems engineering--Software testing--Part 4: Test techniques," ISO/IEC/IEEE 29119-4:2015, 2015, doi: 10.1109/IEEESTD.2015.7346375

[37] L. Inozemtseva and R. Holmes, "Coverage is not strongly correlated with test suite effectiveness," Proc. 36th Int'l Conf. on Software Eng. (ICSE 2014), ACM, 2014, pp. 435-445.

[38] M. Ivanković, G. Petrović, R. Just, and G. Fraser, "Code coverage at Google," Proc. 2019 27th ACM Joint Meeting on European Software Engineering Conf. and Symposium on the Foundations of Software Eng., ACM, 2019, pp. 955-963.

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

[40] Evans Data Corporation. [Online]. Available from: https://evansdata.com/press/viewRelease.php?pressID=293 2023.11.13

[41] The Open Group, "Agile Architecture Modeling Using the ArchiMate Language," Guide (G20E), The Open Group, 2020.

[42] The Open Group, "Using the ArchiMate® Language with UML," White Paper (W134), The Open Group, 2013.

[43] ISO/IEC/IEEE, "ISO/IEC/IEEE International Standard - Systems and software engineering -- Life cycle processes -- Requirements engineering," in ISO/IEC/IEEE 29148:2018(E), 2018, doi: 10.1109/IEEESTD.2018.8559686.

[44] The Open Group, "ArchiMate Model Exchange File Format for the ArchiMate Modeling Language, Version 3.0," U191, 2019.

[45] ReqIFSharp. [Online]. Available from: https://reqifsharp.org 2023.11.13

[46] ReqIF Studio. [Online]. Available from: https://www.reqif.academy/software/reqif-studio/ 2023.11.13

[47] The Open Group: ArchiSurance Case Study, Version 2. The Open Group (2017a).

[48] Archi. [Online]. Available from: https://www.archimatetool.com 2023.11.13

[49] 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

[50] Geocoding.net [Online]. Available from: https://github.com/chadly/Geocoding.net 2023.11.13