CodeCity: On-Screen or in Virtual Reality?

David Moreno-Lumbreras[‡], Roberto Minelli[†], Andrea Villaverde[†], Jesús M. González-Barahona[†], Michele Lanza[†]

[‡]*Escuela Internacional de Doctorado @ Universidad Rey Juan Carlos, Bitergia – Móstoles, Spain*

[†]*REVEAL @ Software Institute – USI, Lugano, Switzerland* [†]*ETSIT @ Universidad Rey Juan Carlos – Fuenlabrada, Spain*

***Abstract*—Over the past decades, researchers proposed numerous approaches to visualize source code. A prominent one is CODECITY, an interactive 3D software visualization that leverages the "city metaphor" to represent software system as cities: buildings represent classes (or files) and districts represent packages (or folders). Building dimensions represent values of software metrics, such as the number of methods or the lines of code. There are many implementations of CODECITY, the vast majority of them running on-screen. Recently, some implementations visualizing CODECITY in virtual reality (VR) have appeared. While exciting as a technology, VR's usefulness remains to be proven.**

The question we pose is: *Is VR well suited to visualize* CODECITY*, compared to the traditional on-screen implementation?*

We performed an experiment in our interactive web-based application to visualize CODECITY. Users can fetch data from any git repository and visualize its source code. Our application enables users to navigate CODECITY both on-screen and in an immersive VR environment, using consumer-grade VR headsets like Oculus Quest. Our controlled experiment involved 24 participants from academia and industry. Results show that people using the VR version performed the assigned tasks in much less time, while still maintaining a comparable level of correctness.

Therefore, our results show that VR is at least equally well-suited as on-screen for visualizing CODECITY, and likely better.

***Index Terms*—codecity, city metaphor, software visualization, software evolution, reverse engineering, virtual reality, web, 3D**

I. INTRODUCTION

The *code city* metaphor is a well known technique for visualizing source code metrics in a 3D environment. It was first used by Knight and Munro in 1999 [1], and became popular with CODECITY [2], to date the most impactful tool developed on top of this metaphor, which inspired many other similar approaches [3]–[5]).CODECITY creates visualizations that are strongly reminiscent of actual cities by using layouts, topology, and metric mappings applied at an appropriate level of granularity. It shows software systems as cities that can be intuitively explored [6], by mapping metrics of artifacts (classes, files, packages) to features of the buildings (height, size, color), and placing those buildings in locations related to the position of artifacts in the system hierarchy (*i.e.,* grouped according to the nesting level of the artifact). Thus, the city metaphor offers a clear notion of locality, supports orientation, and makes explicit the underlying structural complexity.

In the last years new technologies (WEBXR [7], WEBGL [8]) have become available in web browsers, that allow for the development of 3D, VR-ready applications which are multi-platform and easy to integrate with other front-end modules and Web APIs.

In the confluence of both lines, we built our own implementation of CODECITY as a part of a new toolset for VR data visualization, BABIAXR. We wanted to show that something similar to the original CODECITY can be implemented with relative ease, but with greater accessibility (since it runs in any recent web browser), and being able to run both on-screen or in virtual reality (VR) devices, such as the Oculus Quest.

We also reused an existing toolset for the retrieval and analysis of data from software development repositories to easily build automated pipelines capable of producing the kind of data needed by CODECITY visualizations. This allowed for a complete decoupling of data retrieval, data analysis, and data visualization.

Our implementation gave us a unique chance: testing to which extent the added accidental complexity of VR, when compared to traditional on-screen visualizations (*i.e.,* acquiring devices, navigating the immersive environment) is justified by its benefits (*i.e.,* complete immersion). We could also test if correctness of the interpretation of the code city was similar in both environments.

The nature of CODECITY is spatial as the VR environment. As a consequence, we present in this paper a controlled experiment for the comparative evaluation of two CODECITY-based approaches: on-screen and VR immersion. The main aim is to check if VR immersion is at least as effective and efficient as on-screen, for the same visualization. We designed a set of program comprehension tasks and analyzed both the correctness of the task solutions and the task completion time. Our experiment involved 24 participants from both industry and academia divided into two groups, with the same conditions in order to prove that both groups are able to solve software comprehension and software structure tasks in a similar time and correctness.

The main two contributions presented in this paper are:

1) The implementation of CODECITY in BABIAXR, which can be used in any device with a modern browser, including VR devices, making the use of this type of visualization more accessible to users. The implementation is distributed as open source software, and is therefore completely reusable.
2) The results of our experiment, showing that the VR approach is viable and has benefits compared to on-screen applications. Even when these results still have to be validated with larger samples of users, they are a first evidence in this direction.

II. BabiaXR-CodeCity

This section details BabiaXR-CodeCity, our tool to make CodeCity-like visualization accessible both on-screen and in an immersive VR environment. We also detail the process to produce a BabiaXR-CodeCity scene starting from the source code available in a Git repository.

A. BabiaXR

BabiaXR-CodeCity is a part of BabiaXR,[1] a toolset for 3D data visualization in the browser. BabiaXR is based on A-Frame,[2] an open web framework to build 3D, augmented reality (*i.e.,* AR), and VR experiences in the browser. A-Frame extends `HTML` with new entities allowing to build 3D scenes as if they were `HTML` documents, using techniques common to any front-end web developer. A-Frame is built on top of Three.js,[3] which uses the WebGL API available in all modern browsers.

BabiaXR extends A-Frame by providing components to create visualizations, simplify data retrieval, and manage data (*e.g.,* data filtering or mapping of fields to visualization features). Scenes built with BabiaXR can be displayed on-screen, or in VR devices, including consumer-grade headsets. Figure 1 shows a sample scene built with BabiaXR. BabiaXR is open source: Its source code is available on GitLab[4] and it can be installed with npm.[5]

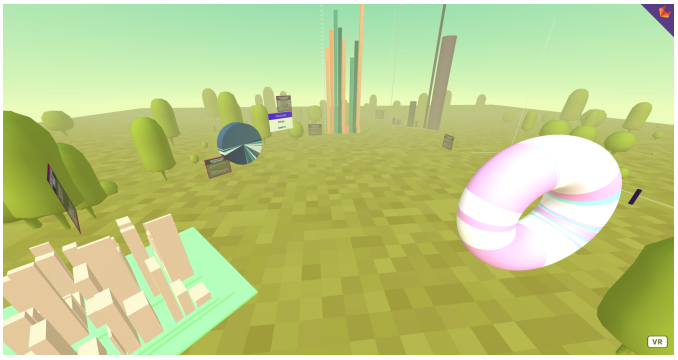


Fig. 1: Example of a BabiaXR Scene

B. CodeCity-*like Visualizations in* BabiaXR

BabiaXR provides two visualizations based on CodeCity. In this paper we present BabiaXR-CodeCity, which reimplements the original CodeCity but uses a different algorithm to layout the city, and presents the 3D visualization in a web browser (*i.e.,* instead of being a desktop Smalltalk application). For the city layout, BabiaXR employs a *spiraling algorithm*: the first element is placed at the center of the spiral and the remaining elements spiral around it.

[1]BabiaXR: **`https://babiaxr.gitlab.io`**
[2]A-Frame: **`https://aframe.io`**
[3]Three.js: **`https://threejs.org`**
[4]**`https://gitlab.com/babiaxr/aframe-babia-components`**
[5]**`https://npmjs.org/package/aframe-babia-components`**

The algorithm is used recursively at all the levels of the software architecture, producing a layout in districts, that are composed of subdistricts, and so on, until the buildings are displayed at the deepest level. Figure 2 shows a scene depicted with BabiaXR-CodeCity.

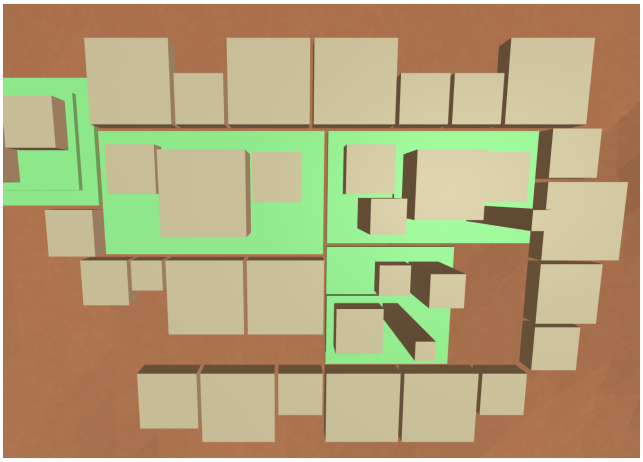


Fig. 2: Example of a BabiaXR-CodeCity Scene

BabiaXR-CodeCity scenes are interactive. The user can hover the cursor on a building to open a tooltip containing the name of the file together with the values of the metrics for the corresponding software artifact (*e.g.,* number of functions, lines of code, and Cyclomatic Complexity Number [9] of a file). The tooltip disappears when the cursor leaves the building, but pinned tooltips can be enabled by clicking on a building. To obtain information on a district (*i.e.,* a folder) the user can click on it. These interactions work in the same way on-screen (*i.e.,* with the mouse as cursor) and in VR (*i.e.,* with the controller of the VR headset as cursor). Like the original CodeCity, BabiaXR-CodeCity maps the values of software metrics to features in the visualization. In the current version of BabiaXR-CodeCity, each building corresponds to a file. Its base area is proportional to the number of functions, its height corresponds to lines of code per function, and its color represents the Cyclomatic Complexity value (*i.e.,* in a blue to red scale).

C. *From Source Code to a 3D Scene*

BabiaXR-CodeCity can produce a scene starting from a commit in a Git repository (*i.e.,* snapshot). A Python script (*i.e.,* cocom_graal2es.py[6]) clones the repository, checks out a given commit, computes the values of metrics for each file, and stores them in an ElasticSearch[7] database. The script uses Graal [10] and Perceval [11] to retrieve and compute the values of metrics. Graal, in turn, uses other tools to compute metrics, via its CoCom backend[8]).

[6]cocom_graal2es.py docs: **`https://gitlab.com/babiaxr/aframe-babia-components/-/tree/master/tools/generate_repository_data`**
[7]ElasticSearch: **`https://www.elastic.co/`**
[8]Graal-CoCom backend: **`https://github.com/chaoss/grimoirelab-graal#backends`**

Once the results of the analysis are stored in the database, another Python script (*i.e.,* GET_LIST.PY[9]) queries ELASTICSEARCH to produce a `JSON` document in the format required by BABIAXR-CODECITY that contains all the information needed for the visualization. It is structured as follows:

```
[
  {
    "file_path": "aaa/bbb/ccc"
    "metric": x,
    "metric2": y,
    ...
  },
  ...
]
```

The scene to visualize this data is composed of a single `HTML` file, which uses the `JSON` document. The `HTML` file imports all the dependencies (*i.e.,* A-FRAME and BABIAXR JavaScript packages) and defines the scene by including the corresponding elements and components: `babia-queryjson` to retrieve the `JSON` document, `babia-treebuilder` to generate the tree-like data structure needed by BABIAXR-CODECITY, and `babia-boats` which is the actual component to generate the visualization. Each component has its own configuration, detailed in the documentation.[10] The listing below shows a sample scene, including some configuration parameters:

```
<a-scene id="scene">
  <a-entity id="rawdata" babia-queryjson="url: data.json"></
      a-entity>
  <a-entity id="treedata" babia-treebuilder="field: field_list;
      split_by: /; from: rawdata"></a-entity>
  <a-entity id="city" babia-boats="from: treedata; area:
      metric1; height: metric2; color: metric3">
  </a-entity>
 ...
</a-scene>
```

Figure 3 summarizes the complete workflow to produce a scene with BABIAXR-CODECITY starting from a source code snapshot in a GIT repository.

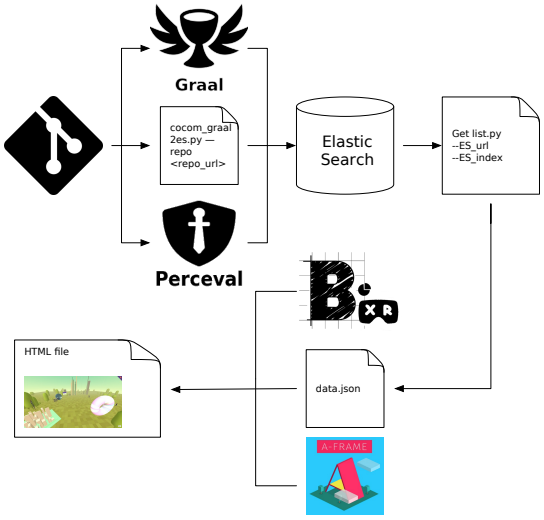


Fig. 3: BABIAXR Workflow: From Source Code to a Scene

III. EXPERIMENT DESIGN

To understand whether VR is well suited to present CODECITY-like visualizations compared to the traditional on-screen representation we conducted a controlled experiment, described in this section. We reviewed and followed the *ACM SIGSOFT Empirical Standards* [12] for designing the experiment, specifically those relevant for the quantitative method for experiments with humans, satisfying all essential attributes and a part of the desirable ones.

A. Research Questions

The same implementation of BABIAXR-CODECITY can be used both in immersive VR and on-screen. We take advantage of this fact to conduct an experiment that compares the same visualization (*i.e.,* the same software system and the same visualization elements) in both environments. Our experiment aims at understanding how accurate (*i.e.,* correctness) and efficient (*i.e.,* time spent) are participants immersed in VR compared to participants working on-screen when performing the same software comprehension tasks. This experiment is inspired by the original experiment performed by the authors of CODECITY [13].

We formulate the following research questions:

- **RQ$_1$**: How does the *accuracy* of participants immersed in VR compare to that of participants using the on-screen version of BABIAXR-CODECITY?
- **RQ$_2$**: How does the *efficiency* of participants immersed in VR compare to that of participants using the on-screen version of BABIAXR-CODECITY?

B. Target System and Visualization Metrics

In our experiment, we ask participants to perform a set of program comprehension tasks, detailed in Section III-D. Participants are shown a visualization of JETUML,[11] a lightweight desktop application to interactively create and edit Unified Modeling Language (UML) diagrams. Participants had to repeat the same tasks in two different versions of JETUML: the snapshot of March 24, 2021 (see Figure 4a) and the snapshot of June 28, 2018 (see Figure 4b).

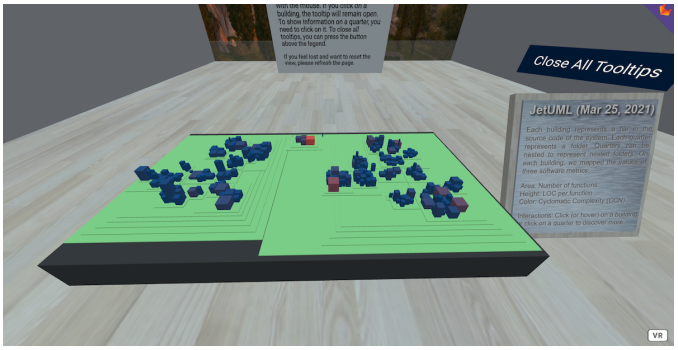
In both cases, we used for the snapshot the first commit of the day in the master branch of the GIT repository. We included two snapshots in the scene to collect feedback from participants about related, but different, city layouts.

The metrics used in the city visualization of BABIAXR-CODECITY (*i.e.,* area, height, and colors of the buildings) are customizable. For our experiment, the configuration is similar to the original CODECITY implementation: Each building represents a file where its base area represents the number of functions (*i.e.,* `num_funs`), its height represents the lines of code per function (*i.e.,* `loc_per_function`), and its color represents the Cyclomatic Complexity Number (*i.e.,* `CCN`, in a blue to red scale). Therefore, its volume represents the number of lines of code (*i.e.,* `LOC`) of the file.

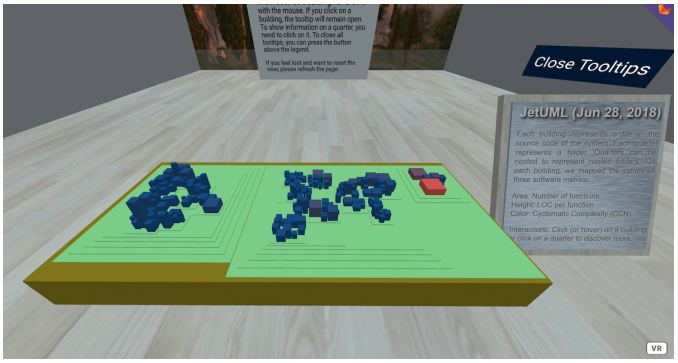
[9] **https://gitlab.com/babiaxr/aframe-babia-components/-/tree/master/tools/generate_from_es**

[10] **https://gitlab.com/babiaxr/aframe-babia-components/-/tree/master/docs/APIs**

[11] JETUML: **https://github.com/prmr/JetMUL**



(a) JETUML as of March 25, 2021



(b) JETUML as of June 28, 2018 (Release 2.1)

Fig. 4: The Two Scenes Used in the Experiment

C. Participants

Our experiment involved 24 participants from both academia and industry. We divided participants into two groups: one group tackled the tasks on-screen (SCREEN-BABIAXR) while the other group tackled the tasks in virtual reality (VR-BABIAXR). All subjects were recruited and did not have any relation with the study or the paper.

VR participants perform the experiment using an Oculus Quest 2 headset, opening the scenes in the Oculus browser. For the whole duration of the experiment, participants are required to talk aloud. The experiment is followed by a supervisor, who takes note of the answers of the participant.

On-screen participants perform the experiment in the web browser on a computer, filling in their answers via Google Forms. The experiment is followed by a supervisor. We designed the form to be easy and quick to fill in, so that there is no additional delay in answering (compared with the talk aloud mechanism used by VR participants).

In both cases the supervisor could see the scene *"with the eyes of the participant,"* and provide support if needed. For on-screen participants, the supervisor could see the screen, and for VR participants the headset was configured to cast the scene to a TV screen. Figure 5 shows a participant during the VR experiment and the screencast for the supervisor.

Demographics. Table I summarizes demographics data for our participants that include developers, Master students, PhD students, postdocs, software analysts, and a project manager.

Fig. 5: A Participant During the VR Experiment

TABLE I: Demographics of participants

ID	Position	OOP Exp/Years	PRP Exp/Years	FNP Exp/Years	REV Exp/Years	IDE Exp/Years	Fam JetUML	VR Exp
V1	Postdoc	Expert 10+	Advanced 10+	Advanced 10+	Beginner <1	Advanced 7-10	No	No
V2	PhD Student	Expert 10+	Knowled. 1-3	Knowled. 1-3	Beginner <1	Expert 10+	No	No
V3	PhD Student	Knowled. 4-6	None <1	Beginner 1-3	Beginner <1	Advanced 4-6	No	Yes
V4	Postdoc	Knowled. 10+	Knowled. 10+	Beginner <1	Knowled. 4-6	Advanced 10+	No	No
V5	PhD Student	Knowled. 7-10	None <1	Beginner <1	None <1	Knowled. 7-10	No	No
V6	Developer	Knowled. 4-6	Beginner 1-3	Knowled. 1-3	None <1	Knowled. 1-3	No	Yes
V7	Postdoc	Expert 10+	Expert 10+	Expert 10+	Expert 7-10	Expert 10+	No	Yes
V8	Master Student	Advanced 4-6	Beginner 1-3	Knowled. 1-3	Advanced 1-3	Advanced 4-6	A little	Yes
V9	PhD Student	Beginner 1-3	None <1	Beginner <1	None <1	Knowled. 7-10	No	Yes
V10	PhD Student	Knowled. 1-3	Beginner <1	Beginner <1	Beginner <1	Advanced 7-10	No	Yes
V11	Postdoc	Knowled. 1-3	Beginner <1	Knowled. 1-3	Beginner <1	Knowled. 7-10	No	No
V12	Master Student	Advanced 4-6	Knowled. 1-3	Advanced 1-3	Beginner <1	Expert4-6	No	Yes
S1	Developer	Advanced 4-6	Advanced 1-3	Advanced 4-6	Knowled. <1	Advanced 7-10	No	N/A
S2	Developer	Advanced 4-6	Knowled. 1-3	Beginner <1	None <1	Advanced 1-3	No	N/A
S3	PhD Student	Advanced 4-6	Advanced 4-6	Beginner 1-3	Beginner 1-3	Advanced 4-6	No	N/A
S4	Developer	Knowled. 4-6	Knowled. 4-6	Knowled. 4-6	Beginner 1-3	Advanced 1-3	No	N/A
S5	Developer	Advanced 4-6	Knowled. <1	Knowled. 1-3	Beginner <1	Advanced 4-6	No	N/A
S6	Developer	Knowled. 4-6	None <1	Knowled. 4-6	None <1	Expert 7-10	No	N/A
S7	Developer	Advanced 4-6	Advanced 4-6	Knowled. 1-3	Beginner <1	Expert4-6	No	N/A
S8	Project Manager	Knowled. 4-6	Knowled. 4-6	Beginner <1	Beginner <1	Knowled. 1-3	No	N/A
S9	Developer	Expert4-6	Advanced <1	Knowled. <1	Expert4-6	Expert4-6	No	N/A
S10	Developer	Advanced 7-10	Beginner <1	Knowled. 1-3	None <1	Expert 7-10	No	N/A
S11	Developer	Expert 10+	Advanced 10+	Advanced 1-3	Beginner <1	Expert 10+	No	N/A
S12	Software Analyst	Advanced 10+	Advanced 10+	Advanced 4-6	Knowled. 10+	Advanced 10+	No	N/A

Figure 6 and Figure 7 summarize experience level and years of experience of participants in Object-Oriented Programming (*i.e.,* `Exp OOP`), Procedural Programming (*i.e.,* `Exp PRP`), Functional Programming (*i.e.,* `Exp FNP`), Reverse Engineering (*i.e.,* `Exp REV`), and in using an IDE (*i.e.,* `Exp IDE`).

None of the participants had ever used a CODECITY-like visualization before, even if some of them have heard of it. Only one participant was *"a little"* familiar with JETUML, meaning she had heard of the system, but never used it.

D. Tasks

Table II summarizes the 8 tasks that participants had to address, repeating the same 4 tasks in two different versions of JETUML: $\mathbf{T_1}$–$\mathbf{T_4}$ for the snapshot of Mar 24 2021, while $\mathbf{T_{5.1}}$–$\mathbf{T_{5.4}}$ for the snapshot of Jun 28 2018.

TABLE II: Tasks performed in the experiment

Task ID	Task Description & Purpose	Category
$\mathbf{T_1}$ / $\mathbf{T_{5.1}}$	**Description.** Explore the city to locate all the test code (*i.e.,* files and directories) of the system. **Purpose.** typically, test classes are defined in separated packages in Java projects. The participants needs to understand how the test classes are organized.	Structural Understanding
$\mathbf{T_2}$ / $\mathbf{T_{5.2}}$	**Description.** Find the three source code files (not testing files) with the highest number of functions (*i.e.,* `num_funs`) in the system. **Purpose.** Classes in an Object-Oriented systems typically have a single responsibility. In this task, participants have to locate classes with the highest number of functions (or methods), which are often good candidates for refactoring (*e.g.,* split class).	Metric Analysis
$\mathbf{T_3}$ / $\mathbf{T_{5.3}}$	**Description.** Find the three source code files (not testing files) with the highest lines of code per function (*i.e.,* `loc_per_function`) in the system. **Purpose.** Lines of code per function is a good indicator to identify classes with very large methods, hence good candidates for refactoring, maintenance activities, or quality assurance.	Metric Analysis
$\mathbf{T_4}$ / $\mathbf{T_{5.4}}$	**Description.** Find the three source code files (not testing files) with the highest Cyclomatic Complexity Number (*i.e.,* `CCN`). **Purpose.** The Cyclomatic Complexity Number measures the number of linearly independent paths through a piece of code. Most of the times, complex pieces of source code are good candidates for refactoring.	Metric Analysis
$\mathbf{T_6}$	**Description.** Now that you are more familiar with JETUML, can you locate the core part of the system, *i.e.,* the most important files or packages? Please explain briefly why do you think that this is the core. **Purpose.** This task was not considered as an integral part of the experiment, but it gave us feedback on the degree to which our visualization supports users in locating the key parts of a software system.	Tool Insight

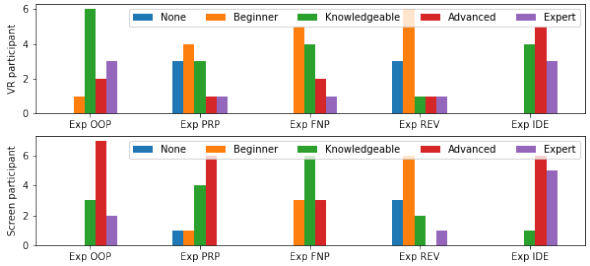


Fig. 6: Demographics: Experience Level

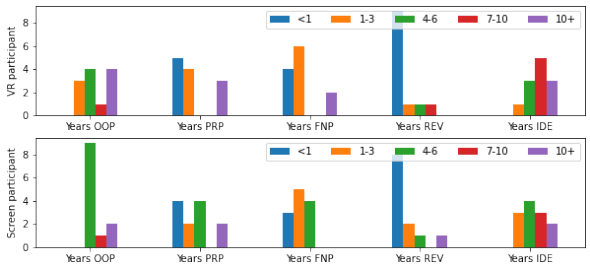


Fig. 7: Demographics: Years of Experience

To design our tasks, we leveraged the maintenance task definition framework by Sillito *et al.* [14]. We collected the data as follows:

Answers. To provide answers to questions in each task, participants of SCREEN-BABIAXR had to fill in a Google Form, while participants of VR-BABIAXR had to speak aloud for the whole duration of the experiment. For each task, participants were also asked to assess the level of difficulty (*i.e.,* from Strongly Disagree to Strongly Agree).

Efficiency. The supervisor tracked the time that each participant spent on each task. As a backup mechanism, in SCREEN-BABIAXR the supervisor also recorded a screencast of each participant's screen while in VR-BABIAXR the supervisor made a video recording of the whole experiment.

Correctness. Before running the experiments, the first author created an oracle with the correct answers to each of the tasks that was validated by the second author. When participants had to find the *Top 3 files* with respect to a metric, we checked if their answer was within the *Top 5 files* to cope with hard to distinguish visual differences.

Feedback. In the final task we asked a few high-level questions (*e.g.,* perceived difficulty, general feedback to improve the visualization). We also asked an open-ended question: *"Can you locate the core part of the system, i.e., the most important files or packages? Please explain briefly why you think that this is the core."* This task, despite not being an integral part of the experiment, gave us feedback on how our visualization supports users in locating key parts of a system.

IV. RESULTS

In this section we show how the results from the experiment allowed us to answer our research questions.

A. *Correctness* ($\mathbf{RQ_1}$) – *How does the* accuracy *of participants immersed in VR compare to that of participants using the on-screen version of* BABIAXR-CODECITY*?*

$\mathbf{T_1}$ was correctly answered by all participants (*i.e.,* both VR-BABIAXR and SCREEN-BABIAXR), who were able to locate the zone of the city where test code is displayed. The same applies also to $\mathbf{T_{5.1}}$ (*i.e.,* the same task for the other snapshot of JETUML). This is an indication that users are at ease easily locate a block of code (*i.e.,* the `test` directory) across different versions of a software system.

Table III summarizes results for $\mathbf{T_{2-4}}$ and $\mathbf{T_{5.2-5.4}}$. Tasks related to *Metric Analysis* show some differences between VR-BABIAXR and SCREEN-BABIAXR participants. For $\mathbf{T_2}$, 66.6% of VR-BABIAXR participants were able to identify the file with the highest number of functions (*i.e.,* the building with the largest base area) versus 83.3% of SCREEN-BABIAXR participants.

TABLE III: Correctness for $\mathbf{T_{2-4}}$ and $\mathbf{T_{5.2-5.4}}$ – VR vs. On-Screen

ID	VR 1st file	VR 2nd file	VR 3rd file	Screen 1st file	Screen 2nd file	Screen 3rd file	VR Top5	Screen Top5	All Top5
$\mathbf{T_2}$	66.6%	75.0%	58.3%	83.3%	83.3%	83.3%	100.0%	100.0%	100.0
$\mathbf{T_3}$	83.6%	66.6%	58.3%	100.0%	83.3%	75.0%	83.6%	100.0%	91.6%
$\mathbf{T_4}$	83.6%	83.6%	83.6%	100.0%	100.0%	91.6%	100%	100.0%	100.0%
$\mathbf{T_{5.2}}$	83.3%	91.6%	50.0%	100.0%	100.0%	91.6%	91.6%	100.0%	95.8%
$\mathbf{T_{5.3}}$	75.0%	100.0%	83.6%	100.0%	75.0%	66.6%	100.0%	91.6%	95.8%
$\mathbf{T_{5.4}}$	83.6%	100.0%	66.6%	100.0%	100.0%	91.6%	91.6%	100.0%	95.8%

The second file was identified by 75% of VR-BABIAXR participants versus 83.3% of SCREEN-BABIAXR participants. The third file was identified by 58.3% of VR-BABIAXR participants versus 83.3% of SCREEN-BABIAXR participants. If we allow for a minimal margin of error, to account for the fact that some visual differences are hard to distinguish, and compare their answers against the *Top 5 files* (instead of *Top 3 files*), all the answers are within these bounds.

Regarding $\mathbf{T_3}$, the first source code file with the highest lines of code per function (*i.e.,* represented by the height of the building) was correctly identified by 83.6% of VR-BABIAXR participants vs. 100% of SCREEN-BABIAXR participants. The second file was identified by 66.6% of VR-BABIAXR participants vs. 83.3% of SCREEN-BABIAXR participants. The third file was identified by 58.3% of participants in VR-BABIAXR vs. 75% in SCREEN-BABIAXR. Allowing for the *Top 5 files* margin of error, 83.3% of the answers in VR-BABIAXR were within these bounds vs. 100% of SCREEN-BABIAXR.

For task $\mathbf{T_4}$, the first two source code files with the highest Cyclomatic Complexity Number (*i.e.,* represented by color) were correctly identified by 83.6% of participants in VR-BABIAXR versus 100% in SCREEN-BABIAXR. The third file was identified by 83.6% in VR-BABIAXR and 91.6% in SCREEN-BABIAXR. The answers of all participants are in the *Top 5 files*.

For $\mathbf{T_5}$ we asked participants to *"go back in time"* and repeat the previous tasks on an older version of the system. We wanted to perform the same tasks in a different version of the same system, to check for consistency. For $\mathbf{T_{5.2}}$, the results are as follows: VR-BABIAXR 83.3% (1^{st} file), 91.6% (2^{nd} file), and 50.0% (3^{rd} file) versus SCREEN-BABIAXR 100% (1^{st} file), 100% (2^{nd} file), and 91.6% (3^{rd} file). The answers of 95.8% of participants are included in the *Top 5 files*. For $\mathbf{T_{5.3}}$, the results are as follows: VR-BABIAXR 75% (1^{st} file), 100% (2^{nd} file), and 83.6% (3^{rd} file) versus SCREEN-BABIAXR 100% (1^{st} file), 75% (2^{nd} file), and 66.6% (3^{rd} file). The answers of 95.8% of participants are included in the *Top 5 files*. For $\mathbf{T_{5.4}}$, the results are as follows: VR-BABIAXR 83.6% (1^{st} file), 100% (2^{nd} file), and 66.6% (3^{rd} file) versus SCREEN-BABIAXR 100% (1^{st} file), 100% (2^{nd} file), and 91.6% (3^{rd} file). The answers of 95.8% of participants are in the *Top 5 files*.

All these results show that VR-BABIAXR participants have consistently answered with lower accuracy with respect to SCREEN-BABIAXR participants. In Section V we discuss possible causes and implications.

Despite the different results, both VR-BABIAXR and SCREEN-BABIAXR participants provided similar answers with respect to the difficulty of the tasks (see Figure 8). However, more VR-BABIAXR participants found some tasks difficult. The main reason was a lack of familiarity with the VR headset and its interactions.

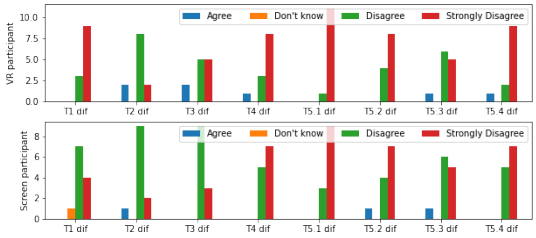


Fig. 8: Answers to "Did You Find the Task Difficult?"

B. *Completion Time* ($\mathbf{RQ_2}$) – *How does the* efficiency *of participants immersed in VR compare to that of participants using the on-screen version of* BABIAXR-CODECITY*?*

Table IV summarizes the results to answer $\mathbf{RQ_2}$, which are also presented as box plots in Figure 9. The difference in completion time between VR-BABIAXR and SCREEN-BABIAXR is remarkable.

TABLE IV: Average Task Completion Time & Statistical Analysis (Mann-Whitney U Test and Cliff's Delta effect size)

	VR (m:ss)	Screen (S) (m:ss)	Mann-Whitney U Test U	Mann-Whitney U Test p-value	Cliff's Delta
$\mathbf{T_1}$	1:47	3:57	128.0	≈0.001	0.77
$\mathbf{T_2}$	1:50	3:46	129.0	≈0.001	0.79
$\mathbf{T_3}$	1:42	3:00	137.0	<0.001	0.75
$\mathbf{T_4}$	0:57	1:38	120.0	≈0.006	0.67
$\mathbf{T_{5.1}}$	0:16	1:00	144.0	<0.001	1.00
$\mathbf{T_{5.2}}$	1:01	1:56	138.0	<0.001	1.00
$\mathbf{T_{5.3}}$	1:05	2:11	128.5	≈0.001	0.78
$\mathbf{T_{5.4}}$	0:28	1:12	144.0	<0.001	1.00
All	**1:08**	**2:20**	**7,531.0**	**<0.001**	**0.63**

Across all tasks, VR-BABIAXR participants were faster than SCREEN-BABIAXR participants. On average, they were 1 minute and 12 seconds faster. This difference reaches its peak in $\mathbf{T_1}$ where VR-BABIAXR participants were 2 minutes and 10 seconds faster than SCREEN-BABIAXR participants to identify the `test` directory. To understand whether these difference are statistically significant, we ran the *Mann-Whitney U* Test [15], a non-parametric unpaired test suitable to compare differences between two independent groups (*i.e.,* VR-BABIAXR and SCREEN-BABIAXR).

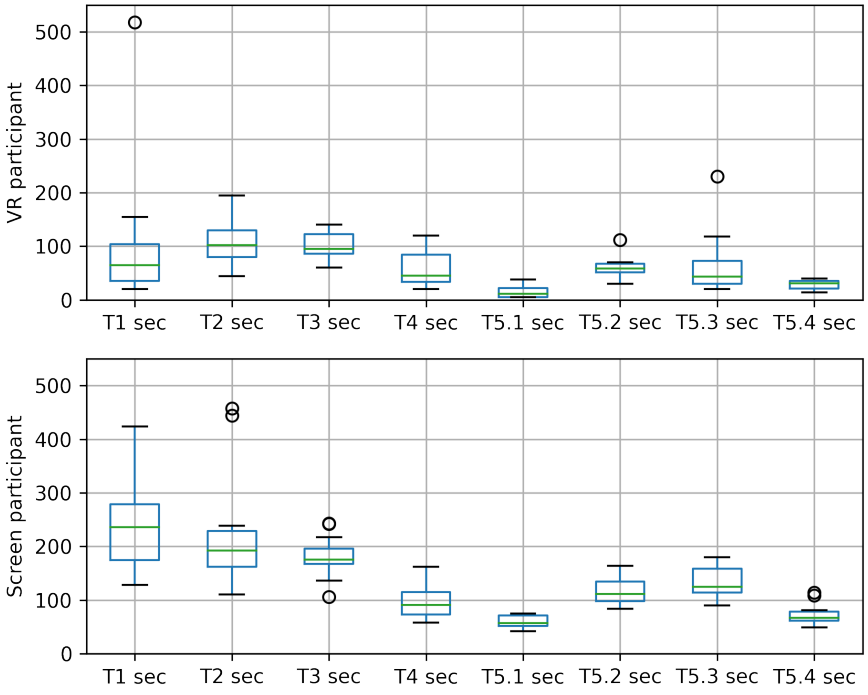


Fig. 9: Completion Times in Seconds – VR-BABIAXR vs. SCREEN-BABIAXR

Columns *U* and *p-value* in Table IV report the results: Across all tasks, differences in task completion times between VR-BABIAXR and SCREEN-BABIAXR are statistically significant. To quantify the amount of difference between both groups we also report the value of the *Cliff's Delta* effect size measure [16] for the magnitude, and the corresponding range of *Cliff's Delta* following the suggestion by Romano *et al.* [17]. Our results show that across all tasks the effect size is large (*i.e.,* $\delta > 0.474$). In Section V we discuss possible causes and implications.

C. Final Task & Feedback

At the end of the experiment, we asked participants to perform a high-level task: *Locating the "core" of JetUML*. Most participants provided a reasonable answer (*e.g.,* the `src/ca/mcgill/cs/jetuml` directory). Some entered into more details and highlighted specific subdirectories (or files) based on different features of the visualization (*e.g.,* high density of buildings, bigger classes, more complex files). Finally, we asked participants to provide us feedback on the experiment. All agreed that the experiment was not too difficult, a few reported trouble moving in the 3D scene.

V. DISCUSSION

The results of our experiment show that VR-BABIAXR participants have lower accuracy with respect to SCREEN-BABIAXR participants (except for $\mathbf{T_{5.3}}$). The difference is not high, but is also clear. There are a number of reasons that might explain this. To begin with, half of the VR-BABIAXR participants had never used a VR headset prior to our experiment. Using a VR headset is an experience which at the beginning can be quite disorienting.

Indeed, if we observe how the tasks were solved throughout the experiment in terms of efficiency and correctness, VR-BABIAXR participants seem to get better task after task. This suggests that if users are more accustomed to the use of VR headsets, the difference in accuracy could be narrowed if not completely nullified. However, more research is needed to understand whether this is really the case and we don't have enough evidence to confirm it.

When we focus on completion time, the picture is completely different: Despite lacking experience in VR headset usage, VR-BABIAXR participants were considerably faster than SCREEN-BABIAXR participants across all tasks. As discussed in Section IV, the difference across all tasks is statistically significant (according to the *Mann-Whitney U* non-parametric unpaired test) and with a large *Cliff's Delta* effect size measure. In some of the tasks, VR-BABIAXR participants took less than half the time of SCREEN-BABIAXR.

This result suggests that VR immersion could play a pivotal role when carrying out tasks in a 3D environment: Interacting

with a VR headset is similar to interacting in the real world, and you use more natural gestures than when working on-screen: you move your head to look around, you kneel down to see the world from a lower perspective, *etc.* Besides, not everybody is used to navigate 3D environments on-screen (*i.e.,* with WASD keys and a mouse). In this environment interactions are less natural and, as suggested by our results, less effective.

To better understand these results, it is important to notice that, despite our best efforts and intentions, BABIAXR-CODECITY is still a prototype and it is not optimized for VR. We see a lot of room for improvement when it comes to a number of concerns that pertain exclusively to VR:

- **Navigation.** Currently the navigation is based on gaze (*i.e.,* to select the orientation of the camera) and single-hand controller (*i.e.,* to gather additional information on entities). Many VR applications, especially games, use both hand controllers in conjunction with gaze, allowing for more natural and complex movement options. Also, there are different types of *Artificial VR Locomotion*[12] that we did not explore. This means there is still untapped potential in making the VR-BABIAXR environment more convenient and efficient in terms of navigation.
- **Physical Size.** In a classical first-person 3D on-screen visualization, there is a tacit assumption that the viewer, while navigating within the 3D environment, has no "physical" size, but is a mere point of view in the environment. In VR this is quite different. As in reality, there is the question of scale: *"How big" is the user within the environment?* If we look at Figure 4a, the user is standing in a room looking at the city visualization which sits atop a slab. From this we can infer that, if the visualization was physical, it would be as big as a large dining table. However, the city could be represented even smaller, or as a real-scale city. Which representation is more convenient and efficient? Would allowing users to change scale at will help them to be more comfortable and efficient? More research is needed to answer these questions, but we think both convenience and performance could be improved tuning this factor appropriately.

In summary, we consider that the results of our experiment show that CODECITY in VR is comparable to CODECITY on-screen, with respect to the accuracy of results, and much better with respect to efficiency. Considering that VR headsets are still a rather novel technology, and that our immersive VR version of the experiment is still under-explored, we believe our study paves the way for a number of follow-up studies and implementations.

We also have shown how our multi-platform implementation of CODECITY, that can work very similarly both on-screen and in VR, can help to fairly compare those environments for metaphors used for software comprehension.

[12]Artificial VR Locomotion:
https://developer.oculus.com/learn/artificial-locomotion

VI. THREATS TO VALIDITY

A. Internal validity

Internal validity is related to uncontrolled factors that can influence the effectiveness. In our case it pertains to:

- **Subjects.** We ensured that all the participants had experience in different relevant topics about programming using a questionnaire, reducing the threat that they were not competent enough. Moreover, we asked for their experience in the relevant topics to mitigate the threat that the participant's experience was not distributed fairly. However, their training for the environment of their experiment (*i.e.,* on-screen or VR) was not uniform, with persons participating in the VR experiment being much less experienced in VR environments than on-screen participants in on-screen environments.
- **Tasks.** The choice of tasks may have been biased in favor of VR-BABIAXR or SCREEN-BABIAXR. We mitigated this threat by developing scenes that were valid for both VR and on-screen, with exactly the same tasks, so that the level of difficulty was as similar as possible. We also included tasks that put both modes at a disadvantage: Tasks focused on precision could be easier on-screen, while tasks focused on locality could be easier in VR. Not controlled aspects (*e.g.,* the relative size of the buildings in VR) could have an influence on the results.
- **Training.** In both environments (*i.e.,* VR-BABIAXR and SCREEN-BABIAXR) the text to be followed for performing the tasks explains how the tool is used and how the interaction with the elements works. No participant had relevant previous experience with BABIAXR or CODECITY. It remains to be investigated whether a practical tutorial on how to interact with a VR headset could reduce the experience gap between VR and on-screen, improving the correctness of VR participants.

B. External validity

External validity relates to the generalizability of the results of the experiment. In our case it pertains to:

- **Sample Size.** The number of participants in the experiment is relatively small. although we have tried to select a representative sample, a larger sample would be needed to have more conclusive results.
- **Subjects.** We mitigate the threat of subject representativeness by categorizing them, including the job position and the years of experience in the programming topics, obtaining a balanced mix of academics and professionals.
- **Target System.** Another threat is represented by the choice of the target system: JETUML. Participants did not know it in advance, except for one who knew it *"a little."* We cannot assess how appropriate or representative JETUML is for the reverse engineering tasks we designed, but the consistent variations in solutions for the same task in both VR and on-screen environments signal that results could be extensible to other systems.

- **Experimenter Effect.** One of the experimenters is one of the authors of BABIAXR-CODECITY, which may have influenced any subjective aspect of the experiment. For example, task solutions may not have been graded correctly. To mitigate this threat, another author carried out part of the experiments as a supervisor. Both experimenters built a model of the responses based on previous experiments in the literature (*e.g.,* [13], [18]). Even if we tried to mitigate this threat extensively, we cannot exclude all possible influences on the results of the experiment.
- **Time Measurement.** To mitigate the threat that task completion times are not exact, in each experiment run, the supervisor was present and noted how long the participant took to complete the task. The participant also notified the supervisor when she had completed each task, thus having a double-check of the completion time. In addition, the writing time of the SCREEN-BABIAXR participants responses in the external form is not significant, so it does not pose an additional threat. Mitigating these threats, the difference of answering using a form for the SCREEN-BABIAXR participants and the aloud method for the VR-BABIAXR participants does not produce a threat. We planned to use in-scene mechanisms for answering the tasks in VR-BABIAXR and on-screen forms in SCREEN-BABIAXR, we failed to produce those for VR-BABIAXR on time, so we revered to talk aloud in VR-BABIAXR participants.

VII. RELATED WORK

A metaphor is a stable and systematic relationship between two conceptual domains, according to the theory developed by Lakoff and Johnson in the cognitive linguistics field [19]. The metaphor used depends directly on the software artifacts to represent [20], and has to be expressive enough to provide mapping for their relevant features. Metaphors have long populated the software visualization field, and the availability of 3D improved the development of more realistic and easier to grasp visual metaphors. Some examples of early 3D oriented metaphors are the landscape metaphor [21] for visualizing the structure of large systems, and the solar system metaphor [22] for visualization of object-oriented software systems. Several years later, CODETREES for visualizing software as a collection of trees [23], a concept which was later extended [24].

Our work is a part of the research line based on the *"city metaphor,"* related to civil architecture, influencing software representation. This line can be traced back to the design patterns by Gamma *et al.* [25], rooted in the architectural pattern language proposed by Alexander *et al.* [26].

A. CODECITY

The first implementation of the city metaphor was SOFTWARE WORLD [1], which already visualized software systems as buildings in a city. After it, several approaches were explored to support developers on maintaining software systems as fulfilling program comprehension tasks. In 2003 Panas *et al.* [27]–[29] presented a software city showing information about static and dynamic data, and Marcus *et al.* [30] a city-like software visualization. VERSO [31] was based on landscapes, but with a influence of the city metaphor.

In 2007 CODECITY was presented [6], raising the approach to a new level implementation-wise. Figure 10 shows the original CODECITY application, including the interface for interacting with the city and the metrics details.

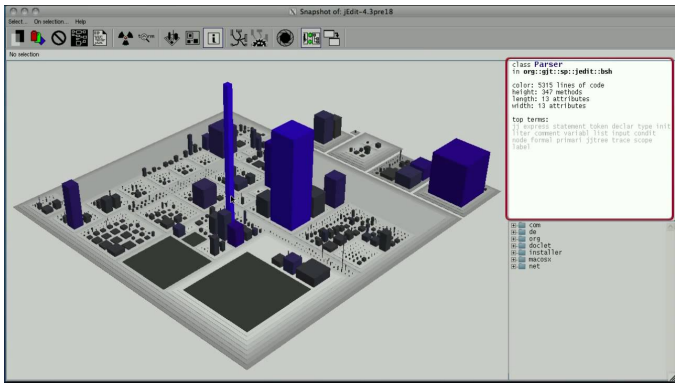


Fig. 10: The Original CODECITY Tool

CODECITY showed that it could not only be used for program comprehension [6], but also for software evolution analysis [32] and design problem analysis [33].

It sparked a flood of tools and approaches building on the same metaphor, leading to slightly different visualizations, showing the power and flexibility of the metaphor.

Scarsbrook *et al.* [34] presented a tool for visualizing and debugging large-scale JavaScript program structure with treemaps, Brito *et al.* [3] presented a similar approach focusing on the Go programming language. Steinbrückner *et al.* [35] proposed a different layout for the city, based on streets and sub-streets for the tree structure, allowing to observe the time evolution of the software system. Gamification has also been used in combination with the city metaphor to perform software comprehension tasks in CODEMETROPOLIS [36], based on the Minecraft game engine. M3TRICITY [37], a recent re-implementation of CODECITY by the original research group, is web application to visualize software systems as evolving cities that treats evolution as a first-class concept.

B. From 3D to Virtual Reality

One of the early explorations of using virtual reality for visualizing software was done by Young and Munro [38], at the time quite a technical feat. Another early VR-based approach is IMSOVISION [39], which focuses on C++, defining some metrics that nowadays are still used in the literature. More recently, thanks to technological advances, software visualizations based in VR become an active field of research.

Fittkau *et al.* proposed a VR implementation of EXPLORVIZ [40], based on the first versions of WEBVR, focusing on the runtime and static characteristics of object-oriented programming software systems. Vincur *et al.* [4], [41] proposed a VR city for analyzing object-oriented software. Steinbrückner and Lewerentz [5] proposes stable city

layouts for evolving software systems, using layouts other than treemaps. GETAVIZ [42] also uses the city metaphor to generate structural, behavioral, and evolutionary views of software systems for empirical evaluation.

CITYVR [43], developed with UNITY3D, provides the same metrics as the original CODECITY, adding interactions using gaze of the user in the VR headset, and its controllers. The technique we used of moving in the scene technique has similarities with their approach.

Capece *et al.* [44] proposed an approach to visualize Java systems with the UNREAL ENGINE 4 using the city metaphor in VR.

Other metaphors have been implemented in VR. Misiak, Schreiber *et al.* [45] proposed the island metaphor for visualizing OSGi-based software systems, introducing and emphasizing the visualization of dependencies. Schreiber *et al.* [46] presented an interactive tool that also visualizes OSGi-based systems with their components, packages, services, and dependencies in 3D, using a different metaphor including boxes.

C. Validation Experiments

Wettel *et al.* [13] proposed one of the first experiments to validate the city metaphor as a way to comprehend some aspects of software systems. Our experiment is designed in a great deal after it. More recently, Romano *et al.* [18] conducted a controlled experiment where they asked the participants to perform program comprehension tasks with the support of the ECLIPSE Integrated Development Environment (*i.e.,* IDE) with a plugin for gathering code metrics and identifying bad smells and a visualization tool of the city metaphor displayed on a standard computer screen and in an immersive virtual reality. Our results are partially in line with those shown in [18], having as similarities that the VR-BABIAXR participants are faster than the SCREEN-BABIAXR participants and that the correctness does not worsen significantly.

VIII. CONCLUSIONS AND FUTURE WORK

We presented a controlled experiment aimed at evaluating whether VR is well suited to visualize CODECITY visualizations compared to the traditional on-screen implementations. The experiment presented the same CODECITY-like visualization of a software system in VR and on-screen using BABIAXR-CODECITY, part of the BABIAXR toolset for 3D data visualization that we developed. Subjects from both academia and industry, with a wide range of experience, performed 8 tasks related to software comprehension in one of the two environments, VR or on-screen.

Our results show that immersion in VR led to much shorter completion time for the tasks, compared to on-screen. On the downside, subjects in VR performed with less accuracy with respect to on-screen participants. However, for most of the tasks if we extend the bounds to the *Top 5* closer results, correctness is always over 90%, so we can affirm that the VR version is valid for solving those tasks. We believe their higher error rate was related to their lack of experience with virtual reality devices, the still prototypical stage of our implementation, and the visual similarity of the buildings in some of the proposed tasks. The feedback collected from participants is consistent with our appreciation. Taking everything into account, the experiment shows that VR provides a better user experience than on-screen for issues related to locating, moving, and searching for elements in the scene. This explains the fact that time to complete is about half in most tasks performed in VR.

Considering these results, it could happen that VR might let CODECITY-style visualizations reach the *"tipping point,"* beyond which they are viable and useful for practitioners.

Virtual reality does not only provide better performance in terms of speed, and some reasonable but still improvable performance in terms of accuracy. It also offers a set of unprecedented features, such as the chances of collaboration between teams of people immersed in the same VR environment, which we find very promising and it deserves to be explored. The improvement of accuracy in VR, and the exploration of these new possibilities, are interesting lines of long-term future work.

With respect to short-term future work, we would like to validate our results by performing similar experiments with larger samples of participants. In addition, the feedback of participants, and some more usability experiments, could be leveraged to improve VR BABIAXR-CODECITY scenes and their user experience, so they become more user friendly. For that, we plan to improve the interaction, experiment with different Artificial VR Locomotion techniques, add an overlay grid to help in appreciating dimensions, use predesigned camera positions to have different types of views, and have a better access to metrics, *etc.*

Replication package: The data obtained for our experiment, and the materials needed to reproduce the experiment are available in the *Replication Package*.[13]

Acknowledgements: We acknowledge the financial support of the Spanish Government for the projects IND2018/TIC-9669 and RTI-2018-101963-B-I00 and the Swiss National Science foundation (SNSF) for the NRP-75 project 167173. We also thank all the participants of our experiments.

REFERENCES

[1] C. Knight and M. Munro, "Comprehension with[in] virtual environment visualisations," in *Proceedings Seventh International Workshop on Program Comprehension*, 1999, pp. 4–11.

[2] R. Wettel and M. Lanza, "Visualizing software systems as cities," in *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis*, 2007, pp. 92–99.

[3] R. Brito, A. Brito, G. Brito, and M. T. Valente, "GoCity: Code city for Go," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2019, pp. 649–653.

[4] J. Vincur, P. Navrat, and I. Polasek, "VR City: Software analysis in virtual reality environment," in *2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, 2017, pp. 509–516.

[5] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proceedings of the 5th International Symposium on Software Visualization*, ser. SOFTVIS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 193–202.

[13]Replication Package: `https://doi.org/10.5281/zenodo.4972931`

[6] R. Wettel and M. Lanza, "Program comprehension through software habitability," in *15th IEEE International Conference on Program Comprehension (ICPC '07)*, 2007, pp. 231–240.

[7] B. Jones and M. Goregaokar, "WebXR device API," W3C Working Draft, Jul. 2020.

[8] D. Jackson and J. Gilbert, "WebGL 2.0 specification," Khronos Group Specification, Dec. 2020.

[9] Harrison, Magel, Kluczny, and DeKock, "Applying software complexity metrics to program maintenance," *Computer*, vol. 15, no. 9, pp. 65–79, 1982.

[10] V. Cosentino, S. Dueñas, A. Zerouali, G. Robles, and J. M. Gonzalez-Barahona, "Graal: The quest for source code knowledge," in *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2018, pp. 123–128.

[11] S. Dueñas, V. Cosentino, G. Robles, and J. M. Gonzalez-Barahona, "Perceval: Software project data at your will," in *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1–4.

[12] P. Ralph, "ACM SIGSOFT empirical standards released," *SIGSOFT Softw. Eng. Notes*, vol. 46, no. 1, p. 19, Feb. 2021.

[13] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: a controlled experiment," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 551–560.

[14] J. Sillito, G. C. Murphy, and K. De Volder, "Questions programmers ask during software evolution tasks," in *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. SIGSOFT '06/FSE-14. New York, NY, USA: Association for Computing Machinery, 2006, p. 23–34.

[15] N. Nachar, "The Mann-Whitney U: A test for assessing whether two independent samples come from the same distribution," *Tutorials in Quantitative Methods for Psychology*, vol. 4, 03 2008.

[16] N. Cliff, "Dominance statistics: Ordinal analyses to answer ordinal questions." *Psychological Bulletin*, vol. 114, pp. 494–509, 1993.

[17] J. Romano, J. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys?" in *annual meeting of the Florida Association of Institutional Research*, 2006, pp. 1–3.

[18] S. Romano, N. Capece, U. Erra, G. Scanniello, and M. Lanza, "On the use of virtual reality in software visualization: The case of the city metaphor," *Information and Software Technology*, vol. 114, pp. 92 – 106, 2019.

[19] G. Lakoff and M. Johnson, *Metaphors we live by*. University of Chicago Press, 1980.

[20] V. Averbukh, "Visualization metaphors," *Programming and Computer Software*, vol. 27, pp. 227–237, 09 2001.

[21] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz, "Software Landscapes: Visualizing the structure of large software systems." 01 2004, pp. 261–266.

[22] H. Graham, H. Yang, and R. Berrigan, "A Solar System metaphor for 3D visualisation of object oriented software metrics." 01 2004, pp. 53–59.

[23] U. Erra and G. Scanniello, "Towards the visualization of software systems as 3D forests: The CodeTrees environment," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, ser. SAC '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 981–988.

[24] K. Maruyama, T. Omori, and S. Hayashi, "A visualization tool recording historical data of program comprehension tasks," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 207–211.

[25] E. Gamma, R. Helm, R. Johnson, and J. M. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1st ed. Addison-Wesley Professional, 1994.

[26] C. Alexander, S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press, August 1977.

[27] T. Panas, R. Berrigan, and J. Grundy, "A 3D metaphor for software production visualization," in *Proceedings on Seventh International Conference on Information Visualization, 2003. IV 2003.*, 2003, pp. 314–319.

[28] T. Panas, R. Lincke, and W. Löwe, "Online-configuration of software visualizations with Vizz3D," 01 2005, pp. 173–182.

[29] T. Panas, T. Epperly, D. Quinlan, A. Sæbjørnsen, and R. Vuduc, "Communicating software architecture using a unified single-view visualization," 08 2007, pp. 217–228.

[30] A. Marcus, L. Feng, and J. Maletic, "3D representations for software visualization." 01 2003, pp. 27–36, 207.

[31] G. Langelier, H. Sahraoui, and P. Poulin, "Visualization-based analysis of quality for large-scale software systems," 01 2005, pp. 214–223.

[32] R. Wettel and M. Lanza, "Visual exploration of large-scale system evolution," in *2008 15th Working Conference on Reverse Engineering*, 2008, pp. 219–228.

[33] R. Wettel and M. Lanza, "Visually localizing design problems with disharmony maps," in *Proceedings of the 4th ACM Symposium on Software Visualization*, ser. SoftVis '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 155–164.

[34] J. D. Scarsbrook, R. K. L. Ko, B. Rogers, and D. Bainbridge, "MetropolJS: Visualizing and debugging large-scale JavaScript program structure with treemaps," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 389–3893.

[35] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *Proceedings of the 5th International Symposium on Software Visualization*, ser. SOFTVIS '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 193–202.

[36] G. Balogh, T. Gergely, A. Beszedes, and T. Gyimothy, "Using the city metaphor for visualizing test-related metrics," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 2, 2016, pp. 17–20.

[37] F. Pfahler, R. Minelli, C. Nagy, and M. Lanza, "Visualizing evolving software cities," in *2020 Working Conference on Software Visualization (VISSOFT)*, 2020, pp. 22–26.

[38] P. Young and M. Munro, "Visualising software in virtual reality," in *Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242)*, 1998, pp. 19–26.

[39] J. Maletic, J. Leigh, A. Marcus, and G. Dunlap, "Visualizing object-oriented software in virtual reality," in *Proceedings 9th International Workshop on Program Comprehension. IWPC 2001*, 2001, pp. 26–35.

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

[41] J. Vincur, I. Polasek, and P. Navrat, "Searching and exploring software repositories in virtual reality," in *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '17. New York, NY, USA: Association for Computing Machinery, 2017.

[42] D. Baum, J. Schilbach, P. Kovacs, U. Eisenecker, and R. Müller, "GETAVIZ: Generating structural, behavioral, and evolutionary views of software systems for empirical evaluation," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, 2017, pp. 114–118.

[43] L. Merino, M. Ghafari, C. Anslow, and O. Nierstrasz, "CityVR: Gameful software visualization," in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2017, pp. 633–637.

[44] N. Capece, U. Erra, S. Romano, and G. Scanniello, "Visualising a software system as a city through virtual reality," in *Augmented Reality, Virtual Reality, and Computer Graphics*, L. T. De Paolis, P. Bourdot, and A. Mongelli, Eds. Cham: Springer International Publishing, 2017, pp. 319–327.

[45] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, and L. Nafeie, "IslandViz: A tool for visualizing modular software systems in virtual reality," in *2018 IEEE Working Conference on Software Visualization (VISSOFT)*, 2018, pp. 112–116.

[46] A. Schreiber and M. Brüggemann, "Interactive visualization of software components with virtual reality headsets," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, 2017, pp. 119–123.