Towards Immersive Software Archaeology: Regaining Legacy Systems' Design Knowledge via Interactive Exploration in Virtual Reality

Adrian Hoff
adho@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Michael Nieke
micni@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

Christoph Seidl
chse@itu.dk
IT University of Copenhagen
Copenhagen, Denmark

ABSTRACT

Many of today's software systems will become the legacy systems of tomorrow, comprised of outdated technology and inaccurate design documents. Preparing for their eventual re-engineering requires engineers to regain lost design knowledge and discover re-engineering opportunities. While tools and visualizations exist, comprehending an unfamiliar code base remains challenging. Hence, software archaeology suffers from a considerable entry barrier as it requires expert knowledge, significant diligence, tenacity, and stamina. In this paper, we propose a paradigm shift in how legacy systems' design knowledge can be regained by presenting our vision for an immersive explorable software visualization in virtual reality (VR). We propose innovative concepts leveraging benefits of VR for a) immersion in an exoteric visualization metaphor, b) effective navigation and orientation, c) guiding exploration, and d) maintaining a link to the implementation. By enabling immersive and playful legacy system exploration, we strive for lowering the entry barrier, fostering long-term engagement, strengthening mental-model building, and improving knowledge retention in an effort to ease coping with the increased number of tomorrow's legacy systems.

CCS CONCEPTS

• **Software and its engineering → Software reverse engineering**; • **Human-centered computing → Visualization**.

KEYWORDS

Legacy Software, Software Engineering, Software Re-Engineering, Software Visualization, Software Archaeology, Virtual Reality

ACM Reference Format:
Adrian Hoff, Michael Nieke, and Christoph Seidl. 2021. Towards Immersive Software Archaeology: Regaining Legacy Systems' Design Knowledge via Interactive Exploration in Virtual Reality. In *Proceedings of the 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '21), August 23–28, 2021, Athens, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3468264.3473128

1 INTRODUCTION

Many of today's large-scale productive systems will become tomorrow's legacy systems when companies do not adapt to advances in technology, design knowledge is rendered inaccurate over time, or original developers move on to other projects/companies [4]. While the ongoing use of a legacy system indicates a continued value for its company, coping with new requirements (e.g., to accommodate for increased load or changed legislation) may ultimately require substantial re-engineering. To scope that re-engineering, it is then necessary to regain design knowledge on a legacy system's structure and behavior [8, 12], and to identify re-engineering opportunities, i.e., a potential for quality improvement. For many legacy systems, the only reliable source for regaining design knowledge commonly consists of the system's implementation and its runtime behavior exposed in current use [4, 12]. While tools for analyzing and visualizing a software system's design exist [2, 13], exploring an unfamiliar codebase still requires significant effort, motivation, and diligence, which, at present, makes regaining legacy systems' design knowledge a challenging and tedious procedure.

In this paper, we propose a paradigm shift in how legacy systems' design knowledge can be regained by presenting a vision for an immersive and interactive software visualization in virtual reality (VR) generated from a system's implementation, execution traces, and quality metrics that allows for engaging and playful legacy system exploration. In particular, we aim to capitalize on the benefits of VR by striving to achieve the following objectives:

- Lowering the entry barrier to exploring legacy systems
- Fostering long-term engagement via a drive for exploration
- Strengthening users' mental model of a system's design, behavior, and quality
- Improving knowledge retention even over periods of absence

In the following, we present our vision and future core contributions for what we call *immersive software archaeology*.

2 STATE OF THE ART

Three principal approaches exist for regaining a software system's design knowledge, i.e., (i) manually browsing through code and models, (ii) guided by knowledgeable programmers who know the subject system, and (iii) computer-aided techniques [8]. As browsing through a large-scale system strictly manually is not feasible and experienced programmers are often not available, computer-aided techniques in form of (semi) automated tools are the only realistically applicable approach. Generally, these fulfill two consecutive tasks. First, they analyze certain aspects of a subject system, e.g., subsystem decomposition, dependency analysis, or metrics calculation [8]. Second, they display respective results as text (tables, lists) or visual 2D/3D representations.

Depending on the purpose of a tool, visual representations can significantly improve the legibility of its output by addressing users' visual cognitive capabilities [1]. Software visualization aids software comprehension by leveraging this benefit and displaying software systems in form of visual representations in 2D (e.g., graphs, diagrams) or 3D space. Doing the latter requires a metaphor that maps the intangible concepts of software (e.g., classes) to visually meaningful objects. These can be subdivided into abstract (e.g., graph or tree-like) and real-world (e.g., cities, islands, solar systems) metaphors [13]. With recent advances in VR, research gained an increased interest in leveraging the technology to improve software comprehension. Studies suggest that VR can help with software comprehension tasks [14] and knowledge retention [6].

We identify shortcomings of existing 3D software visualization methods that we aim to address: A majority of methods rigidly visualize all artifacts of a system at once [2, 13]. This can result in overwhelmingly complex 3D structures (e.g., large 3D cities) and, thus, impaired legibility and comprehensibility. Additionally, a majority of existing concepts use 3D metaphors solely to represent selected fundamental metrics of a system's artifacts (e.g., lines of code for the size of a 3D structure) where an interpretation in terms of quality requires significant software engineering expertise. That constitutes an entry barrier, which is raised further by overloaded visual metaphors and missing explanations. Regarding behavior, existing execution trace visualization approaches focus either on scalability with large systems by providing lucid, yet limited, information or on providing detailed information, which does not scale with large systems. Generally, we observe that most existing methods are constructed to ease the comprehension of only one aspect of a system, i.e., either structure, behavior, or quality. At the same time, most existing tools serve as additional exploration means that depend on simultaneously browsing source code in an IDE. Regarding the potential benefits of immersive capacities offered by modern VR technology, we conclude that current VR methods have yet to overcome the stage of essentially porting existing standard-screen 3D visualization methods into a VR environment.

3 IMMERSIVE SOFTWARE ARCHAEOLOGY

To provide a framework for our envisioned contributions, we briefly elaborate on the model-driven process for creating core artifacts of our method as depicted in Figure 1: Our method will analyze ① a legacy system's implementation for architectural recovery yielding information on hierarchically nested sub-systems, components, etc. and their interrelations, e.g., as mentioned in [3]. Our method will collect behavior information by recording and analyzing ② method-call traces passing through that structure. Our method will further collect quality information ③ by analyzing the models yielded from ① and ② with regards to technical debt on an architectural level, e.g., by conducting dependency analyses and identifying execution bottlenecks. Using a visual metaphor, our method will synthesize ④ these results into a visualization model, i.e., a data model that contains abstract information on the spatial arrangement and appearance of visual elements that meaningfully represent a subject system. Based on that model, we will then assemble a virtual 3D environment ⑤ to use within our VR visualization.

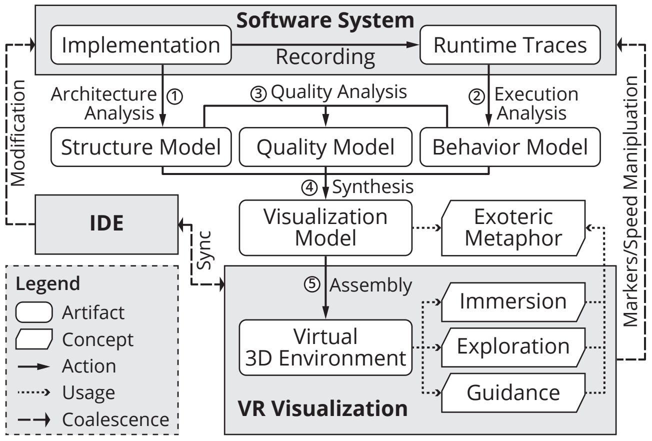

Figure 1: Overview of our envisioned method. An automated process analyzes a subject system and yields a 3D environment that is explored in VR. The visualization is synchronized with an IDE.

The models of this process will allow for the representation of object-oriented languages by successively abstracting from specific target programming language features. We will demonstrate our method and its information retrieval mechanisms on the case of legacy systems implemented in Java, e.g., by instrumenting the Java Virtual Machine to transparently record execution traces. However, we design our concepts generalizable for imperative languages that allow deriving data on encapsulation, modularity, and coupling.

To achieve our vision of *immersive software archaeology*, we will draw on existing work on software re-engineering [8, 12], software comprehension [9], software architecture recovery [3, 5], and software visualization [2, 13]. We will devise innovative concepts that immerse re-engineers into a VR representation of a subject system (Section 3.1), provide means for effective navigation and orientation (Section 3.2), guide users through understanding both our method and the subject system (Section 3.3), while maintaining a link to the actual source code (Section 3.4). The synthesis provided by these concepts will constitute an integrated, interactive, and immersive visualization of a software system's structure, behavior, and quality.

3.1 Immersion: Experiencing the System

Recent research suggests that examining a software system is perceived as more satisfactory when using a 3D visualization instead of a text-based IDE [10]. We will extend that idea by exploiting the immersive properties of VR to foster a drive for exploration.

Exoteric Metaphor with Esoteric Properties. We will support users' mental-model building by relating to real-world knowledge. Therefore, we will design an exoteric (common knowledge) metaphor to represent a software system's structure, behavior, and quality on various levels of granularity as elements from the physical world. For structural aspects, a component could be visualized as a planet in a solar system with its sub-components as cities on that planet and classes as buildings populating the respective city, where dimensions and placement indicate artifact size and logical relation.

For instance, class buildings could be composed of sub-complexes for each method, shaped according to method properties such as the number of splits in their control flow. For behavioral aspects, the control flow of recorded execution traces could be visualized as abstract inhabitants traveling through the representation along the involved software components. For qualitative aspects, metric values could be conveyed as esoteric properties of our representation (common elements with specialized meaning), e.g., communicating a high degree of coupling via visually complex textures or the presence of architectural smells via litter in the environment.

Unlike with the majority of existing software visualizations, users will not only take the role of an overseeing observer but will also explore the represented structures in first-person view. When roaming freely, users may discover detailed phenomena, e.g., congregations of numerous "inhabitants" indicating heavily frequented software components. When guided along specific paths (cf. Section 3.3), users can explore common system usage or manually created routes deemed particularly relevant.

Ambient Visuals and Acoustics. To subliminally communicate behavior and quality information, our method will incorporate ambient effects based on textures, animations, and sound: We will use ambient visuals to highlight particularly frequented routes through the system (e.g., by worn-out paths), which creates a focal point for exploration. When replayed, execution traces may emit 3D positional sounds to indicate location, frequency, and clusters of events. We will also explore ambient music to convey a subliminal impression of software quality for a visited part of the software system, e.g., by using tense music to accentuate bad software quality.

Diegetic Interfaces. We will offer various tools to inspect and interact with the represented software system (cf. Section 3.2 and Section 3.4). Instead of breaking with immersion by a mere overlay interface (e.g., screen-space 2D buttons and menus), we will design access to all context-dependent tools as diegetic interfaces that integrate with the metaphor and the 3D representation [11]. Using the above metaphor, users could be provided with an overview of their current location within a subject system via a holographic projection originating from a device attached to their virtual arm. To compare software elements (e.g., classes) with each other, users will be able to temporarily detach them from their actual environment and transport them to a dedicated comparison environment. Furthermore, users will be able to inspect a system's live behavior by interacting with it directly, e.g., by accessing the public interface of a class to input data and observing how the system reacts.

We will counteract stimulus overflow by offering users to configure what information they are interested in and filtering the VR environment accordingly. For instance, users that are solely interested in behavior can decide to filter out information on quality. We will create predefined profiles for different exploration purposes.

3.2 Exploration: Navigation and Orientation

Exploiting VR's capability of immersing users into a virtual environment requires providing means for effective and efficient navigation. For that purpose, our method will feature navigation along structure, behavior, and quality.

Structural Semantic Zoom. A Global overview is considered crucial for 3D information visualization [2]. As a starting point, our method will show a subject system on its architectural level. Structure and quality will be visualized on component level, between which recorded traces visualize behavior. As part of our visual metaphor, we envision a step-wise semantic zoom that enables navigating along a system's structure to reveal further details. For instance, zooming into a component will visualize its inner structure, behavior, and quality, i.e., of contained sub-components or classes. A key aspect here is to maintain users' orientation, e.g., with an interface showing the current position in the overall system.

As an example, using the metaphor given in Section 3.1, a system could be initially visualized on the level of planets (components). Users could be able to zoom into a planet that is then enlarged (optically) and augmented with further semantic details, e.g., abstract representations of its cities (sub-components). Zooming into a city could be realized by letting users walk among its individual buildings (classes). To the best of our knowledge, our envisioned software visualization method is the first to provide such zooming/abstraction functionality based on functional architectural components integrated into a real-world metaphor.

Temporal Behavioral Zoom. Cross-cutting to the semantic structural zoom, we envision a temporal zoom that enables users to manipulate the time in which recorded execution traces are represented. Users will be able to speed up, slow down, and freeze time as well as to jump back and forth in time. Extending the idea of time travel debugging, this mechanism will leverage both inspecting short-lived phenomena and surveying lengthy procedures.

Context-Dependent Quality Zoom. We will make quality information available on demand by letting users select types of technical debt to be visualized in the currently visited part of a system. Respective information will be represented as exoteric property with esoteric meaning, e.g., a dirty floor within a building as an indicator for a code smell within a class.

3.3 Guidance: Fostering Understanding

Our method will progressively self-explain its usage via built-in tutorials. In parallel, it will adaptively guide users through a subject system, according to specific interests, e.g., in its behavior.

Understanding the Method. Unlike a majority of existing approaches, we will ease the entry into our method by generating tutorial-style quests from the actual system that incrementally introduce tools and diegetic interfaces, e.g., "locate the class x" or "find the metric value for y". We will maintain users' immersion by embedding these tutorials into our metaphor, triggering them according to what the user intends to do, and enabling to skip or entirely disable the tutorials. Additionally, if a user inspects the visual representation of a software entity (such as a class), our method will provide hints on how to "read" it, e.g., via a helping companion.

Understanding a System. We will help users with understanding a subject system more efficiently by using visual effects and audio to create observable phenomena that draw attention to potentially interesting areas. For instance, we will encode information on the quality of components and classes via the texture of their VR representations. Using the metaphor given in Section 3.1, a class with an overall high complexity could be represented by a building with a visually complex texture. To guide users concerning behavioral quality (esp. on high abstraction levels), we will offer to augment execution trace routes with visual aids that encode how highly these paths are frequented, e.g., using a heat map color scheme ranging from dark blue to bright red. Our method will explain the meaning of these phenomena to users, e.g., via a helping companion. Furthermore, we will generate quests to explore the system, e.g., to travel along the most common execution route.

3.4 Coalescence: Mental & Technical Back-Link

We will foster a link between a user's mental model formed in our visual metaphor and the underlying implementation of a subject system. Unlike existing software visualizations, we will therefore coalesce and synchronize our visualization with an IDE, e.g., by extending Microsoft's language server protocol[1] to place and query markers for system parts of interest.

Details in VR. Our method will enable inspecting implementation details directly from within the VR visualization. For instance, using the metaphor given in Section 3.1, that could be achieved by providing users with a diegetic interface (e.g., a virtual tablet computer) which, once attached to a class (building), displays its source code. Upon performing changes to artifacts within the IDE, we will update the VR visualization accordingly.

Annotations (POIs). Users will be able to annotate points of interest (POI) throughout the VR environment. These will conserve gained insights in form of user-created text messages and sound recordings, which are related to structures represented within the VR representation. Using the metaphor given in Section 3.1, POIs could be realized in form of sticky notes that users pin to the facade of a building (class) or, on a higher level of semantic zoom (cf. Section 3.2), to a planet (component). POIs will be synchronized with the IDE so that markers can be created, inspected, modified, and deleted with the respective software elements.

Record & Replay. We will devise a mechanism that allows users to record sequences of previously marked POIs so that they can use contained messages and voice recordings to construct paths through the system. Encoding experiences in that way will allow for building an asynchronous mentor-mentee relationship between collaborators and, thus, ease the exploration of a subject system. We will synchronize recorded paths with the IDE so that they can be recorded and navigated from within both VR and the IDE.

4 EVALUATION

We are in the process of refining our ideas and visions into concrete concepts and a prototype. We will use that prototype to empirically evaluate our concepts in a series of controlled experiments which, in turn, will influence further development of our concepts and prototype. As subject systems, we will use multiple real-world legacy systems, e.g., a long-running, undocumented medical supply system provided by one of our industry partners. To consider diverse backgrounds, we will recruit participants with knowledge in software engineering from various countries, already having arranged for participants from Denmark, Germany, and Switzerland. We will divide participants into a treatment group using our immersive VR method as well as two control groups using a) an existing standard-screen 3D visualization tool and b) a standard IDE along with dedicated tools for architecture recovery, quality metric calculation, etc. After a brief training period on either of the tools, we will use two consecutive experiment sessions that are interleaved with two weeks of absence from the subject system. In the first session, we will evaluate our objectives of lowering the entry barrier, maintaining long-term engagement, and easing mental model building by letting participants conduct a series of extensive tasks, e.g, finding certain system aspects or identifying performance bottlenecks. We will measure metrics such as completion time and correctness, collect participants' subjective verdict (e.g., similar to [7]), and observe their behavior. In the second session, we will evaluate participants' knowledge retention by asking questions about the system's structure, behavior, and quality, e.g., by asking them to recall its architecture from memory or to find the same system aspects as in the first session.

[1]https://langserver.org/

5 CONCLUSION AND FUTURE WORK

With our envisioned method, we aim to reshape the comprehension phase of legacy software system re-engineering. Instead of being labor-intensive, tedious work, we will immerse re-engineers in an engaging, playful exploration that builds a strong mental model of a system while still connecting to its implementation. By making software archaeology less tedious and more accessible to a wider audience, we strive for our method to ease coping with an increased number of legacy systems of the future.

ACKNOWLEDGMENTS

This work is supported by the DFF (Independent Research Fund Denmark) within the project ISA – Immersive Software Archaeology (0136-00070B).

REFERENCES

[1] M. Card. 1999. *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann.
[2] P. Caserta and O Zendra. 2011. Visualization of the Static Aspects of Software: A Survey. *IEEE Transactions on Visualization and Computer Graphics* (2011).
[3] J. Garcia, I. Ivkovic, and N. Medvidovic. 2013. A Comparative Analysis of Software Architecture Recovery Techniques. In *Int. Conf. on Automated Software Engineering (ASE).*
[4] R. Khadka, B. V. Batlajery, A. M. Saeidi, S. Jansen, and J. Hage. [n.d.]. How Do Professionals Perceive Legacy Systems and Software Modernization?. In *Int. Conf. on Software Engineering (ICSE), year = 2014.*
[5] T. Lutellier, D. Chollak, J. Garcia, L. Tan, D. Rayside, N. Medvidovic, and R. Kroeger. 2015. Comparing Software Architecture Recovery Techniques Using Accurate Dependencies. In *37th IEEE Int. Conf. on Software Engineering.*
[6] L. Merino, J. Fuchs, M. Blumenschein, C. Anslow, M. Ghafari, O. Nierstrasz, M. Behrisch, and D. A. Keim. 2017. On the Impact of the Medium in the Effectiveness of 3D Software Visualizations. In *Work. Conf. on Software Visualization (VISSOFT).*
[7] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz. 2017. CityVR: Gameful Software Visualization. In *2017 IEEE Int. Conf. on Software Maintenance and Evolution (ICSME).*
[8] H. A. Müller, J. H. Jahnke, D. B. Smith, M.-A. Storey, S. R. Tilley, and K. Wong. 2000. Reverse Engineering: A Roadmap. In *Conf. on the Future of Software Engineering.*
[9] M. J. Pacione, M. Roper, and M. Wood. 2004. A Novel Software Visualisation Model to Support Software Comprehension. In *Working Conf. on Reverse Engineering.*
[10] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza. 2019. On the use of Virtual Reality in Software Visualization: The Case of the City Metaphor. *Information and Software Technology* (2019).
[11] P. Salomoni, C. Prandi, M. Roccetti, L. Casanova, and L. Marchetti. 2016. Assessing the Efficacy of a Diegetic Game Interface with Oculus Rift. In *Annual Consumer Communications & Networking Conference (CCNC).*
[12] H. Sneed and C. Verhoef. 2019. Re-implementing a Legacy System. *Journal of Systems and Software* (2019).
[13] A.R. Teyseyre and M.R. Campo. 2009. An Overview of 3D Software Visualization. *Transactions on Visualization and Computer Graphics* (2009).
[14] R. Wettel, M. Lanza, and R. Robbes. 2011. Software Systems as Cities: A Controlled Experiment. In *Int. Conf. on Software Engineering (ICSE).*