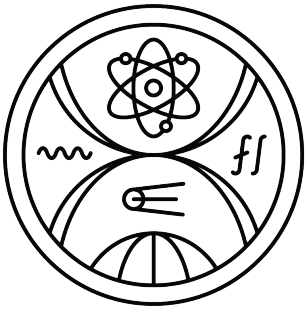
COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS

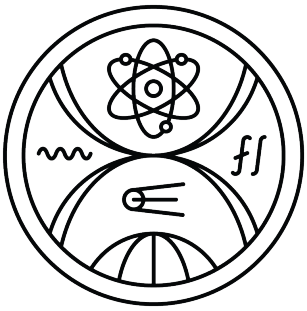


EXTENDING SOFTWARE FUNCTIONS IN VIRTUAL REALITY

Master thesis

2024 Bc. Matúš Granec

COMENIUS UNIVERSITY IN BRATISLAVA
FACULTY OF MATHEMATICS PHYSICS AND INFORMATICS



EXTENDING SOFTWARE FUNCTIONS IN VIRTUAL REALITY

Master thesis

Study program:	Applied informatics
Branch of study:	Applied informatics
Department:	Department of Applied Informatics
Supervisor:	doc. Ing. Ivan Polášek, PhD.
Consultant:	Ing. Juraj Vincúr, PhD.

Bratislava, 2024 Bc. Matúš Granec

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

THESIS ASSIGNMENT

Name and Surname: Bc. Matúš Granec
Study programme: Applied Computer Science (Single degree study, master II. deg., full time form)
Field of Study: Computer Science
Type of Thesis: Diploma Thesis
Language of Thesis: English
Secondary language: Slovak

Title: Extending software functions in virtual reality

Annotation: Programmers most often work with source code within the development framework using a keyboard and mouse,
and this code is displayed to them through a small number of 2D monitors.
Such traditional approach ignores the opportunities afforded by the newly virtual reality display technologies
such as Oculus Quest or augmented reality such as XREAL Light.
We believe that appropriate migration of the individual functionalities of the development environment
into the world of virtual or enriched reality will increase the productivity of programmers, improve their user experience
and reduce the time required for familiarization of a programmer with an unknown software system.

Aim: Analyse existing software development tools and environments. In particular, focus on 3D solutions.
Analyse already identified problems, shortcomings and suggestions for improvements to existing approaches.
Design and implement your own approach applying the technologies of virtual or augmented reality
in the context of software development.
Suggest body extension of the methods based on speech recognition, AI or VR-native Live Programming.
Evaluate the impact of your solution compared to traditional approaches.

Literature: Leonard Geier, Clemens Tiedt, et al.: Toward a VR-Native Live Programming Environment,
PAINT 2022: Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments, 2022

Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonapalawong, Tianye Zhang, Mingliang Xu and Wei Chen. 2020. A systematic literature review of modern software visualization. J. Vis., 23(4), 539–558.

Anthony Elliott, Brian Peiris, and Chris Parnin. 2015. Virtual reality in software engineering: affordances, applications, and challenges. In Proceedings of the 37th International Conference on Software Engineering

Comenius University Bratislava
Faculty of Mathematics, Physics and Informatics

- Volume 2 (ICSE '15), Vol. 2. IEEE Press, Piscataway, NJ, USA, 547-550.

F. Fittkau, A. Krause and W. Hasselbring, "Exploring software cities in virtual reality," 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), Bremen, 2015, pp. 130-134.

Supervisor:	doc. Ing. Ivan Polášek, PhD.
Consultant:	Ing. Juraj Vincúr, PhD.
Department:	FMFI.KAI - Department of Applied Informatics
Head of department:	doc. RNDr. Tatiana Jajcayová, PhD.
Assigned:	02.12.2023

Approved: 05.12.2023 prof. RNDr. Roman Ďurikovič, PhD.
Guarantor of Study Programme

.....
Student

.....
Supervisor

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Matúš Granec
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: anglický
Sekundárny jazyk: slovenský

Názov: Extending software functions in virtual reality
Rozvoj softvérových funkcií vo virtuálnej realite

Anotácia: Programátori pri vývoji softvéru najčastejšie pracujú so zdrojovým kódom v rámci vývojárskeho prostredia pomocou klávesnice a myši, a tento kód im je zobrazovaný prostredníctvom malého počtu 2D monitorov. Takýto tradičný prístup ignoruje možnosti, ktoré nám poskytujú novovzniknuté technológie
pre zobrazovanie virtuálnej reality ako napr. Oculus Quest alebo obohatenej reality ako napr. XREAL Light.
Veríme, že vhodná migrácia jednotlivých funkcionalít vývojárskeho prostredia do sveta virtuálnej alebo obohatenej reality zvýši produktivitu programátorov, zlepší ich používateľský zážitok a skráti čas potrebný na oboznámenie programátora s neznámym softvérovým systémom.

Cieľ: Analyzujte existujúce podporné nástroje a prostredia určené na vývoj softvéru. Zamerajte sa najmä na 3D riešenia. Analyzujte už identifikované problémy, nedostatky a návrhy vylepšení existujúcich prístupov.
Navrhnite a implementujte vlastný prístup, v rámci ktorého aplikujete technológie
virtuálnej alebo obohatenej reality v kontexte vývoja softvéru.
Navrhnite rozvoj tela metod na zaklade speech recognition, AI alebo VR-native Live Programming.
Vyhodnoťte prínos vášho riešenia v porovnaní s tradičnými prístupmi.

Literatúra:
Leonard Geier, Clemens Tiedt, et al.: Toward a VR-Native Live Programming Environment,
PAINT 2022: Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments, 2022

Noptanit Chotisarn, Leonel Merino, Xu Zheng, Supaporn Lonapalawong, Tianye Zhang, Mingliang Xu and Wei Chen. 2020. A systematic literature review of modern software visualization. J. Vis., 23(4), 539–558.

Anthony Elliott, Brian Peiris, and Chris Parnin. 2015. Virtual reality in software engineering: affordances, applications, and challenges. In Proceedings of the 37th International Conference on Software Engineering - Volume 2 (ICSE '15), Vol. 2. IEEE Press, Piscataway, NJ, USA, 547-550.

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

F. Fittkau, A. Krause and W. Hasselbring, "Exploring software cities in virtual reality," 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), Bremen, 2015, pp. 130-134.

Vedúci: doc. Ing. Ivan Polášek, PhD.
Konzultant: Ing. Juraj Vincúr, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. RNDr. Tatiana Jajcayová, PhD.

Dátum zadania: 02.12.2023

Dátum schválenia: 05.12.2023 prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

I hereby declare that I have written this thesis by myself, only with help of referenced literature, under the careful supervision of my thesis supervisor and consultant.

.....

Bratislava, 2024 Bc. Matúš Granec

Acknowledgement

...

Abstract

Keywords: extended reality, software development, code comprehension, code navigation, virtual IDE

Abstrakt

...

Kľúčové slová: rozšírená realita, vývoj softvéru, porozumenie kódu, navigácia v kóde, virtuálne IDE

Contents

List of Figures

List of Tables

Terminology

Terms

- **Extended reality**
  umbrella term to refer to augmented reality, virtual reality, and mixed reality (MR).

Abbreviations

- **IDE** - Integrated Development Environment.
- **XR** - Extended reality.
- **VR** - Virtual reality.
- **AR** - Augmented reality.
- **MR** - Mixed reality.

Motivation

...

Chapter 1

Introduction

Chapter 2

Background and State of the Art

2.1 Introduction to VR/MR Programming Environments

Virtual and mixed reality (VR/MR) environments have the potential to transform programming by providing immersive, spatially enhanced interaction methods. These technologies promise improvements in productivity, comprehension of complex systems, and collaboration among developers. [7]

2.2 Overview of General Challenges in VR/MR Environments

The main problems of VR or MR solutions are separation from the physical world, input and interaction challenges, ergonomic and hardware limitations, and physical strain and distortion. Other issues that should be mentioned are the learning curve and initial complexity, technological limitations, and lower productivity compared to desktops.

2.3 Types of VR and MR Devices

VR and MR devices can be broadly classified into two main categories: standalone and tethered.

Standalone VR devices contain all the computing hardware components required to work in VR without the need to connect additional external hardware.

Tethered devices require a separate computer to work. There is also a hybrid type of devices like Meta Quest 3 or Apple Vision Pro supporting both tethered and standalone operating modes.[2]

In addition to these main categories, there is a subcategory of VR devices known as mobile VR headsets. These devices utilize a smartphone as the display and processing unit, offering a more affordable and accessible entry point into VR. However, they often lack positional tracking and dedicated controllers, limiting their immersion and interaction capabilities.

2.4 Interaction and Input

VR and MR interaction methods typically involve hand tracking, controllers, or a combination of both. Hand tracking allows for more natural and intuitive interactions, enabling users to manipulate virtual objects directly with their hands. [9] Controllers provide haptic feedback and precise input, which can be crucial for tasks that require fine-grained control.

2.5 Software and Development Tools

VR and MR programming environments utilize various software and development tools. Popular game engines like Unity and Unreal Engine provide frameworks for creating VR and MR experiences. Additionally, specialized tools and libraries, such as the Mixed Reality Toolkit (MRTK) for Microsoft HoloLens and the Oculus Integration SDK, offer features and functionalities tailored to specific devices and platforms.

Chapter 3

Analysis of Existing Solutions

3.1 Introduction to the Issues

In this chapter we look at the core issues of programming in virtual and mixed reality environments. We break down the main problems that have arisen when using some of the previous approaches and describe possible solutions to the mentioned problems resulting from evaluation of previous work.

3.2 Separation from the Physical World

When using VR devices, the user is completely separated from the physical world.[8] This can lead to disorientation and motion sickness. The user is also unable to see the keyboard and mouse, which are essential for programming. Therefore, handling the input must be solved in an alternative way. Peers may lack the opportunity to ask questions, and physical communication is stifled. [7]

3.3 Input and Interaction Challenges

Textual input and interaction are one of the most prominent issues while using VR devices. Solutions such as block programming or text suggestions are viable only until you need to enter custom textual input like variable names, function names, etc.

For custom textual input, we can implement the use of a virtual keyboard or an approach like "airwrite". However, these approaches are less efficient and intuitive compared to a physical keyboard. For airwrite, when large amounts of text need to be entered, some participants would have preferred an input method that supports the speeds to which they are used [8].

There is also the possibility of leveraging voice input, but this approach is not suitable for all environments. It can be either too disturbing for others or, the environment

might be too loud for precise and efficient recognition of the voice input. The physical keyboard still remains the fastest and most efficient textual input solution.

3.4 Ergonomic and Hardware Limitations

In VR/MR settings, ergonomic issues emerge from factors like the weight of devices, discomfort experienced by users over extended periods, and hardware constraints including battery duration and display quality.

3.5 Physical Strain and Distortion

Physical strain and distortion are common issues in VR and MR environments. Prolonged use of VR headsets can cause eye strain, headaches, and fatigue. Additionally, the limited field of view and resolution of current VR displays can lead to distortion and discomfort.[1]

3.6 Learning Curve and Initial Complexity

AR and MR technologies may present a steep learning curve for users unfamiliar with the technology. The initial complexity of interacting with virtual objects and navigating through virtual environments can be overwhelming, requiring time and practice to master. [14]

3.7 Technological Limitations

Technological limitations, such as display resolution, processing power, and battery life, can hinder the effectiveness of VR and MR programming environments. These limitations can affect the quality of the virtual experience, the performance of the programming tools, and the overall usability of the system. [7]

3.8 Lower Productivity Compared to Desktops

VR and MR programming environments may result in lower productivity compared to traditional desktop setups. The unfamiliarity of the technology, the limitations of the input methods, and the potential for physical discomfort can contribute to reduced efficiency and a slower pace of development. [8]

3.9 Possible Solutions

3.9.1 Improved Text Input

To address the challenges of text input in VR and MR, alternative input methods, such as voice recognition, gesture recognition, and brain-computer interfaces, could be explored. These methods could potentially provide more efficient and intuitive ways to enter text and interact with virtual environments. [8]

3.9.2 Hybrid Approaches

Hybrid approaches that combine the benefits of VR and MR with traditional desktop setups could offer a more practical and effective solution. These approaches could involve using VR or MR for specific tasks, such as visualizing complex data or collaborating with remote colleagues, while retaining the familiarity and efficiency of desktop programming for other tasks. [8]

3.9.3 Enhancements in Ergonomics

Enhancements in the ergonomics of VR and MR devices, such as lighter headsets, improved weight distribution, and better ventilation, could help reduce physical strain and discomfort. These improvements could enable longer and more comfortable programming sessions, leading to increased productivity and user satisfaction.

3.9.4 Optimized Usability

Optimizing the usability of VR and MR programming environments is crucial for their widespread adoption. This could involve simplifying the user interface, improving the intuitiveness of the interactions, and providing clear and concise feedback to the user.

Chapter 4

Proposed Approach

4.1 Introduction

=NAME of the app= constitutes an augmented reality solution developed utilizing Unity in conjunction with the Mixed Reality Toolkit by Microsoft. This implementation integrates a proprietary extension tailored for the Visual Studio Code integrated development environment and is supported by a Python server, which facilitates the interaction and ensures seamless data exchange between the Unity application and the Visual Studio Code extension.

We anticipate that utilizing =NAME of the app= could enhance code understanding and grasp, reduce code navigation time, and thereby boost coding efficiency. We make use of the boundless capacity of the AR environment to offer an alternative perspective on the code structure, classes, and their relationships.

=NAME of the app= represents an integrative solution that harnesses the benefits of both virtual and physical environments. It amalgamates the conventional desktop setup, inclusive of a monitor, keyboard, and mouse, with the advanced capabilities of HoloLens hand gesture control. Consequently, the programmer is able to utilize the advantages offered by the virtual environment without becoming detached from the tangible world. Moreover, a hybrid approach facilitates a more seamless transition between contexts, as textual input is managed via a keyboard, while the manipulation of virtual objects is executed through HoloLens hand gestures. This integration enables the user to avoid frequent alternation between keyboard usage and controller management.

Some of the key features of our approach are:

- Displaying code as **Code Boxes** in AR.
- Linking AR elements to traditional development tools (e.g., VS Code).
- Visualization of relationships between code elements.

- Support for Python projects through automated analysis.
- Incorporating the OpenAI Coding Assistant for method generation as well as keeping a log of both prompts and their respective responses. This feature allows users to review the sequence of exchanges, offering enhanced transparency and insight into the produced code.

4.2 System Architecture

The system consists of several interrelated components that facilitate the interaction between the development environment (Visual Studio Code), the AR environment, and the Python-based server used for project analysis. The architecture is designed to enable seamless communication between these components for an enhanced development experience using augmented and virtual reality.

4.2.1 Subsystems and Components

The system architecture consists of interconnected subsystems and components, as shown in **Figure 4.1**, which facilitate seamless communication and data flow between the development environment, Python server, AR/VR system, and Unity engine. The main subsystems and their components are as follows:

- **Tethering IDE App Subsystem:**
  - **Unity Client:** Acts as a bridge between the developer's IDE and the Unity engine. It communicates with the Python server and AR/VR subsystems to enable tethered interaction.
- **Visual Studio Code Subsystem:**
  - **VSC Project Data:** Represents the project-related data handled within VS Code.
  - **VSCToUnity Extension:** Integrates VS Code with the AR/VR system and uses OpenAI API to generate methods with given context.
- **Python Server Subsystem:**
  - **VSCDataProcessor:** Processes project data received from VS Code.
  - **UnityEventsProcessor:** Handles Unity event data for further processing.
  - Communication ports include:
    - **VSC Data In Port:** Receives project data from VS Code.

    * **VSC Data Out Port:** Sends processed project data to the Unity Client.

    * **Unity Events In Port:** Receives events from Unity.

    * **Unity Events Out Port:** Sends processed Unity events back to the Unity Client.

- **AR/VR Subsystem:**

  - **Unity Engine:** The core component for rendering and managing interactions within the AR/VR environment. It processes:

    * **Unity Events Data:** Event-related information from the AR/VR environment.

  - **Meta SDK and MRTK3:** Enable AR/VR functionalities, such as interaction and visualization within the immersive environment.

  - **Monitor/User POV/Screen:** Represents the user's perspective or external visualization via screen mirroring.

4.2.2 Component Diagram

Figure 4.1 presents a component diagram of the application, illustrating the data flow between the various components.

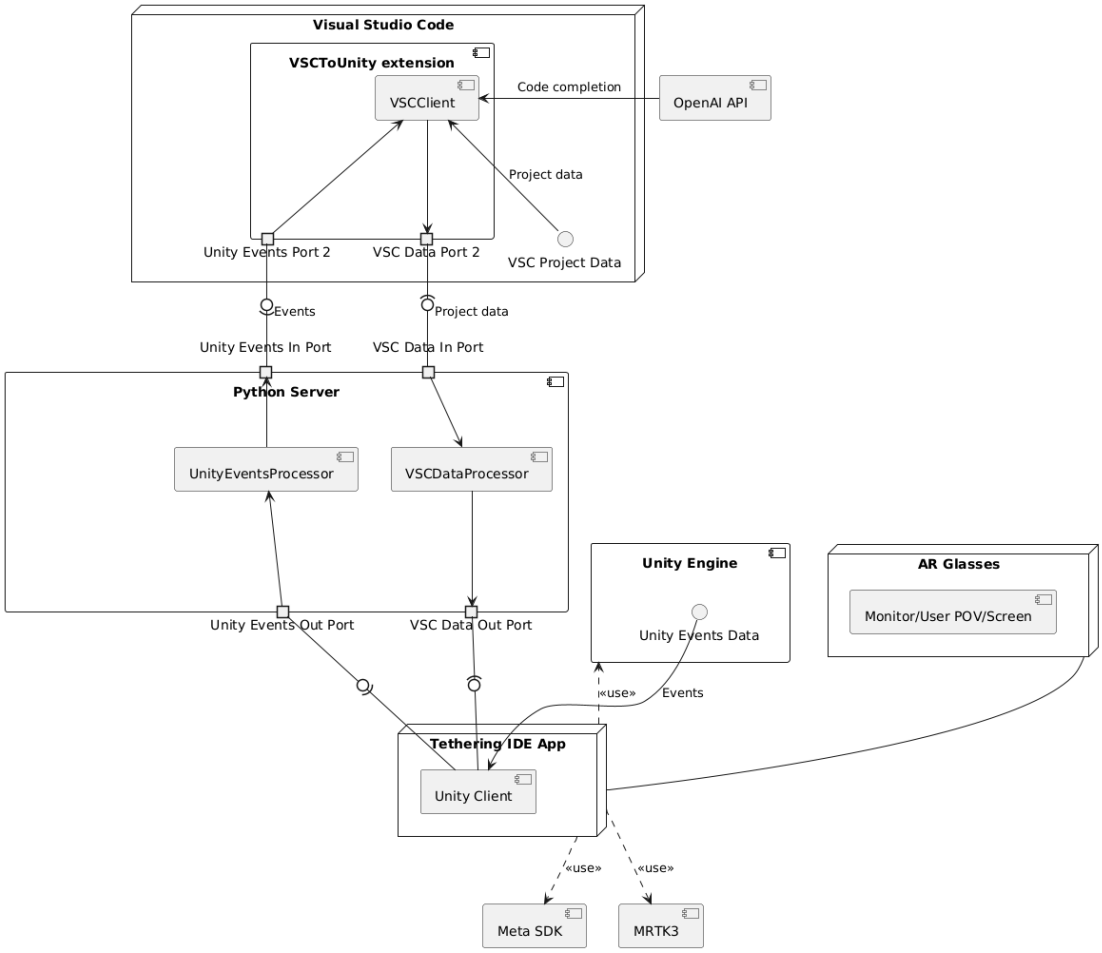


Figure 4.1: A component diagram depicting the architecture of the system.

4.2.3 Data Flow

The data flow between the Python server, Unity app, and VS Code extension is essential for ensuring smooth communication and synchronization of data. This section outlines the key interactions and how data is passed between these components.

VS Code Extension and Unity App Communication

The VS Code extension, through the VSC Client component, exchanges data with Unity:

- **Unity Events** are sent from Unity to the VS Code extension. These events represent user actions or selections, such as clicking a method in the Unity app.
- **Python Server Processing** occurs when the VS Code extension receives Unity Events. The Python server processes these events, determines the appropriate action (e.g., jumping to a specific line in the source code), and triggers the necessary command.

- The **VS Code extension** executes the command triggered by the Python server, such as navigating to a specific line of code, ensuring seamless interaction between the Unity app and the code editor.

- **Project Data**: Sent from the **VS Code extension** through a dedicated VSC Data Port, this data includes project files, configurations, and metadata used by the Unity app. The server receives the data, processes it, and returns it to Unity for further use.

- **Unity Events Data**: Events triggered in Unity are sent to the Python server for processing, where they are handled and forwarded to the VSC IDE for real-time interaction.

- **OpenAI Coding Assistant**: The OpenAI Coding Assistant communicates with OpenAI services via the OpenAI API. It includes a thread history feature that captures the user's prompt and response history.

The communication flow is represented as:

- The VS Code extension (VSCClient) sends *Project Data* to the Python server using the VSC Data Port, which is processed by the `VSCDataProcessor` component of the Python server.

- Unity app events are sent to the Python server via the Unity Events Port, where they are handled by the `UnityEventsProcessor` before being forwarded back to Unity.

- **OpenAI Coding Assistant** : During these exchanges, the thread history feature captures prompts, responses, and processing steps, enriching the understanding of code interactions and actions.

4.3 Code Analysis and Data Representation

This section describes the system's approach to project analysis and the nature of data exchanged between its components.

4.3.1 Using Python AST for Analysis

In this section, we explain how Python's Abstract Syntax Tree (AST) is leveraged to analyze a Python project. The AST provides a structured representation of the Python code, which allows us to extract key elements like classes, methods, attributes, and the

relationships between them. This method helps automate the process of understanding and visualizing code structure, making it particularly useful for generating class diagrams and understanding the dependencies between components in a project.

Extracting Classes, Methods, and Attributes

The first step in the analysis involves parsing the Python code files to extract classes, methods, and attributes. The Python AST is used to traverse the code structure and identify the key components:

- **Classes**: The `visit_ClassDef` method in the `ProjectAnalyzer` class is responsible for identifying all classes defined in the project. For each class, we capture details like the class name, its base classes (inheritance relationships), and the file in which the class is defined.

- **Methods**: Within each class, the methods are identified by checking for instances of `ast.FunctionDef` nodes, which represent function definitions. For each method, we store its name and the line number where it is defined.

- **Attributes**: The attributes of a class are usually defined in the `__init__` method, where we look for assignments to `self` (which represent instance attributes). We also check for class-level attributes outside the `__init__` method.

Analyzing Relationships Between Classes

In addition to extracting individual components, the AST analysis also identifies relationships between classes, including:

- **Inheritance**: We detect inheritance by examining the `bases` attribute of class definitions. If a class inherits from another class, the base class name is captured in the `base_classes` list.

- **Composition**: Composition relationships are identified by analyzing assignments within methods. For example, if a class creates an instance of another class (e.g., `self.books = Book()`), it is considered a composition relationship. These are tracked in the `composition` set.

- **Uses**: The `uses` relationships track any function or class calls that a class makes. For example, if a method in one class calls a function or uses a class defined elsewhere, it is recorded as a "use" relationship.

Example of the Analysis Process

Consider the following simplified example:

```
class Library:
    def __init__(self):
        self.books = []

    def add_book(self, book):
        self.books.append(book)

class Book:
    def __init__(self, title):
        self.title = title
```

Figure 4.2: Example of the Analysis Process

In this example, the `Library` class has a **composition** relationship with the `Book` class because the `Library` class creates instances of the `Book` class in the `__init__` method. The `add_book` method also **uses** the `Book` class when appending a book to the `books` list.

During the AST analysis, the `Library` class would be identified with:

- **Attributes**: `books`
- **Methods**: `__init__`, `add_book`
- **Composition**: `Book`
- **Uses**: `Book`

Similarly, the `Book` class would have:

- **Attributes**: `title`
- **Methods**: `__init__`
- **Uses**: none (in this simple example)

The resulting relationships between the classes are crucial for generating class diagrams that visualize the structure and dependencies within the code.

Generating Output

Once the analysis is complete, the collected data can be saved in multiple formats:

- **JSON**: The `to_json` method converts the analyzed data into a JSON format, which includes information about the classes, methods, attributes, and relationships. This JSON data can then be used for further processing or visualization.
- **PlantUML**: The `to_plantuml` method generates a PlantUML diagram source file that can be used to generate visual class diagrams. This diagram includes the classes, their attributes, methods, and the relationships between them (e.g., inheritance, composition, and uses).

By automating the extraction of code components and their relationships through the AST, we can efficiently create a detailed, accurate representation of the project's structure and generate corresponding diagrams for better code comprehension.

4.3.2 JSON Data Structure for Visualization

Purpose and Overview

The JSON data structure serves as an intermediary representation of the analyzed codebase, enabling visualization of class hierarchies, relationships, attributes, and methods. It encapsulates the extracted information from the source code into a structured format that can be easily parsed and displayed in various visualization tools.

The structure is designed to include:

- Class definitions along with their associated metadata.
- Relationships such as inheritance, composition, and usage between classes.
- Details of attributes and methods for each class.

Data Conversion Process

The analyzed source code is processed using an Abstract Syntax Tree (AST) parser. This parser extracts the relevant components (e.g., classes, methods, attributes, relationships) and serializes them into a hierarchical JSON format. This process ensures consistency and provides a machine-readable format that aligns with the visualization requirements.

JSON Structure Description

The JSON structure comprises the following key elements:

- **classes**: A dictionary where each key is a class name, and the value is another dictionary with detailed information about the class, including:
  - **file_path**: Absolute path of the file where the class is defined.
  - **package**: The package or module containing the class.
  - **line_number**: Line number of the class definition in the source code.
  - **methods**: A list of dictionaries, each representing a method within the class with its name and line number.
  - **attributes**: A list of class attributes.
  - **base_classes**: A list of parent classes (if any) the class inherits from.
  - **composition**: A list of classes (if any) that are composed into this class.
  - **uses**: A list of classes (if any) that this class interacts with but does not compose.
- **functions**: A list of all standalone functions detected in the source code, providing a comprehensive view of functional components.
- **imports**: A list of all imported modules and packages used within the project.

Example JSON Representations

To illustrate the JSON structure, here are examples for various scenarios:

Class Definition

This example represents the `Movie` class, including its file location, attributes, and methods.

```
{
    "Movie": {
        "file_path": "c:\\path\\to\\cinema.py",
        "package": "cinema",
        "line_number": 10,
        "methods": [
            {
                "name": "__init__",
                "line_number": 11
            }
        ],
        "attributes": [
            "duration",
            "title"
        ],
        "base_classes": [],
        "composition": [],
        "uses": []
    }
}
```

Figure 4.3: JSON Representation of the `Movie` Class

Relationships Between Classes

Here, the `CinemaHall` class is shown to compose and use the `Booking` class, indicating a strong relationship.

```
{
    "CinemaHall": {
        "composition": [
            "Booking"
        ],
        "uses": [
            "Booking"
        ]
    }
}
```

Figure 4.4: JSON Representation of `CinemaHall` Class Relationships

Attributes and Methods

This snippet highlights attributes and methods for an `Animal` class.

```
{
    "attributes": [
        "name",
        "species"
    ],
    "methods": [
        {
            "name": "make_sound",
            "line_number": 7
        }
    ]
}
```

Figure 4.5: JSON Representation of `Animal` Class Attributes and Methods

Visualization Use Case

The JSON structure is used to render diagrams and interactive visualizations, helping developers understand the architecture and relationships in the codebase. Tools consuming this JSON can dynamically generate UML diagrams, class hierarchies, or dependency graphs.

Summary

The structured format and semantic organization of this JSON ensure clarity, extensibility, and compatibility with visualization tools. It bridges the gap between raw code analysis and visual representation, forming the backbone of the architecture understanding workflow.

4.4 Integration of OpenAI Assistant Threads History

To further enhance the usability and comprehension of the code visualization tool, we have introduced an additional functionality: the integration of OpenAI Assistant threads history. This new feature is specifically designed to provide users with contextual insights into the generated methods and other code elements within the visualization framework.

Description of the Functionality

The OpenAI Assistant threads history feature allows users to view the sequence of prompts and responses that led to the generation of specific methods or other code artifacts. This is particularly beneficial for understanding the rationale and the decision-making process behind the generated code. By presenting this history alongside the code visualization, users gain a more comprehensive understanding of how the code was created and can better evaluate its accuracy and relevance.

Implementation Details

The implementation leverages the OpenAI API to capture and store the history of interactions with the Assistant. The following components form the core of this functionality:

- **Thread Creation and Management:** Each time a user initiates a request to generate or modify code, a new thread is created using the OpenAI Assistant API. The thread ID is stored for tracking subsequent interactions.

- **Event Handling for Interaction History:** Custom event handlers are implemented to capture and display the incremental output from the Assistant. These handlers process the user prompts, Assistant responses, and tool invocations.
- **Integration with Code Visualization:** A new pane has been added to the visualization tool, which displays the captured thread history in a structured format. This pane is dynamically updated to reflect the ongoing interactions and responses.

User Workflow

When using the visualization tool:

1. Users can generate methods by providing a JSON context.
2. The generated code is displayed in the Codebox.
3. Simultaneously, the associated thread history is displayed in a side panel, detailing the prompts, Assistant responses, and any tool interactions.

This seamless integration ensures that users are not only provided with the generated code but also the contextual information necessary to understand and trust the underlying logic.

4.5 AR Design and Visualization

This section provides an overview of the conceptual design of the AR application, focusing on the design of the interactive codebox that visually represents a class in the context of object-oriented programming.

4.5.1 Hexagonal Codebox Design

The codebox is designed as a hexagonal-shaped visual metaphor that represents a class in object-oriented programming. Each of the six sides of the hexagonal codebox serves a distinct purpose, visually displaying different aspects of the class. The six sides represent:

- **Side 1:** Displays the class name, attributes, and methods of the class.
- **Side 2:** Shows composition and usage relationships of the class.
- **Side 3:** Lists child classes (inheritance).
- **Side 4:** Shows parent classes (superclasses).

- **Side 5:** Provides details about how the class is used in the codebase.
- **Side 6:** Displays metadata such as the file path, package, and line number.

Figure 4.6 illustrates a code cube as seen through a VR device. Each side of the cube represents a specific component of the class, such as attributes, methods, relationships, and references.

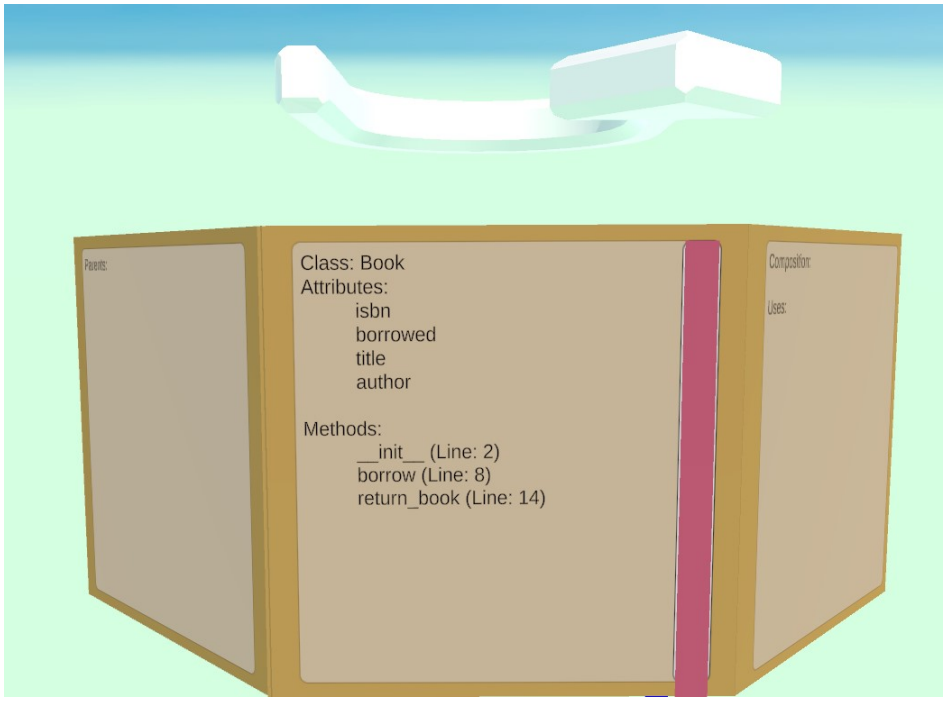


Figure 4.6: Example of a code cube.

The interactive features of the codebox allow the user to explore the class's structure by navigating between its facets. This enables a dynamic and engaging interaction with the class's components, providing a visual representation of its elements and their relationships.

The codebox is populated dynamically in the AR environment using the `CodeboxPopulator` class, written in Unity and using TextMeshPro for text rendering. Each side of the hexagonal codebox corresponds to a specific type of class data that is displayed when the user interacts with it.

This dynamic text population enhances the interactive experience by ensuring that users can access relevant information by navigating between the sides of the codebox.

This setup enables the user to explore the class structure interactively, visualizing its attributes, relationships, and usage in a dynamic, 3D environment.

Inspiration

The design of the codebox is inspired by the concept of a *code cube*, a visual metaphor for representing a class. A class is the primary structural element in object-oriented

programming, and the cube provides a way to represent various facets of a class in a 3D space. This idea was taken from the article *VR-Based User Interactions to Exploit Infinite Space in Programming Activities*[16], where a code cube is used to visualize a class in virtual reality (VR). In this approach, each facet of the cube corresponds to a different aspect of the class.

4.5.2 Relationship Visualization

In the AR application, relationships between code elements are visually represented by lines connecting different codeboxes. These lines are dynamically updated to reflect the relationships between classes based on their interactions, such as composition, usage, and inheritance. The key relationships displayed are:

- **Composition:** Represented by green lines, indicating that one class contains or is composed of another.
- **Usage:** Represented by blue lines, showing that one class uses another, typically through method calls or references.
- **Inheritance:** Represented by red lines, indicating that one class is derived from another, showing an inheritance relationship.

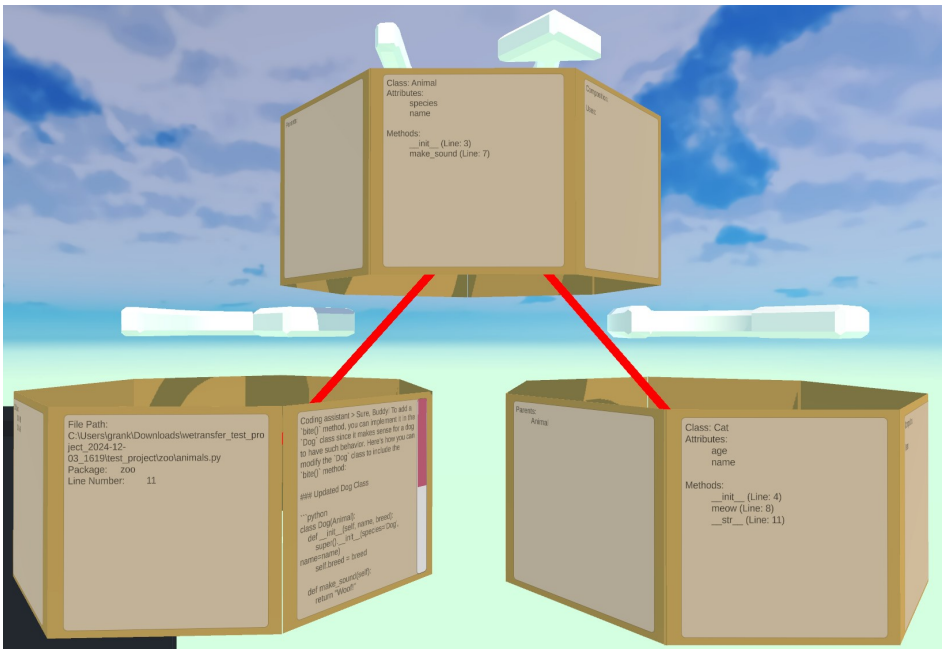


Figure 4.7: Inheritance between Animal, Dog, and Cat classes

As shown in Figure 4.7, the inheritance relationship between the Animal, Dog, and Cat classes is depicted.

These relationships are established through the `RelationshipManager` class, which uses a `LineRenderer` to create lines between the relevant codeboxes. The `RelationshipLine` class is responsible for maintaining and updating the position of these lines as the codeboxes are moved or interacted with in the AR space.

`RelationshipManager` reads the relationship data from the class definitions (e.g., composition, usage, inheritance) and creates lines accordingly. The lines are dynamically updated in the `Update()` method, ensuring that the relationship lines always stay connected to the respective codeboxes, even when their positions change in the AR environment.

4.6 Integration with Traditional Tools

In this section, we explain the relationship between AR and PC-based development to provide a smooth experience.

4.6.1 VS Code Integration

The custom VS Code extension `VSCode To Unity Client` enhances the development process by bridging the gap between traditional text-based coding in VS Code and AR-based visualization in Unity. This integration enables a seamless flow of information between the two environments, providing real-time synchronization and interaction between the code and its AR counterpart.

The extension features the following:

- **Jumping to Code Files:** The extension allows users to interact with the AR visualization in Unity and, by clicking on elements (e.g., classes), directly navigate to the corresponding code in VS Code. This jump to code feature is facilitated by the context menu and keyboard shortcuts, enabling developers to quickly inspect or edit the relevant sections of their code base without leaving the AR environment.

- **Real-Time Synchronization of Interactions:** The extension maintains a WebSocket connection between VS Code and a Python-based backend service. This ensures that any updates to the code (e.g., changes in classes or methods) are immediately reflected in the AR visualization. Developers can interact with the code through the AR interface, and the corresponding changes or class structures are synchronized in real-time. Additionally, selecting a class or method in the AR view triggers updates in the active VS Code editor, displaying the source code in the corresponding file.

- **Command Palette and Keybindings:** The extension registers two main commands in the VS Code Command Palette: `Run Python Code Analyzer` and `Display Class Code Box`. These commands allow the developer to analyze Python code for class relationships, generate visualization diagrams, and display interactive code boxes representing these classes in Unity. Custom keybindings, such as `ctrl+d`, provide quick access to these actions without interrupting the workflow.

- **Context Menu Integration:** A context menu option is added to the Python files in VS Code. When a user right-clicks on a Python file, they can trigger the `Display Class Code Box` command, which generates and displays the relevant code box in Unity based on the class structure at the current cursor position. This enhances the workflow by enabling visualization of code components directly from the editor.

Through these features, the extension facilitates a highly interactive and synchronized development experience, combining the best of traditional development tools with the immersive possibilities of AR-based visualizations.

4.7 Benefits of the Proposed Solution

=NAME of the app= aims to address the challenges of code comprehension and navigation in complex software projects by offering a novel approach that merges the strengths of traditional development environments with the immersive capabilities of augmented reality.

Specifically, this approach tackles the identified problems in the following ways:

- **Combines Familiarity with Enhanced Visualization:** By integrating with VS Code, =NAME of the app= allows developers to continue using their familiar coding environment while augmenting it with AR visualizations. This minimizes the disruption to existing workflows and lowers the barrier to adoption.

- **Improved Code Comprehension:** The spatial representation of code elements as interactive codeboxes in AR provides a tangible and intuitive way to understand the structure and relationships within a codebase. This is particularly beneficial for large or unfamiliar projects where traditional text-based representations can be overwhelming.

- **Efficient Code Navigation:** The ability to directly interact with codeboxes in AR and trigger actions in VS Code (e.g., jumping to specific files or lines of

code) streamlines navigation and reduces the time spent searching for relevant information.

Compared to traditional code exploration methods, =NAME of the app= offers several anticipated advantages:

- **Enhanced Spatial Understanding:** AR visualization allows developers to grasp the code architecture in a three-dimensional space, providing a more intuitive understanding of relationships and dependencies compared to flat diagrams or textual representations.
- **Increased Engagement and Interactivity:** The immersive nature of AR encourages active exploration and interaction with the code, potentially leading to deeper understanding and faster learning.
- **Improved Collaboration:** The shared AR environment can facilitate collaborative code reviews and discussions, allowing developers to visualize and manipulate code structures together.

4.8 Limitations and Considerations

While =NAME of the app= presents a promising approach to code visualization and interaction, it is essential to acknowledge its limitations and potential challenges:

- **Hardware Dependence:** The current implementation relies on specific hardware (HoloLens and a tethered PC), which may limit accessibility and widespread adoption. Future development could explore alternative AR/VR platforms or more flexible configurations.
- **Scalability Concerns:** Visualizing extremely large or complex projects with numerous classes and intricate relationships could lead to visual clutter and performance issues in the AR environment. Strategies for managing complexity and optimizing rendering will be crucial for scalability.
- **Learning Curve:** Developers accustomed to traditional coding environments may require time to adapt to the AR interface and interaction paradigms. Intuitive design and comprehensive tutorials will be essential to minimize the learning curve.

Furthermore, the design and development of =NAME of the app= present several ongoing challenges:

- **Seamless Integration:** Ensuring smooth and reliable communication between VS Code, the Python server, and the Unity application is crucial for a cohesive user experience. Robust error handling and efficient data transfer mechanisms are necessary to maintain synchronization and prevent disruptions.
- **Visual Clutter Management:** As project complexity increases, managing the visual representation of code elements and their relationships becomes critical. Effective layout algorithms, filtering options, and interactive exploration tools are needed to prevent visual overload and maintain clarity.
- **User Experience Optimization:** Balancing the immersive nature of AR with the need for efficient code manipulation and navigation requires careful consideration of user experience factors. Iterative design and user feedback will be vital to refine the interface and interaction modalities.

Chapter 5

Implementation

Chapter 6

Evaluation

This chapter presents the evaluation methodology and results for the application, focusing on its usability, effectiveness, and potential to enhance programming workflows. The evaluation was conducted using both qualitative and quantitative methods, aiming to measure the benefits of =NAME of the app= in real-world programming tasks.

6.1 Evaluation Approach

To comprehensively evaluate the application, we adopted the following methodology:

6.1.1 Evaluation Tools

Three primary instruments were used to collect data and assess the system:

- **System Usability Scale (SUS):** A standardized questionnaire to assess the overall usability of the application.
- **User Experience Questionnaire (UEQ):** A tool to evaluate the user experience, covering aspects such as attractiveness, efficiency, and dependability.
- **Pre- and Post-Task Questionnaires:** Custom questionnaires to gather insights into users' expectations and experiences before and after completing tasks.

6.1.2 Participants

The study involved two distinct groups of participants:

- **Professional Programmers:** Individuals with extensive industry experience.
- **Advanced Students:** Students in the later stages of their computer science or related degrees.

This selection was designed to ensure a diverse range of perspectives and skill levels.

6.1.3 Experiment Design

Participants were divided into two groups, each tasked with solving programming-related problems using different methods:

- **Control Group:** Used a standard computer setup with a monitor, keyboard, and mouse.
- **Experimental Group:** Utilized =NAME of the app=, incorporating augmented reality features and HoloLens interaction.

The tasks were divided into sub-tasks designed to evaluate specific aspects of programming workflows:

1. **Task 1: Code Functionality Identification**
    - Objective: Assess participants' ability to comprehend the functionality of a specific code segment.
    - Example Task: "Analyze the given Python method and explain its purpose. Include details about its inputs, outputs, and overall functionality."
2. **Task 2: Debugging and Error Correction**
    - Objective: Evaluate participants' skill in identifying and resolving issues in a codebase.
    - Example Task: "The provided Python script contains five intentional errors. Identify and correct these errors to achieve the desired output."
3. **Task 3: Class Relationship Mapping**
    - Objective: Measure participants' ability to visualize and interpret class relationships.
    - Example Task: "Using the =NAME of the app= and source code, describe how the classes interact and suggest improvements for clarity or functionality."
4. **Task 4: Code Navigation Efficiency**
    - Objective: Assess the time and effort required to locate specific code elements.
    - Example Task: "Find the method responsible for database operations in the provided project folder. Document its functionality and dependencies."
5. **Task 5: Testing the OpenAI Coding Assistant**

- Objective: Evaluate the utility of the OpenAI Coding Assistant in generating and understanding code.
- Example Task: "Use the OpenAI Coding Assistant to generate a method for a class based on a provided JSON context. Review the generated method, analyze its correctness, and explain how the thread history feature aids in understanding the rationale behind the generated code."

6.1.4 Metrics for Evaluation

The evaluation focused on objective and subjective measures, including:

- **Task Completion Time:** Time taken by participants to complete each task.
- **Error Rate:** The number of errors introduced or left unresolved in the solutions.
- **Task Completion Rate:** Percentage of tasks successfully completed by participants.
- **Usability and Experience Scores:** Derived from SUS and UEQ results.

6.2 Evaluation Results

This section will present the results

6.2.1 Quantitative Results

Tables and graphs

6.2.2 Qualitative Feedback

Summarized insights from open-ended responses in the pre- and post-task questionnaires, highlighting user preferences, challenges, and suggestions for improvement.

6.3 Discussion

The discussion

Conclusion

Bibliography

[1] Benjamin Alexander, Yunfei Hou, Bilal Khan, and Jennifer Jin. Learn programming in virtual reality? a case study of computer science students. In *2022 IEEE Global Engineering Education Conference (EDUCON)*, pages 270–275, 2022.

[2] Vladislav Angelov, Emiliyan Petkov, Georgi Shipkovenski, and Teodor Kalushkov. Modern virtual reality headsets. In *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pages 1–5, 2020.

[3] Christopher Berns, Grace Chin, Joel Savitz, Jason Kiesling, and Fred Martin. Myr: A web-based platform for teaching coding using vr. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 77–83, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Andrew Bragdon, Steven P. Reiss, Robert Zeleznik, Suman Karumuri, William Cheung, Joshua Kaplan, Christopher Coleman, Ferdi Adeputra, and Joseph J. LaViola. Code bubbles: rethinking the user interface paradigm of integrated development environments. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 455–464, 2010.

[5] Mengyu Chen, Marko Peljhan, and Misha Sra. Entanglevr: A visual programming interface for virtual reality interactive scene generation. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, VRST '21, New York, NY, USA, 2021. Association for Computing Machinery.

[6] James Dominic, Brock Tubre, Jada Houser, Charles Ritter, Deborah Kunkel, and Paige Rodeghero. Program comprehension in virtual reality. In *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, page 391–395, New York, NY, USA, 2020. Association for Computing Machinery.

[7] Anthony Elliott, Brian Peiris, and Chris Parnin. Virtual reality in software engineering: Affordances, applications, and challenges. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 547–550, 2015.

[8] Leonard Geier, Clemens Tiedt, Tom Beckmann, Marcel Taeumel, and Robert Hirschfeld. Toward a vr-native live programming environment. In *Proceedings of the 1st ACM SIGPLAN International Workshop on Programming Abstractions and Interactive Notations, Tools, and Environments*, PAINT 2022, page 26–34, New York, NY, USA, 2022. Association for Computing Machinery.

[9] Lorenzo Gerini, Giorgio Delzanno, Giovanna Guerrini, Fabio Solari, and Manuela Chessa. Gamified virtual reality for computational thinking. In *Proceedings of the 2nd International Workshop on Gamification in Software Development, Verification, and Validation*, Gamify 2023, page 13–21, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Martin Hedlund, Adam Jonsson, Cristian Bogdan, Gerrit Meixner, Elin Ekblom Bak, and Andrii Matviienko. Blocklyvr: Exploring block-based programming in virtual reality. In *Proceedings of the 22nd International Conference on Mobile and Ubiquitous Multimedia*, MUM '23, page 257–269, New York, NY, USA, 2023. Association for Computing Machinery.

[11] Rodi Jolak, Khan-Duy Le, Kaan Burak Sener, and Michel R.V. Chaudron. Octobubbles: A multi-view interactive environment for concurrent visualization and synchronization of uml models and code. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 482–486, 2018.

[12] Lucas Kreber, Stephan Diehl, and Patrick Weil. Idevelopar: A programming interface to enhance code understanding in augmented reality. In *2022 Working Conference on Software Visualization (VISSOFT)*, pages 87–95, 2022.

[13] Jakub Kučečka, Juraj Vincúr, Peter Kapec, and Pavel Čičák. Uml-based live programming environment in virtual reality. In *2022 Working Conference on Software Visualization (VISSOFT)*, pages 177–181, 2022.

[14] Rohit Mehra, Vibhu Saujanya Sharma, Vikrant Kaulgud, and Sanjay Podder. Xrase: Towards virtually tangible software using augmented reality. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1194–1197, 2019.

[15] Leonel Merino, Alexandre Bergel, and Oscar Nierstrasz. Overcoming issues of 3d software visualization through immersive augmented reality. In *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, pages 54–64, 2018.

[16] Víctor Stefano Segura Castillo, Leonel Merino, Geoffrey Hecht, and Alexandre Bergel. Vr-based user interactions to exploit infinite space in programming ac-

tivities. In *2021 40th International Conference of the Chilean Computer Science Society (SCCC)*, pages 1–5, 2021.

[17] Nanlin Sun, Annette Feng, Ryan Patton, Yotam Gingold, and Wallace Lages. Programmable virtual reality environments. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pages 619–620, 2021.

[18] Juraj Vincur, Martin Konopka, Jozef Tvarozek, Martin Hoang, and Pavol Navrat. Cubely: virtual reality block-based programming environment. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, VRST '17, New York, NY, USA, 2017. Association for Computing Machinery.