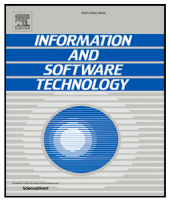
Information and Software Technology 153 (2023) 107064

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



CodeCity: A comparison of on-screen and virtual reality

David Moreno-Lumbreras [a,*], Roberto Minelli [b], Andrea Villaverde [c], Jesus M. Gonzalez-Barahona [c], Michele Lanza [b]

[a] Escuela Internacional de Doctorado @ Universidad Rey Juan Carlos & Bitergia, Móstoles & Leganés, Spain
[b] REVEAL @ Software Institute, USI, Lugano, Switzerland
[c] ETSIT @ Universidad Rey Juan Carlos, Fuenlabrada, Spain

ARTICLE INFO

Keywords:
CodeCity
City metaphor
Software visualization
Software evolution
Reverse engineering
Virtual reality
Web
3D

ABSTRACT

Context: Over the past decades, researchers proposed numerous approaches to visualize source code. A popular one is *CodeCity*, an interactive 3D software visualization representing software system as cities: buildings represent classes (or files) and districts represent packages (or folders). Building dimensions represent values of software metrics, such as number of methods or lines of code. There are many implementations of *CodeCity*, the vast majority of them running on-screen. Recently, some implementations using virtual reality (VR) have appeared, but the usefulness of *CodeCity* in VR is still to be proven.
Aim: Our comparative study aims to answer the question *"Is VR well suited for CodeCity, compared to the traditional on-screen implementation?"*
Methods: We performed two experiments with our web-based implementation of *CodeCity*, which can be used on-screen or in immersive VR. First, we conducted a controlled experiment involving 24 participants from academia and industry. Taking advantage of the obtained feedback, we improved our approach and conducted a second controlled experiment with 26 new participants.
Results: Our results show that people using the VR version performed the assigned tasks in much less time, while maintaining a comparable level of correctness.
Conclusion: VR is at least equally well-suited as on-screen for visualizing *CodeCity*, and likely better.

1. Introduction

Representing source code metrics as features of buildings in a city is a well known metaphor for visualizing software in 3D, introduced in 1999 by Knight and Munro in their Software World [1]. It became popular with *CodeCity* [2], which also inspired some other similar approaches (*e.g.,* [3–5]). *CodeCity* shows software systems as cities by mapping metrics of artifacts (*i.e.,* classes, files) to features of buildings (*i.e.,* height, size, color), and placing buildings in locations related to the position of artifacts in the system hierarchy. Those cities can be intuitively explored [6], offering a clear notion of locality, supporting orientation, and making explicit the underlying structural complexity. Most *CodeCity* implementations work on traditional 2D screens, but recently some are also exploring how well the metaphor works in virtual reality (VR).

VR technology has been under development for decades. During the last few years, new devices have appeared that are both low-cost and standalone (*i.e.,* they do not require to be tethered to a computer that provides the rendering capabilities of its GPU), making VR more affordable and accessible. Browsers have implemented some new APIs, such as *WebXR* [7] and *WebGL* [8], which enables them to support 3D scenes and user interaction in VR. Applications developed using these APIs run in the browser, thus are inherently multi-platform, and are easy to integrate with other web front-end modules and APIs.

Because of this, browsers have become appropriate for comparing on-screen with VR immersion: they are available both in traditional screen devices and in VR devices, and it is not difficult to build applications for them.

In the study presented in this paper, we take advantage of this situation by using a browser-based implementation of *CodeCity* to compare how users interact with it on screen and in immersive VR. Our aim is to explore if the affordances of VR, which build on the spatial capabilities of the human brain but add some extra complexity (*e.g.,* being used to unfamiliar devices, navigating the immersive environment), may help when working with *CodeCity* for performing software comprehension tasks. Since the underlying implementation used for on-screen and VR is the same, differences in the behavior of subjects are expected to be due exclusively to differences between interactions in a screen-based environment and in VR immersion.

* Corresponding author.
E-mail address: d.morenolu@alumnos.urjc.es (D. Moreno-Lumbreras).

https://doi.org/10.1016/j.infsof.2022.107064
Received 17 December 2021; Received in revised form 23 August 2022; Accepted 2 September 2022
Available online 9 September 2022

We decided to focus on two measurable properties that have a clear impact on how helpful *CodeCity* may be in supporting software comprehension tasks: *accuracy* and *time spent*. We designed our experiments as a set of software comprehension tasks and compared how accurate were the results reported by participants, and how much time they spent until they could report a result.

Our study aims to answer two 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*?

In the first experiment 24 subjects performed a set of program comprehension tasks, by interacting with the same *CodeCity* visualization of a given project. Half of the subjects performed the experiment on-screen and the other half used a VR device. Participants were both from industry and academia, and we tried to make all other conditions as similar as possible to avoid confounding variables.

After analyzing the results of this experiment, including the feedback received from participants, we decided to run a second experiment to overcome some of the limitations that could threaten the validity of the results. We expected to confirm or refute those results and obtain more solid evidence. The main factors that we fine-tuned in the second experiment were fixing differences in training, and in measuring some parameters of the experiment, and adapting the VR scene so that subjects could interact more comfortably with it. This experiment involved 26 new participants.

We implemented the *CodeCity* visualization that is used in the experiments as a part of *BabiaXR*,[1] a 3D visualization library designed for being used both in traditional browsers, simulating 3D on 2D screens, and on browsers in VR devices, in this case as VR immersive scenes. The data to be visualized (*i.e.,* the metrics displayed by *CodeCity*) was retrieved, analyzed, and preprocessed using GrimoireLab [9], which allows for complete decoupling of data retrieval, data analysis, and data visualization.

The main contributions presented in this paper[2] are:

1. A **methodology** for comparing the behavior of subjects while interacting with software comprehension applications using traditional screens or VR devices.
2. The **results** of two controlled experiments, showing that the VR approach is viable for performing software comprehension tasks facilitated by *CodeCity*, and has benefits when compared to on-screen applications.
3. An **open source implementation** of *CodeCity* which runs on any modern browser, making *CodeCity* VR visualizations more accessible and our experiments more easily reproducible.

2. BabiaXR-CodeCity

BabiaXR-CodeCity is part of *BabiaXR*,[3] an open source toolset for 3D data visualization in the browser available from the npm repository.[4] *BabiaXR* is based on *A-Frame*,[5] a framework for building 3D, augmented reality (AR), and VR experiences in the browser. *A-Frame* extends `HTML` with new entities, allowing the description of 3D scenes as a part of a `HTML` document, which the browser uses to build a part of its internal DOM. Therefore, the scene can be manipulated by manipulating the DOM, using techniques common to any front-end web developer.

Fig. 1. Example of a *BabiaXR* scene.

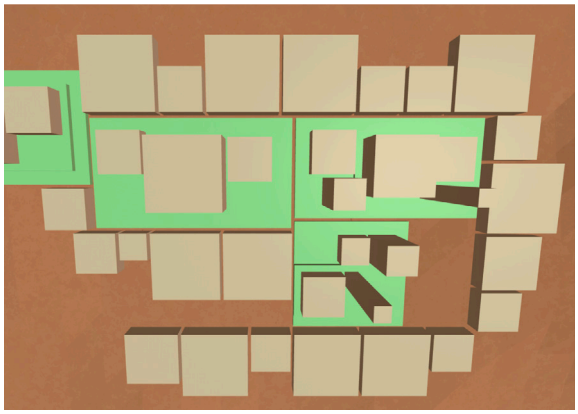


Fig. 2. Example of a *BabiaXR-CodeCity* scene.

A-Frame is built on top of *Three.js*,[6] which in turn uses the *WebGL* API provided by all modern browsers.

BabiaXR extends *A-Frame* by providing components to create visualizations, to retrieve data, and to manage it (*e.g.,* filter). Scenes built with *BabiaXR* can be displayed on-screen or on VR devices, thanks to the facilities provided by browsers. Fig. 1 shows a sample scene.

2.1. CodeCity in BabiaXR

As the original *CodeCity*, our implementation maps metrics to features of buildings, and groups buildings according to the hierarchical structure of the source code. However, there are some differences. The original implementation was a desktop-based *Smalltalk* application, while ours is built as a *JavaScript* application running in the browser. The layout of the original *CodeCity* was based on hierarchically splitting a rectangle using a treemap algorithm, but we use a *spiraling algorithm*: within each hierarchical level, the first building is placed in the center of a spiral, and the remaining buildings are positioned spiraling around it. We use the same algorithm also to layout groups of buildings at the same hierarchical level. Fig. 2 shows an example of the layout produced by our implementation of *CodeCity*.

The spiral grows clock-wise, starting in the green quarter in the center-left of the Figure, following with the next green quarter right of it, then with the next one below it, then with some buildings towards the left, and so on.

The spiral algorithm behaves better when representing the evolution of a project when new elements (*i.e.,* buildings) pop up and disappear in any place in the hierarchy. With rectangle packing, since the allocated space for the whole city is limited to the original outer rectangle, new buildings force all existing buildings to move to make space for them,

[1] https://babiaxr.gitlab.io.

[2] This paper is an extension of our previous work published at VISSOFT 2021 [10].

[3] *BabiaXR*: https://babiaxr.gitlab.io.

[4] https://npmjs.org/package/aframe-babia-components.

[5] *A-Frame*: https://aframe.io.

[6] *Three.js*: https://threejs.org.

not keeping a perceivable relationship with the place they occupied before being moved.

BabiaXR-CodeCity maps the values of software metrics to features of the buildings: each building corresponds to a file and each district to a directory which can have files and subdirectories inside. *BabiaXR-CodeCity* permits to map any metric that is available to either the area of the base, the height, or the color of each building.

BabiaXR-CodeCity scenes are interactive. Users can wander through the city: on-screen with cursor keys and in VR by walking in the physical world. Users can also visualize data about the buildings of their interest. In the on-screen mode, the user can hover the cursor on a building to open a tooltip containing the name of the file together with metric values (*e.g.,* number of functions, lines of code, cyclomatic complexity [11]). In VR the same behavior is enabled by the raycaster offered by the VR controller (*i.e.,* the Oculus Quest 2 pointing mechanism).

2.2. From source code to a 3D scene

Each visualization in *BabiaXR-CodeCity* refers to the source code of a given commit in a *Git* repository. We use a Python script (*cocom_graal2es.py*[7]) to clone the repository, check out a given commit, compute the values of metrics for each file, and store them in an *ElasticSearch*[8] database. The script uses *Graal* [12] and *Perceval* [13] to retrieve and compute the values of the metrics. *Graal*, in turn, uses other tools to compute metrics, via its *CoCom* backend.[9] The *get_list.py*[10] script queries *ElasticSearch* to produce a `JSON` document in the format required by *BabiaXR-CodeCity*. This document has the structure shown in Listing 1.

Listing 1: Structure of a JSON document with data for *BabiaXR-CodeCity*

```
[
  {
    "file_path": "aaa/bbb/ccc"
    "metric": x1,
    "metric2": y1,
    ...
  },
  {
    "file_path": "aaa/ddd"
    "metric": x2,
    "metric2": y2,
    ...
  },
  ...
]
```

To visualize this data, we use an `HTML` document, with the declaration of the `JSON` document to be employed. The `HTML` file imports all dependencies (*A-Frame* and *BabiaXR* JavaScript packages) and defines the scene by declaring the corresponding HTML elements with their attributes (components, in *A-Frame* parlance). Components have their own behavior, programmed in JavaScript, which affects the element for which they are attributes or other components of the same element. Listing 2 summarizes how to define a scene and its relevant components: `babia-queryjson`, which retrieves the `JSON` document and offers it to other components, `babia-treebuilder` which consumes it and generates the tree-like data structure needed by *BabiaXR-CodeCity*, and `babia-boats` is the actual component to generate the visualization, and consumes this data structure. Each component has its own configuration, detailed in the documentation.[11]

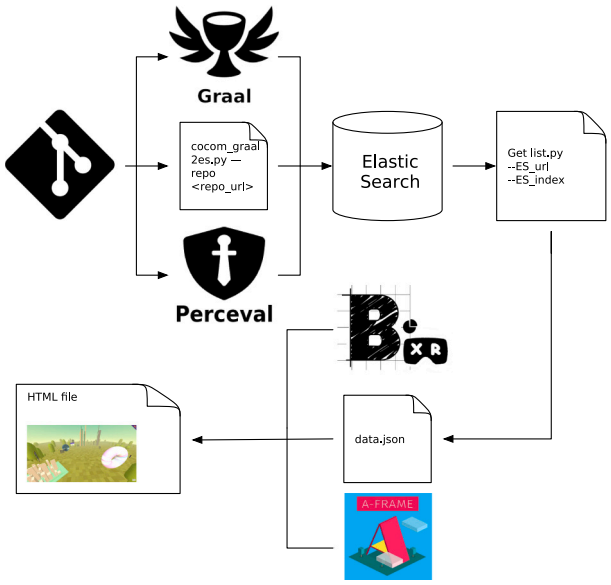


Fig. 3. *BabiaXR* workflow: From source code to a scene.

Listing 2: HTML used for defining a scene

```
<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>
```

Fig. 3 summarizes the complete workflow to produce a scene with *BabiaXR-CodeCity* starting from a source code snapshot in a *Git* repository.

3. Experiments

To compare how users interact with *CodeCity* in VR and on-screen, we conducted two controlled experiments. We reviewed and followed the *ACM SIGSOFT Empirical Standards* [14] for designing the experiments, specifically those related to quantitative methods for experiments with humans, satisfying most of the essential attributes and a part of the desirable ones.

[7] *cocom_graal2es.py* documentation: https://gitlab.com/babiaxr/aframe-babia-components/-/tree/master/tools/generate_repository_data.

[8] *ElasticSearch*: https://www.elastic.co/.

[9] *Graal-CoCom* backend: https://github.com/chaoss/grimoirelab-graal#backends.

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

[11] Documentation for *BabiaXR* components: https://gitlab.com/babiaxr/aframe-babia-components/-/tree/master/docs/APIs.

Fig. 4. A participant during the VR experiment.

Our experiments are inspired by the original experiment performed by the authors of *CodeCity* [15]. Our aim is to understand how accurate (*i.e.,* correctness) and efficient (*i.e.,* time spent) are subjects immersed in VR, compared to subjects working on-screen, when performing the same software comprehension tasks. Since the underlying implementation of *CodeCity* is the same, the metrics used and its mapping to building features are also the same. The way of interacting with the scene is very similar, thus we expect differences to be due to the differences between using screens simulating 3D on 2D, or VR providing the illusion of real 3D. In both experiments, we divided participants into two groups: one group solved the tasks on-screen (*Screen-BabiaXR*) while the other group solved them in VR (*VR-BabiaXR*). All subjects were recruited and did not have any relation with the authors and did not have knowledge of the software systems being visualized during the experiment.

VR participants performed the experiment using the browser in a VR device in immersed VR mode, while on-screen participants performed the experiment using the browser on a desktop computer with a traditional screen. In all cases, tasks were defined by a question related to the comprehension of the software visualized with *CodeCity*. Participants interacted with the visualization and completed the task by providing an answer to the question, which was later evaluated for correctness and time to complete.

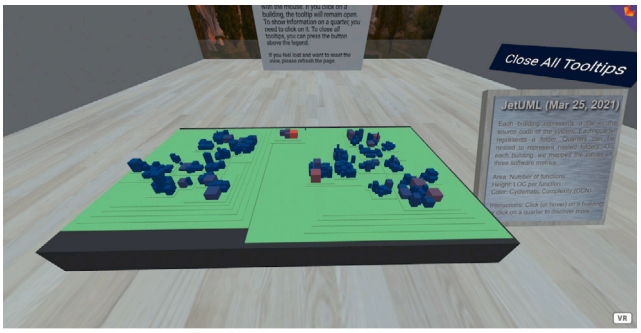
In all cases, the supervisor could see the scene *"with the eyes of the participant"*, and provide support if needed. For on-screen participants, the supervisor could see the screen, and for VR participants the headset was configured to cast the scene to a TV screen.

Fig. 4 shows a participant during the VR experiment and the screencast for the supervisor.

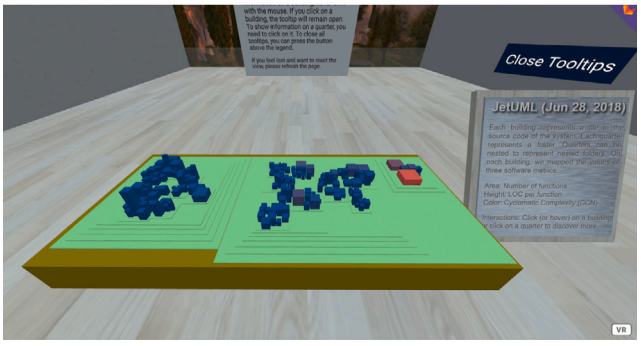
We ran two experiments because when analyzing the results of the first one, and the qualitative feedback provided by participants, we found some aspects of the experiment that could be improved, in order to avoid some potential confounding variables and to get more comparable results. In both experiments, the aim was the same, and the design was very similar. However, some of the details that changed could have their importance, as we will discuss later in Section 4. The participants in both experiments were different, although the demographics of both groups of participants are similar. We conducted the second experiment about six months after the first experiment.

In addition, in the second experiment, various tasks that did not offer significant results were removed while some were added for a more extended analysis. All the details of the changes between the first experiment and the second, both in the experiment scene and in the definition of tasks and methodology are detailed in Section 3.2

Visualized system. In our experiments, participants interact with a *CodeCity* visualization of the source code of *JetUML*.[12] For the first experiment, we use two source code snapshots: the first commit on June 28, 2018 (see Fig. 5(b)), and the first commit of March 25, 2021 (see Fig. 5(a)). However, in the second experiment, we kept only one snapshot of the project, the first commit of March 25, 2021, because we observed during the first experiment that this added some unnecessary confusion to subjects.



(a) *JetUML* as of March 25, 2021 (last commit available when the experiment was defined)



(b) *JetUML* as of June 28, 2018 (Release 2.1)

Fig. 5. First experiment: *CodeCity* visualizations of *JetUML*. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Mapping of metrics. In both of our experiments we use a mapping of metrics to features of the buildings which are similar to those in the original *CodeCity* implementation. Each building represents a file. The area of its base corresponds to the number of functions (`num_funs`), its height to the lines of code per function (`loc_per_function`), and its color the Cyclomatic Complexity Number (`CCN`), in a blue to red scale. Therefore, the volume of the building represents the number of lines of code (`LOC`) of the file.

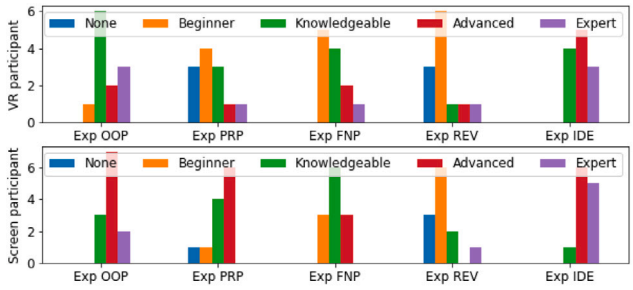
3.1. First experiment

Participants in the experiment were presented with a *BabiaXR-CodeCity* visualization of two different commits of *JetUML*, and were asked to complete a number of software comprehension tasks using the visualization.

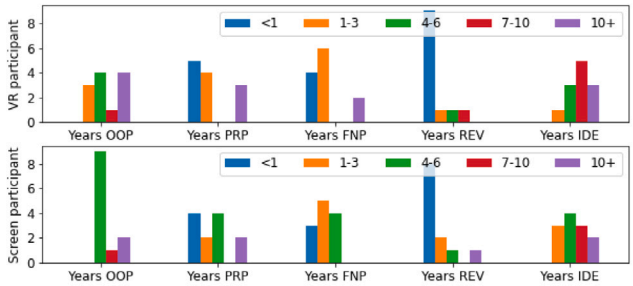
Demographics. The experiment was completed by 24 participants from both academia and industry. Fig. 6 summarizes their experience level and years of experience. None of the participants had ever used a *CodeCity*-like visualization before, even if some of them had heard of it. Only one participant was *"a little"* familiar with *JetUML*, meaning she had heard of the system, but never used it. More details are available in the replication package (see Section 8).

Environment. The data used, and the implementation of the visualizations, were exactly the same for all participants. All participants were presented with two visualizations corresponding to the first commit of June 28, 2018 (see Fig. 5(b)) and if March 25, 2021 (see

[12] *JetUML*: https://github.com/prmr/JetMUL.



(a) Experience level. Each bar shows the number of participants that self-identified their experience as "expert", "advanced", "knowledgeable", "beginner", or "none" in several areas.



(b) Years of experience. Each bar shows the number of participants with a certain range of years of experience in several areas.

Fig. 6. Experience of participants (first experiment). Areas are: object-oriented (Exp OOP), procedural (Exp PRP), or functional (Exp FNP) programming, reverse engineering (Exp REV), and use of IDE (Exp IDE).

Fig. 5(a)). Half of the participants (12) interacted with them immersed in VR (*VR participants*) while the other half (12) interacted with them on a desktop with a traditional 2D screen (*on-screen participants*). The detailed setup for each half is as follows:

VR participants performed the experiment using the Oculus browser in an Oculus Quest 2 headset, in immersed VR mode. Before the experiment, participants were offered the "First Steps" tutorial that comes standard with Oculus Quest devices, to make them familiar with the headset. During the experiments, participants are required to talk aloud when starting each task, and when answering the question posed for determining that a task is completed. The experiments were followed by a supervisor who took note of the answers of the participants.

On-screen participants performed the experiment using the Chrome browser on a desktop computer with a traditional screen. During the experiments, they recorded their answers via Google Forms. The experiments were followed by a supervisor. The form was easy and quick to fill in so that there was little or no additional delay in answering (compared with the talk aloud mechanism used by VR participants.

Tasks. In this first experiment participants were asked to repeat 4 tasks on visualizations for the two commits mentioned earlier: $\mathbf{E_1\ T_1}$–$\mathbf{E_1\ T_4}$ for March 25, 2021 and $\mathbf{E_1\ T_5}$–$\mathbf{E_1\ T_8}$ for June 28, 2018. A ninth task ($\mathbf{E_1\ T_{11}}$, given this number to facilitate consistency with the second experiment) was also proposed to participants, to get some more qualitative information. Table 1 summarizes all tasks that participants had to address. To classify our tasks, we used the maintenance task definition framework by Sillito et al. [16] (the category of each task according to this framework is the last column in the table). Tasks identifiers have the $\mathbf{E_1}$ prefix to make it clear they are of the first experiment.

Answers. To provide answers to questions in each task, on-screen participants had to fill in a Google Form after each task was completed, while VR participants had to speak aloud when they have the final answer for the task. For each task, participants were also asked to assess the level of difficulty, from "strongly disagree" to "strongly agree" with the task being difficult.

Tracking results. For each $\mathbf{E_1\ T_1}$–$\mathbf{E_1\ T_8}$ task, two parameters are tracked as results: how long did it take for the subject to produce the result, as a proxy of efficiency; and the actual answer, as a proxy for correctness:

- **Efficiency.** The supervisor tracked the time that each participant spent on each task. As a backup, the supervisor also recorded a screencast (on-screen) or an actual video recording of the whole experiment (VR).
- **Correctness.** For tasks $\mathbf{E_1\ T_1}$ and $\mathbf{E_1\ T_5}$ participants should identify the geographical location where the test code is located (*e.g.,* the west zone of the city). For tasks $\mathbf{E_1\ T_2}$, $\mathbf{E_1\ T_3}$, $\mathbf{E_1\ T_4}$, $\mathbf{E_1\ T_6}$, $\mathbf{E_1\ T_7}$, and $\mathbf{E_1\ T_8}$ participants had to find the *Top 3 files* with respect to a metric, ordered from highest to lowest in value of that metric. The first author created an oracle with the correct answers to each of the tasks, and an oracle with the *Top 5* correct answers of each task, for close match analysis, these oracles were validated by the second author. With this answer we extract six measurements for the correctness analysis, three of these correspond to whether the three chosen files correspond to those of the oracle exactly in the same order, that is if the first chosen by the participant corresponds to the first of the oracle, the second chosen by the participant corresponds with the second file of the oracle and the third chosen by the participant corresponds with the third file of the oracle. The other three correspond to whether each of the three files chosen by the participants is found in the oracle of the *Top 5* files, regardless of the order of choice, that is, if the first, second, and third chosen files are found in the oracle of 5 files.

In the final task, we asked a few high-level questions (*e.g.,* perceived difficulty, and general feedback to improve the visualization). We also asked an open-ended question, the task $\mathbf{E_1\ T_{11}}$: *"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 gave us qualitative feedback on how our visualization supports users in locating key parts of a system.

3.2. Experimental setup: Analysis and changes

After completing the first experiment, we analyzed the results and the qualitative feedback we obtained from participants. We identified a number of issues that could have an impact on the accuracy of the results.

Measuring task completion time. A potential source of inaccuracy when measuring task completion time was that on-screen participants used a form to fill in the results, while VR participants talked aloud with the supervisor taking notes of the answers. For on-screen participants, the complete description of tasks was in the form, while for VR participants, it was in banners in the scene. Our assumption was that both setups would take approximately the same time, not impacting the completion time, but it could be otherwise.

Training. We had the assumption that on-screen participants did not need a specific training to use the visualization because all of them were used to on-screen interfaces. Their feedback confirmed our hypothesis since nobody reported special issues with the interface except with one minor detail: the use of the arrow/WASD keys for moving in the scene. Several participants were not familiar with that way of navigating the scene, and had some trouble until they adapted, which could cause delays in the completion of tasks, not because of the tasks themselves, but because of the time it took for them to move through the scene to visualize it the way they wanted. This effect could bias the measures of completion time for tasks in the on-screen scenario.

Presentation of the city. Some participants reported problems with how the city was presented. A VR participant reported that *"is hard*

Table 1
Tasks performed in the first experiment.

Task ID	Task description & Purpose	Category
$\mathbf{E_1\ T_1/E_1\ T_5}$	**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 need to understand how the test classes are organized.	Structural Understanding
$\mathbf{E_1\ T_2/E_1\ T_6}$	**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 system 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
$\mathbf{E_1\ T_3/E_1\ T_7}$	**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
$\mathbf{E_1\ T_4/E_1\ T_8}$	**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 time, complex pieces of source code are good candidates for refactoring.	Metric analysis
$\mathbf{E_1\ T_{11}}$	**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 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

to compare different heights and because of the fact that some buildings hide each others"., another one that *"often times it's hard to see some of the entities because they are hidden or have similar metrics".* Some on-screen participants had similar comments: *"it is difficult to find the highest buildings sometimes",* or *"the very even blocks seen from certain sources seem the same or can even lead to mistakes".*

Repeating tasks on a different commit. Our expectation was that repeating tasks on a different commit of the same project would allow us to measure how people behaved when they had some familiarity with the setup. However, the results and feedback showed that the results of that second set of tasks were not really different from the first one, but added some complexity and overall, made the experiment much longer for participants.

Collected demographics. We found that the demographics we collected for participants had room for improvement. On the one hand, they were in some aspects too specific, asking them about their experience in narrow domains, such as object-oriented, functional, or procedural programming, which did not add meaningful information to our analysis. On the other hand, we missed some important aspects such as previous experience with VR devices, which could be interesting to analyze.

Completion time of the last task. Due to an error in our experiment design, in the first experiment we did not track the time to complete for the last task ($\mathbf{E_1\ T_{11}}$).

Starting a task while still reading a task. We realized that VR participants read the description of the tasks while looking toward the city. Although they were told not to do so, that would mean that they could, even unconsciously, start to address the task while still reading its description, not being fair to how we measured time to completion.

3.2.1. Changes

After this analysis, we decided to introduce a number of changes for the second experiment:

To mitigate problems in measuring task completion times we were more precise with the mechanisms for informing participants about the tasks to complete and to measure when they start and finish. All texts explaining tasks, instructions, and information were in the scene itself, for both VR and on-screen participants. All participants were requested to voice they start with a task when they start reading its description. In addition, all participants were required to provide answers by speaking aloud, in addition to notifying the supervisor aloud when they finished the tasks, removing the questionnaire form for on-screen participants.

To ensure a base level of familiarity with the system, we added some training for on-screen participants. Before starting the experiment, all participants were shown a training scene. This scene consisted of a sample scene with a city to let participants get used to the environment and the interaction. With this change, we try to ensure that on-screen participants not familiar with using arrow/WASD keys for moving have the opportunity to become familiar with that.

To facilitate interaction with the city, we changed some presentation parameters. In particular, we changed the scale (making the visualization smaller), the relative height of buildings and neighborhoods (making differences in height more apparent), and the placement of the city in the scene, placing it closer to the camera. We changed nothing in the location algorithm. With this measure, we sought to improve the effectiveness and efficiency of responses, and also minimize bias due to different buildings being hidden by others (the new angle of view allows for a clean sight of all buildings).

To make the experiment shorter, we removed the second set of tasks on a different commit (*e.g.,* $\mathbf{E_1\ T_5}$). Instead, we added two new tasks which explored different aspects of software comprehension.

To improve the demographic data collected, we merged all questions about programming experience in one and added a new question for VR participants, about their previous experience with virtual reality.

We also measured the completion time for the last task (*e.g.,* $\mathbf{E_2\ T_{11}}$).

To avoid the problem of participants starting with a task while still reading about it, in the second experiment we placed the descriptions of the tasks on one side and the city on the other side, with the participant in the middle. Therefore, when they are done reading the task, they can turn, have the city in sight, and start with the task.

3.3. Second experiment

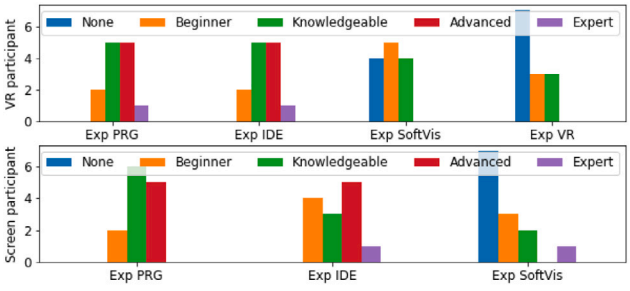
Participants in the experiment were presented with a *BabiaXR-CodeCity* visualization of a single commit of *JetUML*, and were asked to complete a number of software comprehension tasks using the visualization. None of the participants had participated in the first experiment.

Demographics. This experiment involved 26 participants from both academia and industry. Fig. 7 summarizes experience level and years of experience. None of the participants had ever used a *CodeCity*-like visualization before, even if some of them had heard of it. Only one of the participants had heard about *JetUML*, the others had no knowledge of the system. More details about the demographics are available in the Replication Package (Section 8).

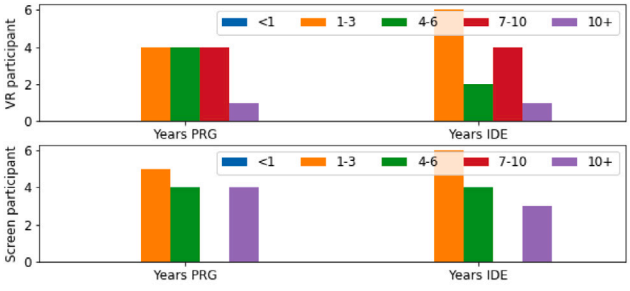
Environment. The data used, and the implementation of the visualizations was exactly the same for all participants. All participants were presented with a single visualization of the *JetUML* source code,

Table 2
Tasks performed in the second experiment. New tasks ($\mathbf{E_2\ T_9}$ and $\mathbf{E_2\ T_{10}}$) are highlighted in red.

Task	Task description & Purpose	Category
$\mathbf{E_2\ T_1}$	**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 need to understand how the test classes are organized.	Structural Understanding
$\mathbf{E_2\ T_2}$	**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 system 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
$\mathbf{E_2\ T_3}$	**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
$\mathbf{E_2\ T_4}$	**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 time, complex pieces of source code are good candidates for refactoring.	Metric analysis
$\mathbf{E_2\ T_9}$	**Description.** The base/area of the buildings represents the number of functions in the code file (num_funs), and the height represents the lines of code per function in the code file. Therefore, the volume (base x height) corresponds to the total number of lines of code. Find the three source code files (not testing files) with the highest lines of code in the system. Say their names in order. **Purpose.** Lines of code, as lines of code per function, is a good indicator to identify classes with very large files, hence good candidates for refactoring, maintenance activities, or quality assurance. Moreover, the whole volume is another metric that can be analyzed by the tool	Metric analysis & Tool insight
$\mathbf{E_2\ T_{10}}$	**Description.** Find the source code file (not testing files) with the highest lines of code per function (highest loc_per_function) inside the `src/ca/mcgill/cs/jetuml/diagram/nodes` folder. **Purpose.** Finding a specific file in a folder has the purpose of verifying if the tool provides the necessary information to search for a specific file that is difficult to locate in a general view.	Code location
$\mathbf{E_2\ T_{11}}$	**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 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



(a) Experience level. Each bar shows the number of participants that self-identified their experience as "expert", "advanced", "knowledgeable", "beginner", or "none" in several areas.



(b) Years of experience. Each bar shows the number of participants with a certain range of years of experience in several areas.

Fig. 7. Experience of participants (second experiment). Areas are: programming (Exp PRG), use of IDE (Exp IDE), software visualization (Exp SoftVis), and virtual reality (Exp VR).

corresponding to the first commit of March 25, 2021 (see Fig. 5(a)), Half of the participants (13) interacted with them immersed in VR (*VR participants*) while the other half (13) interacted with them in a desktop with a traditional 2D screen (*on-screen participants*). The detailed setup for each half is as follows:

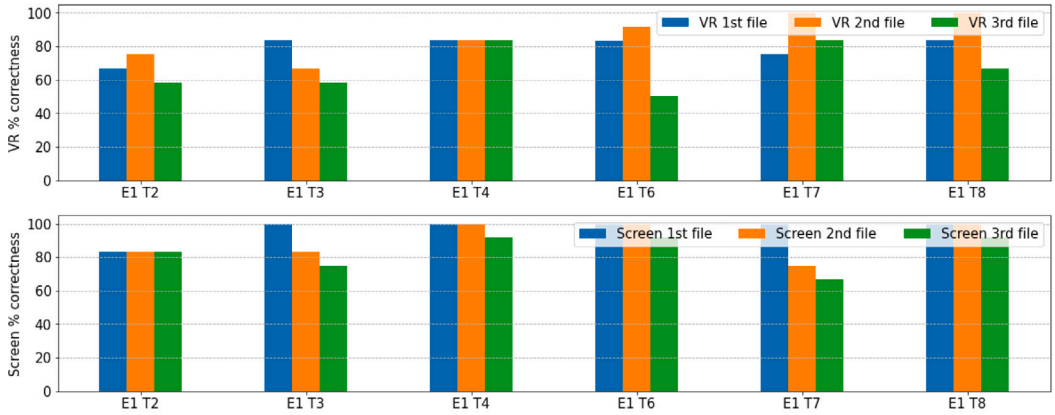
- **VR participants** performed the experiment using the Oculus browser in an Oculus Quest 2 headset, in immersed VR mode.
- **On-screen participants** performed the experiment using the Chrome browser on a desktop computer with a traditional screen.

In both cases, during the experiments participants are required to talk aloud when starting each task, and when answering the question posed for determining that a task is completed. The experiments were followed by a supervisor who took note of the answers of the participants. Before their experiment, participants were invited to enter a training scene in the same environment (VR or on-screen) of their experiment, to become as much familiar with it as reasonably possible. The training scene was a *BabiaXR-CodeCity* visualization with random data, and instructions about how to move around and interact with the visualization. During the experiment, all participants received the description of the tasks by means of banners in the scene.

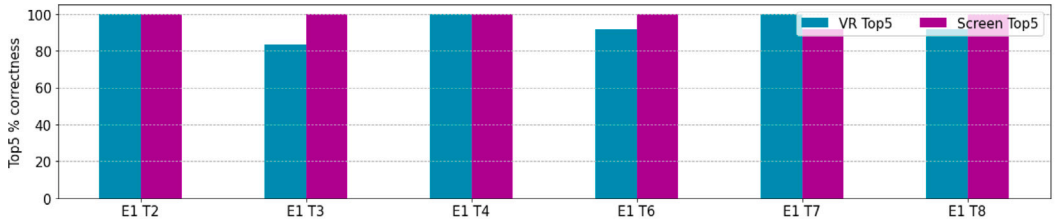
Tasks. In this second experiment, participants were asked to complete 7 tasks, summarized in Table 2. With respect to the first experiment, there are two new tasks ($\mathbf{E_2\ T_9}$ and $\mathbf{E_2\ T_{10}}$) that are highlighted in red in the table. The task identifiers have the $\mathbf{E_2}$ prefix to make it clear that they are of the second experiment. Tasks $\mathbf{E_2\ T_9}$–$\mathbf{E_2\ T_{11}}$ are numbered this way to facilitate consistency with the first experiment (*i.e.,* task $\mathbf{E_2\ T_{11}}$ is the same as $\mathbf{E_1\ T_{11}}$ in the first experiment).

Answers. To provide answers to questions in each task, all participants had to speak aloud when they have the final answer for the task. For each task, participants were also asked to assess the level of difficulty, from "strongly disagree" to "strongly agree" with the task being difficult.

Tracking results. For each task in $\mathbf{E_2\ T_1}$–$\mathbf{E_2\ T_{10}}$ the same parameters were tracked as in the first experiment: how long did it take for the subject to produce the result (as a proxy of efficiency), and the actual answer (as a proxy for correctness). Additionally, we also tracked time to completion for $\mathbf{E_2\ T_{11}}$.



(a) Exact match: correctness percentage for each file (top: VR, bottom: on-screen)



(b) Close match: correctness percentage for all files

Fig. 8. First experiment: Correctness percentage for tasks asking for top 3 files according to a certain metric.

4. Results

This section summarizes the results of the two experiments in terms of correctness (see Section 4.1) and completion time (Section 4.2).

4.1. Correctness ($\mathbf{RQ_1}$)

For answering $\mathbf{RQ_1}$ (How does the *accuracy* of participants immersed in VR compare to that of participants using the on-screen version of *BabiaXR-CodeCity*?) we analyzed how correct were the answers that participants in the experiment provided for $\mathbf{E_1\ T_1}$–$\mathbf{E_1\ T_8}$ (in the first experiment) and $\mathbf{E_2\ T_1}$–$\mathbf{E_2\ T_4}$, $\mathbf{E_2\ T_9}$, and $\mathbf{E_2\ T_{10}}$ (second experiment):

- $\mathbf{E_1\ T_1}$, $\mathbf{E_1\ T_5}$ and $\mathbf{E_2\ T_1}$ asked to identify the part of the city that corresponded to test code. These tasks were evaluated by checking if the participant identified the correct geographical part of the city. We defined the correctness percentage for these tasks as the percentage of participants whose answer was correct.
- All other tasks asked for a top 3 list of files, according to some metric. They were evaluated for exact match and for close match:
  - For exact match, each of the files answered was compared with the corresponding position in the top 3 of the oracle. The result was considered "correct" if the answer was exact (*e.g.,* same position in the oracle). We defined correctness percentage for each of the three files in each answer as the percentage of participants whose answer was correct for that file.
  - For close match, each answer was checked against the top 5 files in the oracle, thus allowing for some error. The result was considered "correct" if that file was in the top five of the oracle. We defined correctness percentage as the percentage of files in the answers that were correct.

First experiment ($\mathbf{E_1}$)

Task $\mathbf{E_1\ T_1}$ and task $\mathbf{E_1\ T_5}$ were correctly answered by all participants (both VR and on-screen), who were able to identify the zone of the city corresponding to test code. This is a clear indication that users had no difficulty in identifying a block of code (*e.g.,* the `test` directory).

Fig. 8 summarizes the percentages of correctness for exact match and for close match for tasks $\mathbf{E_1\ T_2}$–$\mathbf{E_1\ T_4}$ and $\mathbf{E_1\ T_6}$–$\mathbf{E_1\ T_8}$. For exact match, Fig. 8(a) shows the correctness percentage for each of the three files in the answers to the corresponding task. In general, results for on-screen are better than for VR, by a margin of about 20%, but still results for VR are well over 60% in most cases. For close match, Fig. 8(b) shows the correctness percentage for all files in the answers, split by VR or on-screen participants. In this case, the results are very similar for both VR and on-screen participants.

These results show that VR participants have consistently answered with lower accuracy when compared to on-screen participants. As expected, the results are better for both types of participants when comparing their answers to the *Top 5 files*, since a certain margin of error is allowed.

Despite the different results, for all 8 tasks both VR and on-screen participants provided similar answers with respect to the difficulty of the tasks (see Fig. 9).

However, more VR participants found some tasks more difficult than on-screen participants ("agree" with "did you find the task difficult?"). Checking the qualitative feedback obtained after the experiment, it seems the main reason for this difference was the lack of familiarity with the VR headset and its interaction possibilities.

Second experiment ($\mathbf{E_2}$)

$\mathbf{E_2\ T_1}$ was correctly answered by all participants (both VR and on-screen), who were able to locate the zone of the city where test code is displayed. This result, consistent with the first experiment, reinforces the fact that participants can easily locate a block of code.

For tasks $\mathbf{E_2\ T_2}$–$\mathbf{E_2\ T_4}$, $\mathbf{E_2\ T_9}$, and $\mathbf{E_2\ T_{10}}$, Fig. 10 details the percentages of correctness for exact match and for close match. For

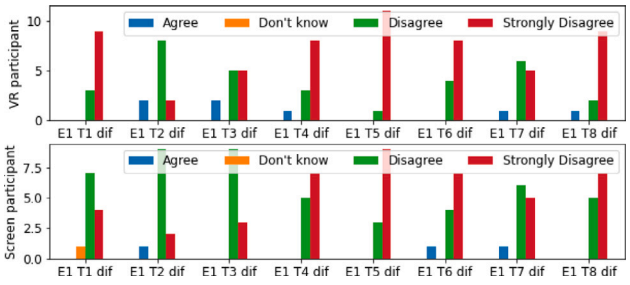


Fig. 9. First experiment: answers to "Did You Find the Task Difficult?".

exact match, Fig. 10(a) shows the correctness percentage for each of the three files in the answers to the corresponding task. For four tasks ($\mathbf{E_2\ T_2}$, $\mathbf{E_2\ T_4}$, $\mathbf{E_2\ T_9}$ and $\mathbf{E_2\ T_{10}}$), results are better for VR participants. For task $\mathbf{E_2\ T_3}$, results are mixed. These results are in general much better for VR participants (when compared to on-screen participants) than in the first experiment.

For close match, Fig. 10(b) shows the correctness percentage for all files in the answers, split by VR or on-screen participants. In this case, the results are again very similar for both VR and on-screen participants, in line with the corresponding results of the first experiment.

In the first new task, $\mathbf{E_2\ T_9}$, we asked participants to identify the buildings with the larger volume. It is interesting to observe that its results are significantly better for VR participants, which could suggest that volume is better appreciated in VR.

In Task $\mathbf{E_2\ T_{10}}$, instead, several buildings have very similar heights. That explains the poor results both for VR and on-screen participants. However, the slightly better results for VR participants could suggest that small differences in height can be better appreciated in VR.

Contrary to the first experiment, the new results show that VR participants have in general answered with better accuracy than on-screen participants. This could be due to the adjustments that we performed. Once again, despite the different results, both VR and on-screen participants provided similar answers with respect to the difficulty of the tasks (see Fig. 11). However, some more VR participants found some tasks difficult. We attribute that to the lack of familiarity with the VR headset and its interactions, as we did for the first experiment.

4.2. Completion time ($\mathbf{RQ_2}$)

For answering $\mathbf{RQ_2}$ (How does the *efficiency* of participants immersed in VR compare to that of participants using the on-screen version of *BabiaXR-CodeCity*?) we analyzed the completion time for the tasks $\mathbf{E_1\ T_1}$–$\mathbf{E_1\ T_8}$ (*i.e.,* first experiment) and $\mathbf{E_2\ T_1}$–$\mathbf{E_2\ T_4}$, and $\mathbf{E_2\ T_9}$–$\mathbf{E_2\ T_{11}}$ (*i.e.,* second experiment). Table 3 summarizes the results of this analysis, which are also presented as box plots in Figs. 12 and 13. The difference in completion time between VR participants and on-screen participants is significant.

Across all tasks, VR participants were faster than on-screen participants. On average, they were 1 min and 12 s faster in the first experiment and almost 1 min in the second one. This difference reaches its peak in the first experiment in task $\mathbf{E_1\ T_1}$ where VR participants were on average 2 min and 10 s faster than on-screen participants to identify the `test` directory. In the second experiment, the peak is in task $\mathbf{E_2\ T_2}$ where *VR-BabiaXR* participants were 1 min and 37 s faster.

To understand whether these differences are statistically significant, we ran the *Mann–Whitney U* Test [17], a non-parametric unpaired test suitable to compare differences between two independent groups (VR and on-screen participants). Columns *U* and *p-value* in Table 3 report the results: across all tasks, differences in task completion times between VR and on-screen participants 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 [18] for the magnitude,

Table 3
Completion time for all tasks, and statistical analysis (Mann–Whitney U Test and Cliff's Delta effect size).

ID	VR	Screen (S)	Mann–Whitney U Test		Cliff's
	(m:ss)	(m:ss)	U	p-value	Delta
$\mathbf{E_1\ T_1}$	1:47	3:57	128.0	≈0.001	0.77
$\mathbf{E_1\ T_2}$	1:50	3:46	129.0	≈0.001	0.79
$\mathbf{E_1\ T_3}$	1:42	3:00	137.0	<0.001	0.75
$\mathbf{E_1\ T_4}$	0:57	1:38	120.0	≈0.006	0.67
$\mathbf{E_1\ T_5}$	0:16	1:00	144.0	<0.001	1.00
$\mathbf{E_1\ T_6}$	1:01	1:56	138.0	<0.001	1.00
$\mathbf{E_1\ T_7}$	1:05	2:11	128.5	≈0.001	0.78
$\mathbf{E_1\ T_8}$	0:28	1:12	144.0	<0.001	1.00
$\mathbf{E_1}$ All	**1:08**	**2:20**	**7531.0**	**<0.001**	**0.63**
$\mathbf{E_2\ T_1}$	0:47	2:13	156.0	<0.001	1.00
$\mathbf{E_2\ T_2}$	1:24	3:01	162.0	<0.001	1.00
$\mathbf{E_2\ T_3}$	1:37	2:05	124.0	0.045	0.78
$\mathbf{E_2\ T_4}$	1:35	2:18	127.0	0.031	1.00
$\mathbf{E_2\ T_9}$	0:41	1:03	142.5	≈0.002	1.00
$\mathbf{E_2\ T_{10}}$	1:49	2:31	144.0	≈0.003	1.00
$\mathbf{E_2\ T_{11}}$	1:58	2:46	126.0	0.035	1.00
$\mathbf{E_2}$ All	**1:24**	**2:16**	**6429.5**	**<0.001**	**0.63**

and the corresponding range of *Cliff's Delta* following the suggestion by Romano et al. [19]. Our results show that across all tasks the effect size is large ($\delta > 0.474$).

4.3. Final task & feedback

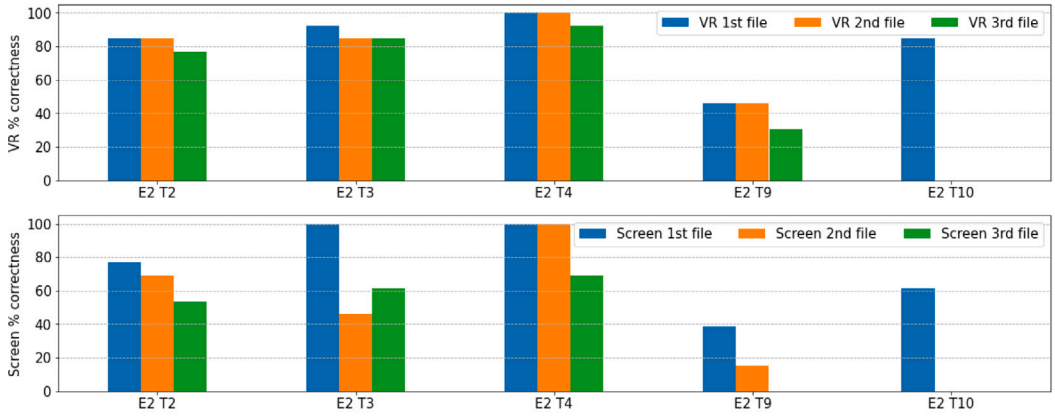
At the end of the experiment, we asked participants to perform a high-level task ($\mathbf{E_1\ T_{11}}$ in the first experiment, $\mathbf{E_2\ T_{11}}$ in the second): *to locate the "core" of JetUML*. All participants provided a reasonable answer (*e.g.,* the `src/ca/mcgill/cs/jetuml` directory) and reasoned about why that was their answer. 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). In any case, we did not observe remarkable differences between VR and on-screen participants in any of the two experiments. In the second experiment, where we measured completion time of $\mathbf{E_2\ T_{11}}$, we observed that VR participants were faster than on-screen participants, also in this deliberately vague task.

At the end of the experiment, we asked participants to provide us with some feedback on the experiment itself. All participants agreed that the experiment was not too difficult. We also collected some qualitative feedback that we leveraged to improve the second experiment.

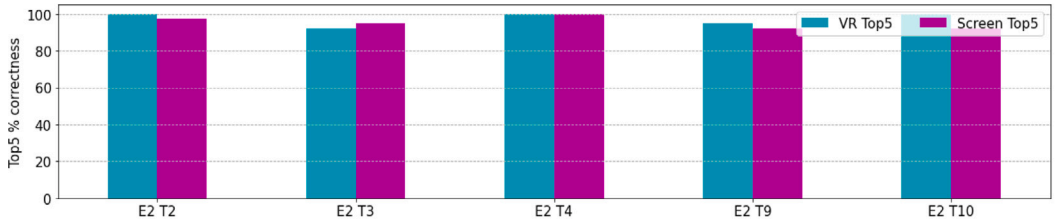
4.3.1. Feedback on the first experiment

On-screen participants agreed that the experiment was not difficult. Some of them even agreed that was easy (*e.g., "Not at all. For me, it was easier to identify different buildings per color", "No, it was quite easy", "This experiment is not difficult once you read the instructions", "Overall it wasn't difficult"*). Some of them detailed that some parts of the experiment were difficult to address, *e.g.,* differentiate geometric features, *"In the heights, it was a bit difficult for me to differentiate", "The other factor that made the experiment more complicated than it could be are the colors, the difference is not enough to make a distinction in some cases, so I needed to use the tooltips", "With respect to the experiment itself, perhaps the most challenging was to differentiate the metrics that are represented by the building"*, and *"And it is difficult to find the highest buildings sometimes"* are comments that represent the problem. This led us to improve the scene in the second experiment, regarding the differences between the characteristics of the building (*i.e.,* height, area, and color).

On-screen participants also reported issues with the user interface of our tool when it comes to movements, interactions, and camera. We received feedback like *"I would have liked to be able to adjust the scrolling*



(a) Exact match: correctness percentage for each file (top: VR, bottom: on-screen)



(b) Close match: correctness percentage for all files

Fig. 10. Second experiment: Correctness percentage for tasks asking for top 3 files according to a certain metric.

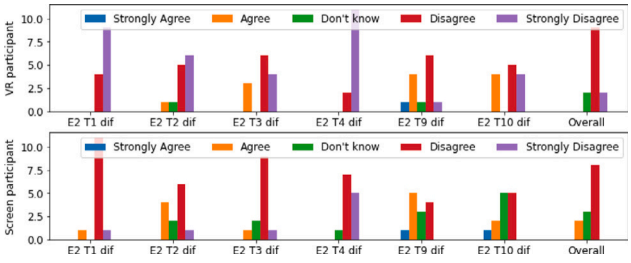


Fig. 11. Second experiment: answers to "Did You Find the Task Difficult?".

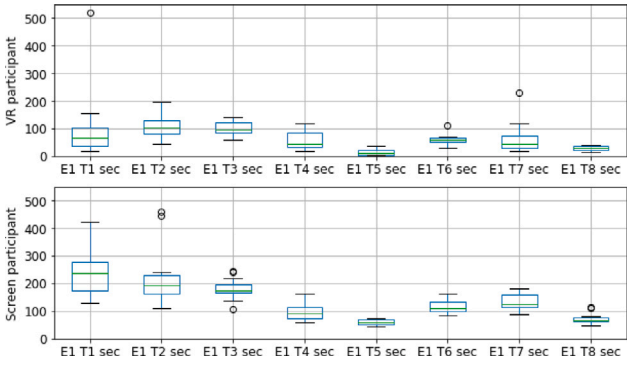


Fig. 12. First experiment: completion times (in seconds). VR participants (top), on-screen participants (bottom).

sensitivity, as there were times when it was difficult for me to position myself in a certain position/height for a better view of the city", "WASD controls are very sensitive", "The trickiest part was getting used to the controls as they are not as precise as they could be, so finding the right positioning to correctly perceive the details was a bit difficult and I spent some time finding out ways of moving the camera around properly". To mitigate

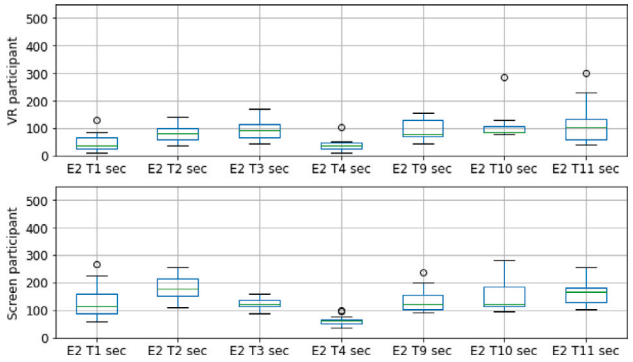


Fig. 13. Second experiment: completion times (in seconds). VR participants (top), on-screen participants (bottom).

these problems, in the second experiment we added a training scene for on-screen participants to familiarize themselves with the movements.

VR participants also agreed that the experiment was not difficult. Some of them agreed with on-screen participants that it is difficult to distinguish between heights (*e.g., "it is hard to compare different heights and because of the fact that some buildings hide each other", "oftentimes it's hard to see some of the entities because they are hidden or have similar metrics"*). We took this problem into account when refining the setup for the second experiment. Only one participant reported that being not familiar with VR was a difficulty for realizing the experiment (*"I was not familiar with VR so it was difficult for me to point to things and hard to read because of glasses"*).

VR participants also provided some suggestions: *"Raise the floor level, use colors also for platforms (i.e., to identify the complexity)", "Buildings could be taller (or ground level could be lifted)",* or *"You could add means to automatically highlight the top 3 source code files matching a particular filter, say the top 3 files ordered by complexity".* We tried to address some of them in the second experiment.

In Section 3.2 we detailed the problems that were identified, and the changes we made to the design of the second experiment to address them.

4.3.2. Feedback on the second experiment

In this second experiment, two on-screen participants agreed that the experiment was difficult. One of them gave us the reason, *"It was difficult because I'm not used to 3D environments and video games, so the use of keyboard and mouse is a bit tricky for me. So I would improve movement"*. This is something that we tried to mitigate by adding the training before the experiment, but apparently, for this participant, the training was not enough. In general, compared with the first experiment, on-screen participants had fewer problems in distinguishing between buildings with similar features. Only one of them reported a problem: *"I would improve the height scale to notice small changes"*. The inclusion of the task about the volume was specifically difficult for some on-screen participants (*"The volume is the most difficult task"*, *"The task about the volume doesn't contribute much"*). On-screen participants still reported problems with the interface and the movements: *"Movement issues, hard to put the camera in a specific point (to see the heights for instance). Task 7 is difficult to know"*, *"It is hard to move the camera with the mouse and keyboard"* and *"The interface is fuzzy when you open more than one-quarter tooltip"*.

Some on-screen participants also provided positive feedback about the tool and scenario, *"The system is clearly represented, with its modules"* and *"The building image looks really useful, the color was really useful, the height was helpful"*.

On the other hand, all VR participants reported that the experiment was not difficult, but three of them reported that the new task related to the volume was hard: *"Show the value for the volume, it's tough to compute the volume by hand"*, *"The volume was difficult because I would multiply the values and the differences are not enough"*, and *"It's hard to estimate the volume"* are the comments. We wanted to add this task due to its difficulty level forcing the user to perceive volume without providing a number. After the improvements thanks to the feedback of the first experiment, there were some VR participants that pointed out other issues to improve regarding the scene: *"I suggest raise the city level"*, *"It's hard to point to very thin entities (e.g., Node.java)"*, and *"Some buildings are hard to point, the thinnest"* are examples of these troubles.

One participant had trouble with the VR glasses: *"Oftentimes the view is blurry, but I believe it's because I wear glasses"*. VR participants also provided positive feedback, *"I like the fact that you can crouch to see the heights of buildings"* shows a good point about VR immersion, and *"The more you do tasks, the easier is. It is easy to understand"*.

5. Discussion

Our results show that VR participants were in general less correct in their answers than on-screen participants in almost all the tasks of the first experiment. However, in the second experiment, the VR participants improved their results, being more correct than on-screen participants in several tasks. This effect is more clear in tasks that require finding buildings with small variations among the largest ones. For example, in the task of finding a specific building in a specific neighborhood with very little differences between the buildings ($\mathbf{E_2}$ $\mathbf{T_{10}}$) VR participants have better accuracy. The difference is not high, but it is noticeable.

There are a number of reasons that might explain the improvements in the second experiment. One of the main reasons could be the training we included in the second experiment both for on-screen and for VR subjects, which we introduced with the aim to balance both environments. In the first experiment, half of the VR participants had never used a VR headset prior to our experiment. But using a VR headset is an experience that in the beginning can be quite disorienting. Adding a dedicated training session gives the subject the opportunity to get familiar with VR immersion, the use of the headset and the controls, and their effect on movement. This could explain that subjects in VR are more accurate in the second experiment. The training would have a smaller effect on on-screen participants since they would already be familiar with the on-screen environment, and the adaptation is easier since for using on-screen technologies the use of keyboard and mouse is much more widespread (although they would need to adapt to using the arrow/WASD keys for movement). Training would have little or no effect in improving their accuracy when performing the tasks.

Indeed, if we observe how the tasks were solved throughout the experiment in terms of efficiency and correctness, VR participants seem to get better task after task, especially in the first experiment. 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, it remains to be investigated whether this is actually the case.

Following the feedback of the participants in the first experiment, as we described in Section 3.2, we also introduced some changes in the scale and layout of the software city in VR, without changing the layout algorithm. This could also contribute to the improvements that can be observed in the comparison of the results of the two experiments.

However, as a general note, it is important to realize that the use of VR devices is still not widespread, as we have seen in the demographic data for both of our experiments: participants had little or no previous experience with VR immersion. This means that participants in the experiment started with a certain disadvantage since they had to learn to work in VR immersion at the same time that they completed the tasks in our experiments. It remains to be investigated whether, as more people become accustomed to VR devices, the correctness of people using a VR improves.

When we focus on completion time, the picture is completely different. Despite lacking experience in VR headset usage, VR participants were considerably faster than on-screen participants across all tasks. As discussed in Section 4, the difference across all tasks is statistically significant. In some of the tasks, VR participants completed them in less than half the time of on-screen participants. We obtained similar results also on the second experiment, which provide more evidence about program comprehension tasks being indeed performed faster in VR.

This result suggests that VR immersion could play a pivotal role when carrying out software comprehension tasks in a 3D environment. Interacting with a VR headset is similar to interacting in the real world, using more natural gestures than when working on-screen. You can move your head to look around, kneel down to see the world from a lower perspective, or just move around to get a better angle of vision. Besides, not everybody is used to navigating 3D environments on-screen with arrow/WASD keys and a mouse. Even with the training inserted in the second experiment, which slightly improved the results, 3D on-screen interactions are more cumbersome, less natural, and, as suggested by our results, less effective when compared with VR immersion.

For putting all the results in context, it is also important to notice that, despite our best efforts and intentions, *BabiaXR-CodeCity* is still a prototype, not yet fully optimized for VR. We think there is still a lot of room for improvement when it comes to a number of concerns that pertain exclusively to VR, such as:

Navigation. Currently, navigation in *BabiaXR-CodeCity* is based on walking around (which needs real space, and is usually not appropriate in many environments), or based on gaze (to select the orientation of the camera) and a single-hand controller (to gather additional information on entities). Many VR applications, especially games, use both hand controllers in conjunction with gaze and walking around, allowing for more natural and complex movement options. Also, there are different types of *artificial VR locomotion*[13] that we did not yet

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

explore. This means there is still untapped potential in making our 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 (*i.e.,* a camera) 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 Fig. 5(a), 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 as even smaller, or much larger, than a real-scale city. Which representation is more convenient and efficient? Would allowing users to change the scale 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 by tuning this factor appropriately. In the second experiment, we improved this physical size for the VR environment, making it more accessible while immersed in VR, which maybe was one of the reasons for the gains in correctness.

Interaction. In the current version, interactions with the visualization in *BabiaXR-CodeCity* are very limited. Users can just point at buildings and neighborhoods, and see banners with extended information. Other interactions could be used to improve the user experience. For example, in a small-scale city, users could "touch" buildings, getting extended information that way. Or they could just rotate the whole city, and enlarge and shrink it as they want to see details or the whole picture.

In summary, we consider that the results of our experiments show that VR *CodeCity* is comparable to on-screen *CodeCity*, with respect to the correctness 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. As VR devices improve, as people get more accustomed to using them, and as software for building applications for them becomes more capable of producing good interactions and user experiences, we expect that metaphors such as *CodeCity* get increasingly more useful and usable.

We also have shown how our multi-platform implementation of *CodeCity*, which can work very similarly both on-screen and in VR, can help to fairly compare those environments for metaphors used for software comprehension. In fact, it is a lucky chance that the implementation in the browser makes it so easy to compare the same scene, and essentially the same app, in both on-screen and VR environments: only a handful of technologies offer this opportunity. Since most of the details of running on different platforms are dealt with by the browser, we could focus on building a high level application implementing the *CodeCity* metaphor, and not dealing with the details of how to render the scene, or how to code the interaction of the user with it.

Finally, the availability of the application in the browser also simplifies the experiments. No specific hardware needs to be deployed, since it is easy to find devices (both screen-based and VR-based) with a browser supporting the needed standards. Users can just launch a browser, point it to a given URL, and start running the experiment. We expect that this approach may also help, in the future, to have experiments with a much larger number of subjects, since in principle, anyone with access to on-screen or VR browsers could volunteer to be one of them. This even opens the door to semi-automated experiments, where subjects select when and where they want to run the experiment, and everything is performed and recorded for future examination, online. From this point of view, our second experiment was much more automated than the first one, and we feel there is still a lot of room for improvement in this area, with relatively little effort. However, ensuring that the conditions in which the experiment is performed, and minimizing the effect of confounding variables, will be a challenge if automation is pushed further. And of course, it will depend on the fraction of potential participants with access to VR devices.

6. Threats to validity

6.1. Internal validity

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

Subjects. In both experiments, we ensured that all the participants had experience in different relevant topics about programming using a questionnaire, mitigating 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 (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 or on-screen participants. 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.

Lack of training. We had two problems with respect to a potential lack of training, which could lead to incorrect results, compared to the same experiment with adequate training. In the case of VR participants, they had little or no previous experience with VR immersion, and with the VR devices we were using. To mitigate this problem, we offered all of them to run the generic starter tutorial *First Steps* of the Oculus Quest 2 headset. But then we had also a potential lack of training in the use of *BabiaXR* or *CodeCity*. In both cases (VR and on-screen participants) the text to be followed for performing the tasks explains how the tool is used and how the interaction with the elements works. But some training could be convenient. To mitigate this threat, in the second experiment we provided the participants with a training scene to familiarize themselves with the environment. For on-screen participants, this training scene included a tutorial about the movements and interaction controls, forcing the participants to get used to the arrow/WASD keys for moving and mouse clicking for moving the camera.

6.2. 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 experiments is relatively small. Although we tried to recruit a representative sample and we repeated the experiment, a larger sample would lead to 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 two (one for each experiment) 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 also 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.,* [15,20]). Even if we tried to mitigate this threat extensively,

we cannot exclude all possible influences on the results of the two experiments.

Time measurement. In the first experiment, to mitigate the threat that task completion times are not exact, 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 between 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 did that in the second experiment, forcing all the participants (*VR-BabiaXR* and *Screen-BabiaXR*) to speak aloud, standardizing the time spent reporting answers (*i.e.,* with respect to the first experiment where *Screen-BabiaXR* participants had to fill an external form to report the answers).

Task texts. To mitigate the threat that a participant could start performing a task while still reading the task description (*i.e.,* with the city is in front of her, as in the first experiment), in the second experiment we placed the task descriptions in a different position (*i.e.,* rotated 90 °CW with respect to the city), forcing *VR-BabiaXR* participants to move the head to read the assignments and *VR-BabiaXR* participants to move the camera to read the assignments, in both cases losing sight of the city.

On-Screen and VR movement. The differences in the moving interface presented by *Screen-BabiaXR* and *VR-BabiaXR* (*i.e.,* sensitivity, speed, etc.) can pose a threat to the validity of results, specifically in what refers to completion time. *Screen-BabiaXR* participants from the first experiment reported that they experienced some sensitivity issues when handling the camera (moving around in the scene). To mitigate this threat, we added a training for the second experiment with no time limit for both *VR-BabiaXR* and *Screen-BabiaXR*, for participants to get used to the environment, movements, and interaction. After the second experiment, most participants did not report this sensitivity issue, but some of them still reported movement problems, indicating that an improvement to the *Screen-BabiaXR* interface should be made to fully mitigate this threat.

7. Previous and 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 [21]. The metaphor used depends directly on the software artifacts to represent [22] 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 [23] for visualizing the structure of large systems, and the solar system metaphor [24] for visualization of object-oriented software systems. Several years later, *CodeTrees* for visualizing software as a collection of trees [25], a concept which was later extended [26].

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. [27], rooted in the architectural pattern language proposed by Alexander et al. [28].

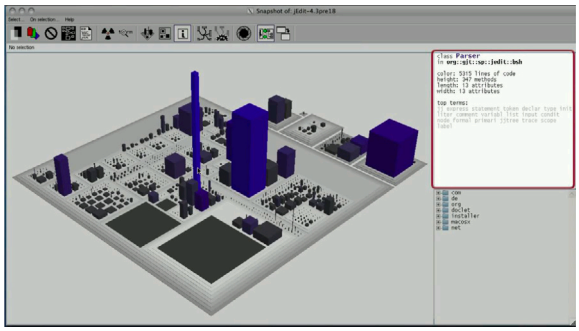


Fig. 14. The original *CodeCity* tool.

7.1. Codecity

The first implementation of the city metaphor was *Software World* [1], which visualized software systems as buildings in a city. Afterward, several approaches were explored to support developers in maintaining software systems as fulfilling program comprehension tasks. Panas et al. [29–31] presented a software city showing information about static and dynamic data, and Marcus et al. [32] a city-like software visualization. *Verso* [33] was based on landscapes, but with the influence of the city metaphor.

In 2007 *CodeCity* was presented [6], raising the approach to a new level implementation-wise. Fig. 14 shows the original *CodeCity* application, including the interface for interacting with the city and the metrics details.

CodeCity showed that it could not only be used for program comprehension [6], but also for software evolution analysis [34] and design problem analysis [35].

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. [36] 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. [37] 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* [38], based on the Minecraft game engine. *M3tricity* [39], a recent re-implementation of *CodeCity* by the original research group, is a web application to visualize software systems as evolving cities that treats evolution as a first-class concept.

7.2. From 3D to virtual reality

One of the early explorations of using VR for visualizing software was done by Young and Munro [40], at the time quite a technical feat. Another early VR-based approach is *Imsovision* [41], 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* [42], based on the first versions of *WebVR*, focusing on the runtime and static characteristics of object-oriented programming software systems. Vincur et al. [4,43] proposed a VR city for analyzing object-oriented software. Steinbrückner and Lewerentz [5] proposed stable city layouts for evolving software systems, using layouts other than treemaps. *Getaviz* [44] also uses the city metaphor to generate structural, behavioral, and evolutionary views of software systems for empirical evaluation.

CityVR [45], developed with *Unity3D*, provides the same metrics as the original *CodeCity*, adding interactions using the gaze of the user in the VR headset, and its controllers. The technique we used for moving in the scene technique has similarities with their approach.

Capece et al. [46] visualized Java systems with the *Unreal Engine 4* using the city metaphor in VR. Most of the current approaches are based on non-web technologies (*i.e.,* Unreal Engine or Unity), which differentiates them from our approach, since *BabiaXR* is web-based, allowing the visualization of the city in any modern browser.

Other metaphors have been implemented in VR. Misiak, Schreiber et al. [47] proposed the island metaphor for visualizing OSGi-based software systems, introducing and emphasizing the visualization of dependencies. Schreiber et al. [48] 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.

7.3. Validation experiments

Wettel et al. [15] 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 a great deal after it. Merino et al. [49] conducted a controlled experiment using 3D visualization of the city metaphor using a computer screen, an immersive 3D environment, and a 3D printed model. The authors found that the on-screen participants perceived the least difficulty to identify outliers, and in terms of completion time, our results are in line, showing that the VR participants were faster in resolving the program comprehension tasks. Rüdel et al. [50] conducted a controlled experiment with 20 participants of the city metaphor but using a different algorithm for the layout, they compared as well VR to on-screen and they found that the on-screen participants were faster in resolving the tasks proposed, not in line with our study, which may indicate that the placement algorithm is a very important point within the city metaphor. More recently, Romano et al. [20] 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 [20], having as similarities that the *VR-BabiaXR* participants are faster than the *Screen-BabiaXR* participants and that the correctness does not worsen significantly.

8. Conclusions and future work

We presented two controlled experiments aimed at evaluating whether VR is well suited to visualize the *CodeCity* metaphor when compared with the traditional on-screen implementation. In both experiments, subjects were shown a *CodeCity*-like visualization of the same software system, either in VR or on-screen. For that, we used *BabiaXR-CodeCity*, part of the *BabiaXR* toolset for 3D data visualization that we developed for visualization, and GrimoireLab for the collection of data from the corresponding software repository. Subjects included people from both academia and industry, with a wide range of experience. Each subject performed 9 (*i.e.,* first experiment, repeating 4 of them in the old snapshot of the *JetUML* system) or 7 (*i.e.,* second experiment) tasks related to software comprehension, either immersed in VR or on-screen. In the first experiment the tasks corresponded to two moments in the development history of the system, while in the second, they corresponded to a single snapshot. The feedback collected in the first experiment helped us improving the design of the second experiment.

The results of both experiments show that immersion in VR led to a much shorter completion time for the tasks, compared to on-screen. In this respect, results were even more clear in the second experiment, which was more carefully crafted to minimize the effect of confounding variables.

In the first experiment, results show that subjects immersed in VR have less accuracy than those working in the on-screen environment. However, in the second experiment, with a more balanced setup, the accuracy in VR increased considerably, being even better than on-screen. However, for most of the tasks and in both environments, if we consider close matches (*i.e.,* answers in *Top 5*) instead of exact matches, correctness is always over 90%, making both environments well suited to solve these tasks.

The higher error rate in the first experiment was due to the lack of experience with VR devices of participants, the prototypical stage of our implementation, and the visual similarity of the buildings in some of the proposed tasks. The qualitative feedback that we received from subjects in the first experiment allowed us to mitigate these problems in the second one, which is, in our opinion, the main reason for the increased accuracy in the *VR-BabiaXR* environment. The feedback collected from participants in the second experiment is consistent with our appreciation.

Overall, we conclude 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, compared to on-screen. Considering these results, it could happen that VR might allow *CodeCity*-style visualizations to reach the *"tipping point",* beyond which they are viable and useful for practitioners.

An open issue that our study does not answer is to which extent subjects are directly comparing features of buildings, or relying on values provided by tooltips when performing the tasks. Tooltips provide exact numbers, easy to compare, and are obtained just by pointing at buildings. It would be an interesting future work to know their impact on results, finding when subjects directly use features of buildings, and when tooltips, to answer the questions in the study. However, this would not affect the overall result, since tooltips were available in both environments.

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 which deserve to be explored. The improvement of accuracy in VR, and the exploration of these new possibilities, are interesting lines of long-term future work.

As a result of the work to conduct our experiments, we also developed an implementation of *CodeCity* for the browser, which can run both on virtual reality and screen-based devices. This is a reusable artifact that other researchers may find useful for extending our studies, or for exploring what can be done with this technology.

Replication package

The data obtained for our experiments, and the materials needed to reproduce the experiment are available in the *Replication Package*.[14]

CRediT authorship contribution statement

David Moreno-Lumbreras: Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Validation, Visualization, Writing – original draft, Writing – review & editing. **Roberto Minelli:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Writing – original draft, Writing – review & editing. **Andrea Villaverde:** Investigation, Methodology, Software, Visualization, Writing – review & editing. **Jesus M. Gonzalez-Barahona:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Software, Supervision, Visualization, Writing

[14] Replication Package of the experiments: https://doi.org/10.5281/zenodo.7010180.

– original draft, Writing – review & editing. **Michele Lanza:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Supervision, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: David Moreno-Lumbreras reports financial support was provided by Community of Madrid. Andrea Villaverde reports financial support was provided by Community of Madrid. Jesus M. Gonzalez-Barahona reports financial support was provided by Community of Madrid. Jesus M. Gonzalez-Barahona reports financial support was provided by Spanish Government. Roberto Minelli reports financial support was provided by Swiss National Science Foundation. Michele Lanza reports financial support was provided by Swiss National Science Foundation.

Acknowledgments

We acknowledge the financial support of the Community of Madrid, Spain for the project IND2018/TIC-9669, the Spanish Government for the project RTI-2018-101963-B-I00 and the Swiss National Science foundation (SNSF), Switzerland for the project "INSTINCT" (SNF Project No. 190113). We also thank all the participants of our experiments

References

[1] C. Knight, M. Munro, Comprehension with[in] virtual environment visualisations, in; Proceedings Seventh International Workshop on Program Comprehension, 1999, pp. 4–11.
[2] Richard Wettel, Michele 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, 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, 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] Frank Steinbrückner, Claus Lewerentz, Representing development history in software cities, in: Proceedings of the 5th International Symposium on Software Visualization, SOFTVIS '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 193–202.
[6] Richard Wettel, Michele Lanza, Program comprehension through software habitability, in: 15th IEEE International Conference on Program Comprehension (ICPC '07), 2007, pp. 231–240.
[7] Brandon Jones, Manish Goregaokar, WebXR device API, in: W3C Working Draft, 2020.
[8] Dean Jackson, Jeff Gilbert, WebGL 2.0 Specification, Khronos Group Specification, 2020.
[9] Santiago Dueñas, Valerio Cosentino, Jesus M. Gonzalez-Barahona, Alvaro del Castillo San Felix, Daniel Izquierdo-Cortazar, Luis Cañas-Díaz, Alberto Pérez García-Plaza, GrimoireLab: A toolset for software development analytics, PeerJ Comput. Sci. 7 (2021) e601.
[10] David Moreno-Lumbreras, Roberto Minelli, Andrea Villaverde, Jesus M. Gonzalez-Barahona, Michele Lanza, CodeCity: On-screen or in virtual reality? in: Working Conference on Software Visualization, VISSOFT 2021, Luxembourg, September 27-28, 2021, IEEE, 2021, pp. 12–22.
[11] Harrison, Magel, Kluczny, DeKock, Applying software complexity metrics to program maintenance, Computer 15 (9) (1982) 65–79.
[12] V. Cosentino, S. Dueñas, A. Zerouali, G. Robles, 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.
[13] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, Jesus M. Gonzalez-Barahona, Perceval: Software project data at your will, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings, ICSE '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–4.
[14] Paul Ralph, ACM SIGSOFT empirical standards released, SIGSOFT Softw. Eng. Notes 46 (1) (2021) 19.
[15] R. Wettel, M. Lanza, R. Robbes, Software systems as cities: a controlled experiment, in: 2011 33rd International Conference on Software Engineering (ICSE), 2011, pp. 551–560.
[16] Jonathan Sillito, Gail C. Murphy, Kris De Volder, Questions programmers ask during software evolution tasks, in: Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, in: SIGSOFT '06/FSE-14, Association for Computing Machinery, New York, NY, USA, 2006, pp. 23–34.
[17] Nadim Nachar, The mann-whitney u: A test for assessing whether two independent samples come from the same distribution, Tutor. Quant. Methods Psychol. 4 (2008).
[18] N. Cliff, Dominance statistics: Ordinal analyses to answer ordinal questions, Psychol. Bull. 114 (1993) 494–509.
[19] J. Romano, J.D. Kromrey, J. Coraggio, 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.
[20] Simone Romano, Nicola Capece, Ugo Erra, Giuseppe Scanniello, Michele Lanza, On the use of virtual reality in software visualization: The case of the city metaphor, Inf. Softw. Technol. 114 (2019) 92–106.
[21] George Lakoff, Mark Johnson, Metaphors we Live by, University of Chicago Press, 1980.
[22] Vladimir Averbukh, Visualization metaphors, Program. Comput. Softw. 27 (2001) 227–237.
[23] Michael Balzer, Andreas Noack, Oliver Deussen, Claus Lewerentz, Software landscapes: Visualizing the structure of large software systems, in: Eurographics / IEEE VGTC Symposium on Visualization, The Eurographics Association, 2004.
[24] Hamish Graham, Hong Yul Yang, Rebecca Berrigan, A solar system metaphor for 3D visualisation of object oriented software metrics, in: Australasian Symposium on Information Visualisation, InVis.Au, Christchurch, New Zealand, 23-24 January 2004, in: CRPIT, vol. 35, Australian Computer Society, 2004, pp. 53–59.
[25] Ugo Erra, Giuseppe Scanniello, Towards the visualization of software systems as 3D forests: The codeTrees environment, in: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, Association for Computing Machinery, New York, NY, USA, 2012, pp. 981–988.
[26] Katsuhisa Maruyama, Takayuki Omori, Shinpei Hayashi, A visualization tool recording historical data of program comprehension tasks, in: Proceedings of the 22nd International Conference on Program Comprehension, in: ICPC 2014, Association for Computing Machinery, New York, NY, USA, 2014, pp. 207–211.
[27] Erich Gamma, Richard Helm, Ralph Johnson, John M. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, first ed., Addison-Wesley Professional, 1994.
[28] Christopher Alexander, Sara Ishikawa, Murray Silverstein, A Pattern Language: Towns, Buildings, Construction, Oxford University Press, New York, 1977.
[29] T. Panas, R. Berrigan, 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.
[30] Thomas Panas, Rüdiger Lincke, Welf Löwe, Online-configuration of software visualizations with Vizz3D, in: Proceedings of the 2005 ACM Symposium on Software Visualization, SoftVis '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 173–182.
[31] Thomas Panas, Thomas Epperly, Daniel Quinlan, Andreas Saebjornsen, Richard Vuduc, Communicating software architecture using a unified single-view visualization, in: 12th IEEE International Conference on Engineering Complex Computer Systems (ICECCS 2007), 2007, pp. 217–228.
[32] Andrian Marcus, Louis Feng, Jonathan I. Maletic, 3D representations for software visualization, in: Proceedings of the 2003 ACM Symposium on Software Visualization, SoftVis '03, Association for Computing Machinery, New York, NY, USA, 2003, pp. 27–ff.
[33] Guillaume Langelier, Houari Sahraoui, Pierre Poulin, Visualization-based analysis of quality for large-scale software systems, in: Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering, ASE '05, Association for Computing Machinery, New York, NY, USA, 2005, pp. 214–223.
[34] R. Wettel, M. Lanza, Visual exploration of large-scale system evolution, in: 2008 15th Working Conference on Reverse Engineering, 2008, pp. 219–228.
[35] Richard Wettel, Michele Lanza, Visually localizing design problems with disharmony maps, in: Proceedings of the 4th ACM Symposium on Software Visualization, SoftVis '08, Association for Computing Machinery, New York, NY, USA, 2008, pp. 155–164.
[36] J.D. Scarsbrook, R.K.L. Ko, B. Rogers, 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.
[37] Frank Steinbrückner, Claus Lewerentz, Representing development history in software cities, in: Proceedings of the 5th International Symposium on Software Visualization, SOFTVIS '10, Association for Computing Machinery, New York, NY, USA, 2010, pp. 193–202.
[38] Gergo Balogh, Tamas Gergely, Arpad Beszedes, Tibor 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.

[39] Federico Pfahler, Roberto Minelli, Csaba Nagy, Michele Lanza, Visualizing evolving software cities, in: 2020 Working Conference on Software Visualization (VISSOFT), 2020, pp. 22–26.
[40] P. Young, M. Munro, Visualising software in virtual reality, in: Proceedings. 6th International Workshop on Program Comprehension. IWPC'98 (Cat. No.98TB100242), 1998, pp. 19–26.
[41] J.I. Maletic, J. Leigh, A. Marcus, G. Dunlap, Visualizing object-oriented software in virtual reality, in: Proceedings 9th International Workshop on Program Comprehension. IWPC 2001, 2001, pp. 26–35.
[42] Florian Fittkau, Alexander Krause, Wilhelm Hasselbring, Exploring software cities in virtual reality, in: 2015 IEEE 3rd Working Conference on Software Visualization (VISSOFT), 2015, pp. 130–134.
[43] Juraj Vincur, Ivan Polasek, Pavol Navrat, Searching and exploring software repositories in virtual reality, in: Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology, VRST '17, Association for Computing Machinery, New York, NY, USA, 2017.
[44] D. Baum, J. Schilbach, P. Kovacs, U. Eisenecker, 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.
[45] L. Merino, M. Ghafari, C. Anslow, O. Nierstrasz, CityVR: Gameful software visualization, in: 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017, pp. 633–637.
[46] Nicola Capece, Ugo Erra, Simone Romano, Giuseppe Scanniello, Visualising a software system as a city through virtual reality, in: Augmented Reality, Virtual Reality, and Computer Graphics, Springer International Publishing, Cham, 2017, pp. 319–327.
[47] Martin Misiak, Andreas Schreiber, Arnulph Fuhrmann, Sascha Zur, Doreen Seider, Lisa 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.
[48] Andreas Schreiber, Marlene Brüggemann, Interactive visualization of software components with virtual reality headsets, in: 2017 IEEE Working Conference on Software Visualization (VISSOFT), 2017, pp. 119–123.
[49] Leonel Merino, Johannes Fuchs, Michael Blumenschein, Craig Anslow, Mohammad Ghafari, Oscar Nierstrasz, Michael Behrisch, Daniel A. Keim, On the impact of the medium in the effectiveness of 3D software visualizations, in: 2017 IEEE Working Conference on Software Visualization (VISSOFT), 2017, pp. 11–21.
[50] Marc-Oliver Rüdel, Johannes Ganser, Rainer Koschke, A controlled experiment on spatial orientation in VR-based software cities, in: 2018 IEEE Working Conference on Software Visualization (VISSOFT), 2018, pp. 21–31.