XRaSE: Towards Virtually Tangible Software using Augmented Reality

XRaSE is an approach that leverages augmented reality to automatically create an immersive 3D representation of input software code.

Current Problems

Existing virtual reality (VR)-based approaches come with several significant drawbacks. They restrict collaboration and comprehension because of a limited field-of-view, require disruptive context switching, and disconnect users from the real world, leading to less natural interaction environments.

XRaSE addresses these issues by using AR to create immersive 3D representations of software. This method enhances comprehension and collaboration without severing ties to the real world, as AR augments the field-of-view while maintaining a connection to the physical environment.

The prototype of XRaSE allows users to interact with software as if it were a tangible object, significantly improving understanding and communication.

The primary hypothesis behind XRaSE is that it will make activities such as software comprehension, architecture analysis, knowledge communication, and dynamic analysis more intuitive, richer, and collaborative.

The key advantage of using AR in XRaSE is that it offers a real-world context and natural interactions. However, this approach may introduce some initial complexity in interaction and comprehension for the users.

Technology

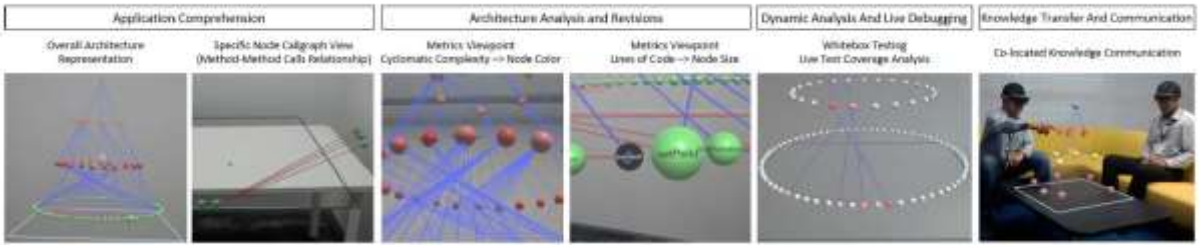
The implementation of XRaSE utilizes Microsoft HoloLens, Unity, the Mixed Reality Toolkit, Neo4j, and Spring Boot to create its immersive AR environment.

Evaluation Design

The evaluation of XRaSE involves comparing a 2D medium with traditional 2D visualization and a 3D AR environment using HoloLens. Five groups were focused on different aspects: structure,

metrics, dependency, code, and debugging. The evaluation process included a pre-study questionnaire, quantitative measurements by a moderator, and a post-study questionnaire to gather comprehensive feedback.

In summary, XRaSE offers a novel and promising approach to software visualization and comprehension through augmented reality, though it still requires further refinement based on user feedback and consistency improvements.



Toward a VR-Native Live Programming Environment

The paper introduces a VR-native editor that utilizes direct manipulation in a structured editor specifically designed for **Smalltalk** programming. This novel approach aims to enhance the programming experience within a virtual reality environment.

Previous Work

Earlier attempts to integrate programming with VR primarily involved **projecting** traditional 2D editors into a 3D VR space, often using text to represent code. These methods did not fully capitalize on the potential of VR environments.

One of the main challenges identified is the ergonomic and hardware limitations that affect extended programming sessions in VR. These issues can impact user comfort and efficiency.

The study demonstrates that with prior instructions, users can effectively utilize the VR-native editor for implementing small features. This finding suggests that, with proper guidance, the VR environment can be helpful with programming tasks.

The paper hypothesizes that minimizing text entry and emphasizing direct manipulation within the VR environment could enhance productivity for VR programming.

Results

The results showed a significant decrease in the number of failed text input attempts. Participants were faster when using stroke letters, although some expressed a preference for touch typing or speech-to-text methods. The code browsing interface was well-received, and the live execution feature was described as intuitive and easy to use.

Pros and Cons

The advantages of this VR-native programming environment include **immediate feedback** and effective utilization of 3D space.

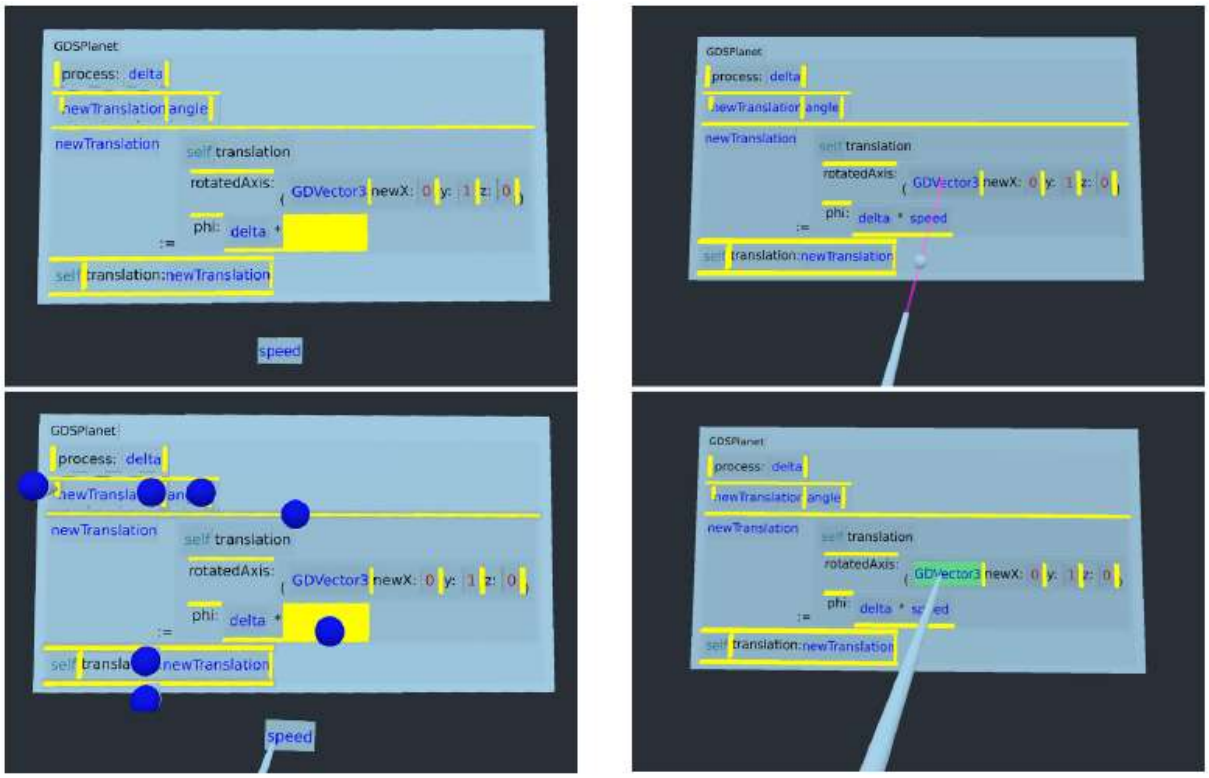
However, the approach also has its drawbacks, such as **lower productivity** compared to traditional desktop environments and the **disconnection from the real world** during VR sessions.

The study highlights the need for more readable code representation and improved text input methods as areas requiring further research and development.

Evaluation

The evaluation process involved participants with no prior instructions initially, followed by a quick tutorial for similar programming tasks. This method allowed the researchers to assess the effectiveness of the VR-native editor in facilitating programming activities.

In conclusion, this VR-native live programming environment shows promise in leveraging VR's capabilities to enhance programming experiences, though it still faces challenges related to ergonomics, text input, and overall productivity compared to traditional environments.



VR-Based User Interactions to Exploit Infinite Space in Programming Activities

The article discusses **VRIDE**, a virtual reality based programming environment designed to leverage the infinite space available in VR for software development.

Previous Work

Traditional VR environments for programmers have typically projected classical integrated development environment (IDE) windows into the VR space. This approach limits the potential benefits of VR's infinite spatial capabilities, as it does not fully utilize the **three-dimensional nature** of the medium.

Current State of the Art

This paper makes significant contributions to the state-of-the-art by identifying an overlooked opportunity in classical VR-based environments and proposing the concept of the **"code cube."** The code cube is a visual metaphor designed to help programmers exploit the infinite space offered by VR more effectively.

The article says that VRIDE could enhance software development processes by providing a spatially unlimited environment, thus allowing programmers to organize and manage code more efficiently than on traditional screens.

Capabalities of VRIDE

Code cube is a visual metaphor to represent a class, which is the primarily structural element in object-oriented programming. A class is visually rendered as a cube, in which each facet represents a particular aspect like:

Class definition, List of methods, Method source code, Scatterplot, Incoming dependencies, Outgoing dependencies, Superclass and subclasses, Log Entry

Code cube unfolding – unfold the cube to show all facets at the same time. Focus on particular class.

Spawning code cube – spawn any number of code cubes in VR environment.

Removing code cube – remove unnecesary code cubes.

Playground Cube – user can execute scripts in here. It supports editing end executing scripts.

Users have a virtual keyboard avilable for textual input or a real one.

Pros:

Full and permanent immersion, designed to fully develop an application in the virtual environment. It exploits the infinite virtual space and offers immmersive software interaction through cube metaphor.

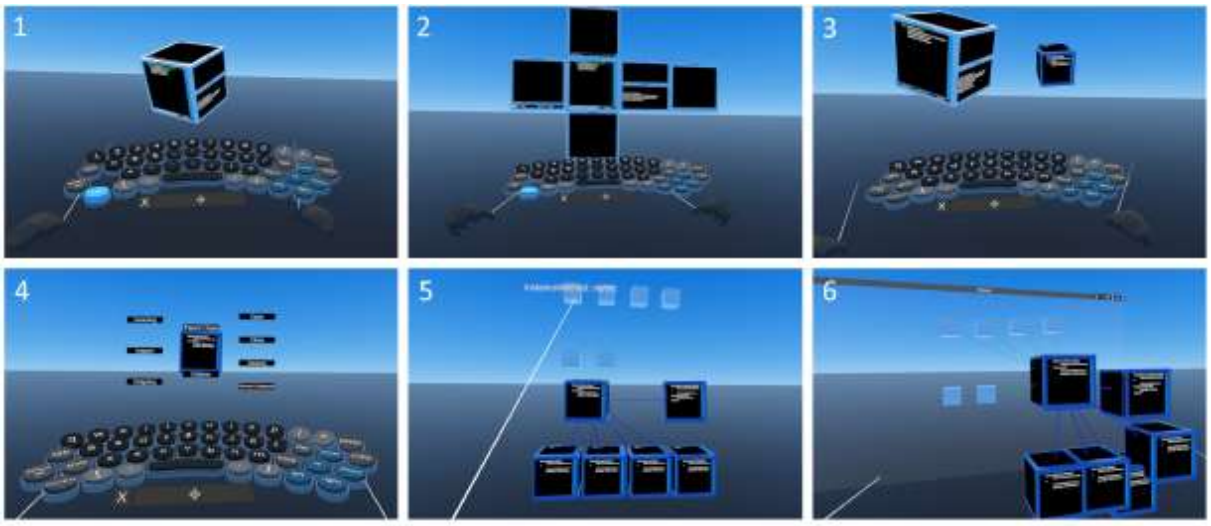
Cons:

Textual input is not as efficient as with a real keyboard.

Real world is cut off.

Results

VRIDE enables programmers to effectively navigate and manipulate an expansive virtual space. This capability suggests that VRIDE could significantly improve programming tasks by overcoming the spatial limitations inherent to physical screens.

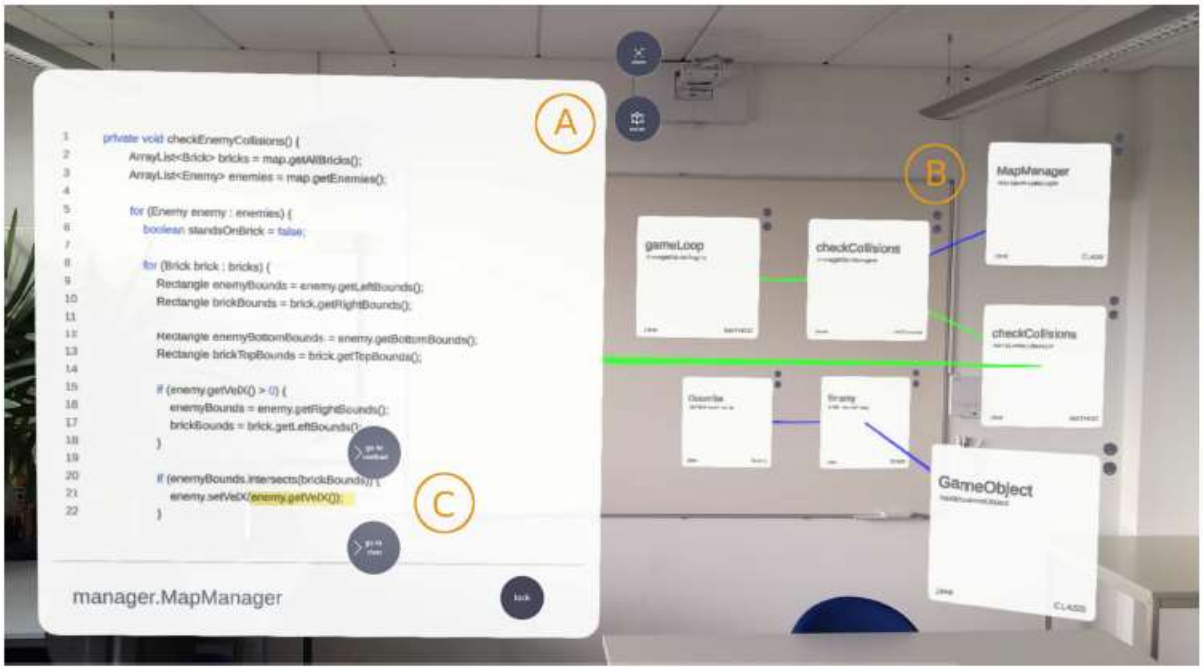


IDEVELOPAR: Enhancing Code Understanding in Augmented Reality

IDEVELOPAR is a tool designed to improve code comprehension using augmented reality. It consists of an AR application on the HoloLens 2 and an (IDE) running on a regular computer.

Previous Work

The development of IDEVELOPAR builds on previous tools like Code Bubbles, a 2D extension for IDEs, which suffer from **screen size limitations**, and Code Canvas, which provides an infinite canvas for visualizations and graphs in 2D. These tools have paved the way for exploring new methods of code visualization and interaction.



The article demonstrates that developers using IDEVELOPAR achieve **better code understanding** and **potentially higher productivity**. It presents evidence that AR can significantly enhance the visualization of code.

The authors hypothesize that a significant learning effect occurs with prolonged use of IDEVELOPAR, which could lead to increased effectiveness and productivity over time.

Architecture:

The tool consists of two parts, the actual augmented reality application running on the HoloLens 2 and any IDE out of the JetBrains family running on a regular computer.

Linked Code Panels in AR

The code panel is the core element of the tool, which visually represents a code fragment of the project using augmented reality. Various buttons are available in the upper right area of the panel. These are used resize the panel or to close the panel or the complete sub-tree with the panel as its root.

In a code panel a user can click at a class name, a method or a constructor call. As a result a new code panel with the related code fragment will be opened and a visual link is drawn to connect it to the newly opened code panel.

Code Navigation

Basic functionalities are controlled via the hand menu.

The hand menu has three options: Switching the programming mode on and off, compiling and running the program in the desktop IDE, and opening the project overview.

Users can open a new code panle via clicking on a method in an open code panel or via hand menu.

When a new code panels are created by navigation, an automatic layout places the new panel right of its parent below already existing siblings. Open code panels create a tree structure.

Programming

The source code can be changed directly in each code panel when the programming mode is activated.

Text input is provided via a bluetooth keyboard and full code completion is provided.

Changes are synced to the connected IDE if the programming mode is activated

Pros:

- Enhanced code visualization through AR, which helps in understanding complex code structures.

- Potential productivity gains as developers can interact with and visualize code in a more intuitive way.

Cons:

- Hardware limitations, such as the need for AR-capable devices like the HoloLens.

- Gesture-based interactions on the HoloLens may have a learning curve and can be less intuitive for some users.

The study suggests that the user interface needs further improvements based on user feedback and study results to enhance the overall experience and usability.

Evaluation

The evaluation of IDEVELOPAR was conducted using the cognitive dimension framework, assessing 14 different cognitive dimensions to analyze the usability of visual programming environments.

TABLE I
LIST OF COGNITIVE DIMENSIONS BASED ON [9]

Dimension	Description
Abstraction [CD1]	types and availability of abstraction mechanisms
Closeness of mapping [CD2]	closeness of representation to domain
Consistency [CD3]	similar semantics are expressed in similar syntactic forms
Diffuseness [CD4]	verbosity of language
Error-proneness [CD5]	notation invites mistakes
Hard mental operations [CD6]	high demand on cognitive resources
Hidden dependencies [CD7]	important links between entities are not visible
Premature commitment [CD8]	constraints on the order of doing things
Progressive evaluation [CD9]	work-to-date can be checked at any time
Role-expressiveness [CD10]	the purpose of a component is readily inferred
Secondary notation [CD11]	extra information in means other than formal syntax
Viscosity [CD12]	resistance to change
Visibility [CD13]	ability to view components easily
Provisionality [CD14]	degree of commitment to actions or marks

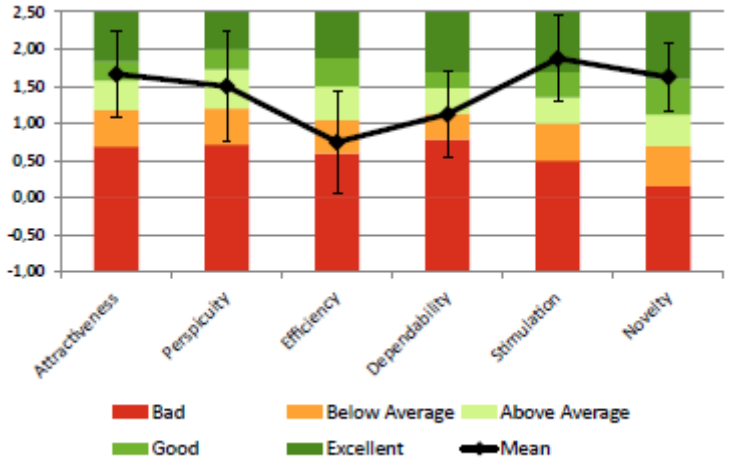
Quantitative Analysis:

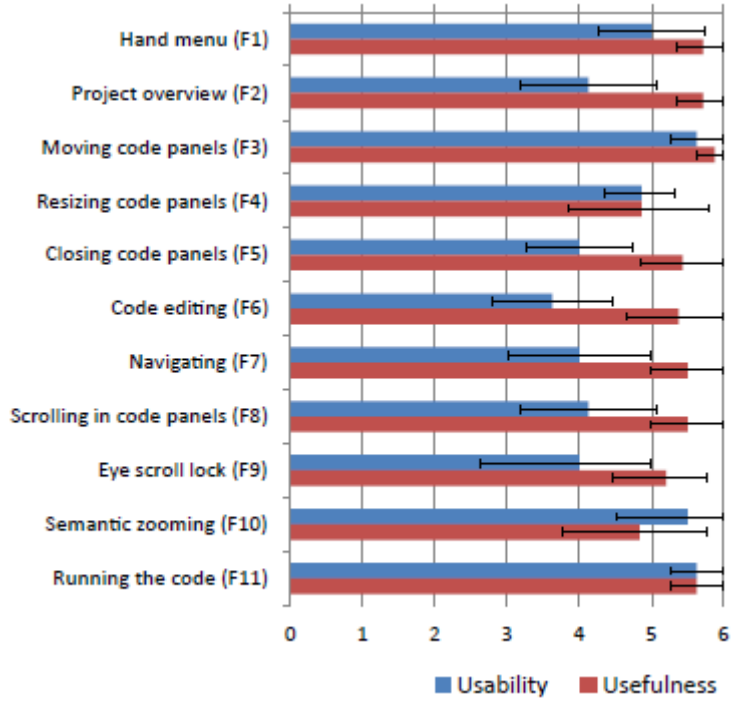
- Overall usability of the prototype was rated as acceptable (in the OK-to-good range).

- The feature "resizing a code panel and semantic zooming" received slightly below average ratings.

- Generally, the usability feedback was primarily positive, and it is expected that usability ratings may improve with extended use.

- The tool was rated above average in most categories except for efficiency.





Qualitative Analysis:

- Recorded interviews were transcribed for in-depth analysis.

- Several features had no reported problems, and many users found the AR-enhanced visualizations significantly easier to understand than traditional computer screens.

- An observable learnability effect was noted, particularly within the last group of participants, supporting the hypothesis that prolonged use enhances effectiveness.

- Most participants developed their own placement strategies for code panels and visual elements.

Suggested Extensions

- Implementing visual cues to indicate unused classes, methods, or variables.

- Adding basic error highlighting in the code panels to improve code debugging and comprehension.