On the Use of Virtual Reality in Software Visualization: The Case of the City Metaphor

Simone Romano, Nicola Capece, Ugo Erra, Giuseppe Scanniello
University of Basilicata, Potenza, Italy

Michele Lanza
Università della Svizzera Italiana (USI), Switzerland

Abstract

Background: Software visualization is a program comprehension technique used in the context of software maintenance, reverse engineering, and software evolution analysis. In the last decade, researchers have been exploring 3D representations for visualizing software. Among these representations, one of the most popular is the city metaphor, which represents the source code of a target program as a virtual city. Recently, this metaphor has been also implemented in interactive software visualization tools that use virtual reality in an immersive 3D environment medium.

Aims: We assessed the city metaphor implemented in a traditional 3D visualization tool and in a virtual reality visualization tool with respect to the support these tools provide in the comprehension of Java software systems.

Method: We conducted a controlled experiment where we asked the participants to fulfill program comprehension tasks with the support of: (i) a traditional 3D visualization tool; (ii) a virtual reality visualization tool; and (iii) a non-visual exploration tool (i.e., Eclipse).

Results: The use of the city metaphor implementations (i.e., based on tradi-
tional 3D and virtual reality visualizations) significantly improved the correctness of the solutions to program comprehension tasks with respect to the use of a non-visual exploration tool. Moreover, when carrying out these tasks, the participants using the city metaphor implemented in a virtual reality visualization tool were significantly faster than those using a non-visual exploration tool.

Conclusions: Virtual reality is a viable means for software visualization.

Keywords: Code City, Software Visualization, Virtual Reality, Experiment

1. Introduction

Software visualization studies techniques and methods for graphically representing different aspects of software [1]. Software visualization is considered an effective means for program comprehension that is widely used in the context of software maintenance, reverse engineering, and re-engineering [2, 3].

In the last two decades, we have witnessed a proliferation of software visualization approaches defined to support a broad range of software engineering activities. In the last decade, researchers have been exploring 3D representations for visualizing software [3]. Among these 3D representations one of the most popular is the city metaphor [2, 4, 5, 6, 7, 8]. For example, a given application (or also system from here onward) is visualized as a city whose buildings represent its classes and whose districts depict the applications packages. The visual properties of the city artifacts represent software metrics. This metaphor was initially designed to let developers solve high-level program comprehension tasks on one system version [5]. This metaphor was later adapted to analyze the evolution of software applications throughout its versions [11] and to identify possible design issues [10]. The City metaphor has been also recently implemented in interactive software visualization tools that use virtual reality in an immersive 3D environment [11, 12].

There is a growing need for the assessment of software visualization approaches to demonstrate their effectiveness. Unfortunately, only a few software visualization approaches have been empirically validated so far (e.g., [2, 13]),
which is detrimental to the development of the software visualization field [3]. One of the reasons behind the shortage of empirical evaluation in that field lies in the considerable accidental complexity of such experiments [11, 13]. In addition, the variety of software visualization approaches makes it difficult to reuse the experimental material and the design of past empirical evaluations.

We present the results of a controlled experiment conducted to compare the city metaphor implemented in a traditional 3D software visualization tool and in virtual reality immersive 3D environment.

We also compared these tools with Eclipse, a popular IDE (Integrated Development Environment) in both academia and industry, which represents the current state-of-the-practice and therefore is the natural baseline for the comparison performed in our experiment. The controlled experiment focuses on the support these tools provide in the comprehension of Java source code. Our experiment can be considered an external replication of the experiment by Wettel et al. [2]. With respect to this experiment, we deliberately introduced changes to some parameters, e.g., a virtual-reality-based implementation of the City metaphor as the new treatment and the design to mitigate possible threats to conclusion validity.

We make the following contributions:

- A controlled experiment to assess the use of virtual reality in software visualization with a considerable number of participants (i.e., 42). This is valuable because it provides evidences on the benefits related to the use of an emerging and cheap technologies (e.g., Oculus Rift, a Head Mounted Display—HMD) in the software engineering field. The practitioner could take advantages from these evidences to support adoption decisions of these technologies.

- A replication of the study by Wettel et al. [2]. From a scientific perspective, this replication allows collecting further evidences, so bringing credibility to software visualization and the City metaphor, in particular.

The paper is organized as follows. In Section 2 we present related work.
In Section 3 we provide background information useful to better comprehend the research presented in this paper. The design of our controlled experiment is shown in Section 4 while the obtained results are presented and discussed in Section 5 and Section 6 respectively. We conclude the paper in Section 7.

2. Related Work

There are a number of software visualization approaches based on graph representations\(^1\) and polymetric views\(^2\) (e.g., [14, 15, 16, 17]). SNE (Synthetic Natural Environment) based approaches have been also proposed (e.g., [18, 19]). These kinds of approaches use synthetic natural environment to represent a subject application/system. That is, they use well-understood elements of the world to provide insights about an application [1]. We focus here only on visualization approaches and techniques based on SNE because our study focus on the city metaphor, which falls in this category. A deeper discussion on different kinds of software visualization approaches can be found in the literature review and taxonomy by Koschke [20] and Price et al. [21], respectively.

As for SNE based approaches, the city metaphor is one of the most explored (e.g., [4, 18, 22, 23, 24]) ones. For example, Wettel and Lanza [4] propose its use to allow a large- and small-scale understanding of object-oriented applications. In their proposal, classes are represented as buildings and packages as districts. The metaphor is implemented in the CodeCity tool. To improve the realistic aspect of the city, the authors focused on the design of the urban domain. To assess CodeCity and the underlying metaphor, the authors conducted a controlled experiment [2] with both students and professionals. Results suggest that the use of CodeCity leads to a statistically significant improvement in terms of task correctness and significantly reduces task completion time. The experiment presented in our paper is an external replication of their experiment. We describe

---

\(^1\)A target application is represented by means of a graph, where typically nodes are software entities and edges indicate relations between entities.

\(^2\)These views are lightweight visualizations enriched with software metrics.
both the city metaphor and its experimental assessment in Section 3.

Erra et al. [19] propose an SNE based approach that visualizes a subject application as a groups of atolls in a 3D environment. Atolls and palms represent packages and classes, respectively. The authors did not conduct an experimental assessment with users to validate their proposal.

A botanical tree metaphor is proposed by Kleiberg et al. [25]. They propose the use of forests of trees for the visualization of huge hierarchical structures and apply the proposed metaphor to the visualization of directory structures. Directories, files, and their relations are visualized using trees. The approach is a natural visual metaphor for hierarchically structured information. Also, Erra and Scanniello [26] propose a metaphor based on forests of trees and implemented it in the CodeTree tool. Differently from Kleiberg et al., the proposal by Erra and Scanniello supports the comprehension of object-oriented software applications and the visualization of their evolution [27]. In their research, the authors map information of source-code (e.g., software metrics) to take advantages of familiar concepts such as agglomerates of trees (or sub-forest), trunk, branches, leaves, and color of the leaves. The authors do not experimentally assess their SNE based approaches.

Graham et al. [28] propose a solar system based metaphor where each sun represents a package and planets are classes, while orbits represent the inheritance level within the package. The metaphor is used to analyze either static or evolving code and to show risky parts of the code.

Ghandar et al. [29] present a jigsaw puzzle metaphor. Each component of a software is represented as a piece of a jigsaw puzzle. The surface of a piece is used to graphically show the complexity of the software component. The metaphor does not provide a view at class granularity level.

SNE based approaches are also proposed to visualize information from the development process of software projects. For example, Martínez et al. [30] suggest a metaphor based on landscapes to visualize the integrated representation of software development processes. This metaphor is conceived to describe a number of aspects related to the development of a software application, but it
is not focused on source-code.

In recent years, the video game industry has shown a great interest in Virtual Reality (VR) technologies. Small and large companies have developed hardware and software technology to grow the VR by improving gradually its feature in terms of performance and level of immersion in video games.

Today, the context of use has also extended to other application contexts such as military, educational, and engineering. For example, Maletic et al. [24] describe a system called Imsovision for the visualization of the object-oriented software application. VR visualization was performed in a system composed of a 10 square feet room in which stereoscopic images are projected, creating the illusion of coexistence between objects and users in the room. The user wears liquid crystal glasses for viewing stereoscopic images and uses a controller to interact with the virtual scene. Differently, P.Kapec et al. [31] present a tool which allows visual analysis through graphs representing the applications using a VR interface based on standard desktop components such as icons, menu, etc. The visualization through augmented reality uses spatial optical see-through display, i.e., a Kinect V1 and a Leap motion sensor for interaction.

To date, the city metaphor is the most implemented metaphor in VR-based environments [11, 12, 32, 33]. We can speculate that this is due to the fact that this metaphor is very suitable to transmit a good sense of habitability and locality to the users [5], for example, when dealing with program comprehension and software maintenance tasks. For example, Fittkau et al. [32] propose an approach for the exploration and visualization of the software through the Oculus Rift and Microsoft Kinect. In particular, the authors developed ExplorViz to provide different gestures using a Natural User Interface (NUI). Souza et al. [34] propose a visualization of a subject application through augmented reality using the city metaphor to represent the evolution of software. Their system, called SkyscrapAR, represents packages such as districts, sub-packages as stacked districts and classes as buildings positioned in their respective packages. Merino et al. [11] propose CityVR—an interactive visualization tool that implements the city metaphor by using virtual reality in an immersive 3D environment. CityVR
targets the software maintainer, who has to perform software comprehension tasks to correct and evolve a software application. The authors present the results of an preliminary qualitative empirical evaluation, namely semi-structured interviews with six experienced developers. The most important findings can be summarized as follows: (i) developers felt curious, immersed, in control, excited, and challenged, (ii) developers spent considerable interaction time navigating and selecting elements, and (iii) developers were willing to spend more time using CityVR to solve software comprehension tasks since they perceived that time passed faster than in reality. Our virtual reality immersive 3D environment is inspired by CityVR. Our proposal is based on the use of the Oculus Rift (costumer version), through which we tried the feeling of total immersion in the visualized application. The Oculus Rift is equipped with a controller that allows easy interaction with a given virtual environment.

Later, Merino et al. [13] conducted a controlled experiment with nine participants and a qualitative study to compare the visualization of software as cities on a standard computer screen with an immersive augmented reality environment in the context of program comprehension tasks. They found that immersive augmented reality facilitates navigation and reduces occlusion, while performance (i.e., completion time, correctness, and recollection) is adequate, and developers obtain an outstanding experience.

We present the results of a controlled experiment with 42 participants. We also provide qualitative results from structured surveys the participants filled in before and after the experiment. Our study is larger than the second study by Merino et al. [13] and has a different baseline for comparison (i.e., Eclipse). We conducted a rigorous data analysis and to mitigate experimenters’ biases we used experimental material from the study by Wettel et al. [2].

The adoption of empirical methods is not very common in the software visualization field to assess approaches and their supporting tools [9, 13, 35, 36, 37, 38]. This might be due to the difficulty and the risks in conducting both qualitative and quantitative empirical studies. Similar to Wettel et al. [9] and Merino et al. [13], we aim to fulfill this lack by conducting a quantitative study.
(i.e., a controlled experiment with users) on one of the most popular and widely used metaphors in software visualization. We complement the results from this experiment by an analysis of qualitative feedback from the participants in this experiment. Our investigation has also the merit to replicate the study by Wettel et al. [9].

3. Background and Motivation

We first introduce the city metaphor (Section 3.1) and discuss how it has been implemented in a traditional 3D software visualization tool and in virtual reality immersive 3D environment (Section 3.2 and Section 3.3, respectively). We conclude by introducing the experiment by Wettel et al. [2] and presenting motivations behind our decision to replicate their experiment.

3.1. The City Metaphor

The city metaphor relies on the similarities between software constructs and cities. Researches have instantiated the city metaphor in different ways (e.g., [2, 3, 11, 6, 40]) differing on how the software constructs and their characteristics are visually described through the city metaphor. For example, the instance by Wettel et al. [2] and the one by Bacchelli et al. [40] both depict a class as a building, but the former maps the color of the building to the Lines Of Code (LOC) of that class, while the latter does not. Nevertheless, the instances of the city metaphor share the goal of representing a subject software application as a city. We consider the instance by Wettel et al. [2], which we simply call city metaphor from here on. Through the city metaphor, types (i.e., classes and interfaces) are depicted as buildings (parallelepipeds). The visual properties of each parallelepiped represent software metrics of the class:

- The height of the building reflects Number Of Methods (NOM)—the taller the building, the higher the number of methods.
- The base size of the building corresponds to Number of Attributes (NOA)—the larger the base, the higher the number of attributes.
• The color of the building is mapped to LOC—dark blue means few lines of code while light blue means many lines of code.

Blocks represent packages. Classes in the same package are placed in the same block. The color of the blocks ranges from dark grey to light grey based on the nesting level of the packages.

In Figure 1, we show how the visual representation of the FindBugs application, through the considered instance of the city metaphor implemented.

Figure 1: The building corresponding to the class IsNullValue of the FindBugs application is selected and the corresponding information is shown.

This figure depicts how FindBugs is shown in our implemented solutions, i.e., the 3D software visualization tool and the virtual reality immersive 3D environment. We refer to the first implementation of the city metaphor as Code2City and to the second implementation as Code2CityVR.

3.2. Code2City: a 3D Implementation of the City Metaphor

The user interacts with Code2City by using mouse and keyboard. These devices allow moving inside the city in any direction. In addition, Code2City offers also various features useful to understand the subject application. In particular, the user can identify an object of interest (i.e., building or district)
and select it through a viewfinder. When an object is selected (see Figure 1), it changes color becoming yellow and further information on the corresponding object is shown. If the object corresponds to a type, the system shows: type name, number of methods, number of properties, number of code lines, and the package name to which it belongs. If the object corresponds to a package, its full name and the contained classes is shown to the user.

Code2City allows searching by name and by callers. The latter allows identifying the types which perform calls to methods of the type provided by the user. Types that respect the search criterion are highlighted in red (see Figure 2).

![Figure 2: Buildings in red are the types satisfying a given search criterion.](image)

Our 3D Implementation of the city metaphor supports the identification of possible design problems using different colors to distinguish among the following code smells: brain classes, data classes, and god classes (See Figure 3).

We did not use the tool by Wettel et al. (i.e., CodeCity) because it was no more available for download and because it would have been incompatible with Oculus Rift. Therefore, we re-implemented CodeCity as Code2City.

3.3. Code2CityVR: a Virtual Reality implementation of the City Metaphor

Code2CityVR implements the same features as Code2City, while the interaction and visualization modes are different. The user is provided with a controller.
and a head mounted display (i.e., the Oculus Rift) that allow the immersion in the 3D environment. Figure 4 shows how a city is shown in the Oculus Rift.

The user interacts with the Code2City\textsubscript{\textit{VR}} through head movements. The user can select the virtual component using the direction in front of his head and a controller button. The user can move and observe the city from the perspective of a pedestrian who walks inside it because the Code2City\textsubscript{\textit{VR}} has been developed so that the size of the scene displayed is proportional to the size of the user. In this way, the user has the feeling of being really in a city. Through the virtual reality the user can fly over the city (bird’s-eye view), climb on top of the buildings, and look at the city from the higher buildings.

The user can change her movement speed, can view classes that represent code smells and perform searches. These searches are supported through the visualization and the interaction with a virtual keyboard, which can be activated and deactivated as desired. To interact with the virtual keyboard, the controller has to be used. To speed up the writing time for text queries, Code2City\textsubscript{\textit{VR}} provides an automatic text completion feature.

We did not use the implementation by Merino \textit{et al.} [11] (i.e., CityVR) because it does not provide the same features as CodeCity. Therefore the use
Figure 4: Visualization of the city through the Oculus Rift. The user views the city using stereoscopy providing the feel of 3D and total immersion within the virtual scene.

of CityVR would have biased the validity of the results from our replication of the experiment by Wettel et al. [2].

3.4. Assessing the City Metaphor and Motivations for the Replication

Wettel and Lanza proposed CodeCity [4] and conducted a controlled experiment with participants from academia and industry to empirically assess it in the execution of program comprehension tasks [4]. In Table 1, we summarize the main characteristics of that experiment.

| Table 1: Main characteristics of the baseline experiment. |
|---------------------------------|--------------------------------------------------|
| Participants                     | 45 (41 after data-cleaning) from industry and academia |
| Experimental Objects             | FindBugs, Azureus                                |
| Main Factor                      | Tool (i.e., CodeCity vs. Eclipse+Excel)           |
| Secondary Factor                 | Object (i.e., FindBugs vs. Azureus)              |
| Design                           | 2x2 factorial (i.e., between-subjects)           |
| Null Hypotheses                  | NH1. Tool does not significantly impact the correctness of the solutions to program comprehension tasks. |
|                                  | NH2. Tool does not significantly impact the completion time of program comprehension tasks. |
The authors involved 45 participants from both industry (i.e., professional developers) and academia (e.g., students or professors). Each participant was asked to perform comprehension tasks (see Table 2 for a description of the tasks) on Java programs, i.e., either FindBugs or Azureus (i.e., the experimental objects).

To comprehend them, the participants were provided with either CodeCity or Eclipse with some Excel spreadsheets—they reported information on source code metrics and bad smells of the program on which they had to accomplish the tasks. Summarizing, the experiment manipulated two factors: Tool and Object. The experiment design was a between-subjects. Wettel et al. formalized two null hypotheses, i.e., NH1 and NH2 as shown in Table 1. NH1 aimed to investigate on the correctness of the solutions the participants provided to comprehension tasks, while NH2 the time to complete these tasks. The results suggested that the participants who used CodeCity attained significant better correctness of the solutions with respect to who used Eclipse (+24%). As for the completion time, the results indicated that the participants who used CodeCity spent significantly less time (-12%).

The software engineering community has been embracing replications more readily (e.g., [41]). A replication can be intended as the repetition of an experiment. From the scientific and industrial perspectives, we can identify two primary motivations for conducting replications: (i) they are necessary to solve problems and collect evidences because they bring credibility to a given research and (ii) they are valuable because they provide evidence on the benefits of a software engineering practice thus allowing industrial stakeholders to use this information to support adoption decisions [42, 43, 44].

<table>
<thead>
<tr>
<th>Task</th>
<th>Concern</th>
</tr>
</thead>
</table>

Table 2: Task description of the baseline experiment.
A1. Locate all the unit tests of the program and identify the convention (or lack thereof) used by the developers to organize the tests. 

**Rationale.** Test classes are typically defined in packages according to a project-specific convention. Before integrating their work in the system, developers need to understand how the test classes are organized. Software architects design the high-level structure of the system (which may include the convention by which test classes are organized), while quality assurance engineers monitor the consistency of applying these rules in the system.

A2.1. Look for the term $T_1$ in the names of types and their fields and methods, and describe the spread of these types in the system.

**Rationale.** Assessing how domain knowledge is encapsulated in source code is important in several scenarios. To understand a system they are not familiar with, developers often start by locating familiar domain concepts in the source code. Maintainers use concept location on terms extracted from change requests to identify where changes need to be performed in the system. Software architects want to maintain a consistent mapping between the static structure and the domain knowledge. Each of these tasks starts with locating a term or set of terms in the system and assess its dispersion.

A2.2. Look for the term $T_2$ in the names of types and their fields and methods, and describe the spread of these types in the system.

**Rationale.** Same as for task A2.1. However, the term $T_2$ was chosen such that it had a different type of spread than $T_1$.

A3. Evaluate the change impact of the class $C$ by considering its caller types. The assessment is done in terms of both intensity (i.e., number of potentially affected types) and dispersion (how these types are distributed in the program).

**Rationale.** Impact analysis allows one to estimate how a change to a part of the system impacts the rest of the system. Although extensively used in maintenance activities, impact analysis may also be performed by developers when estimating the effort needed to perform a change. It also gives an idea of the quality of the system: A part of the system which requires a large effort to change may be a good candidate for refactoring.

A4.1. Find the 3 types with the highest number of methods.

**Rationale.** Classes in object-oriented systems ideally encapsulate one single responsibility. Since methods are the class unit of functionality, the number of methods metric is a measure of the amount of functionality of a class. Classes with an exceptionally large number of methods make good candidates for refactoring (e.g., split class), and therefore are of interest to practitioners involved in either maintenance activities or quality assurance.

A4.2. Find the 3 types with the highest average number of lines of code per method.

**Rationale.** It is difficult to prioritize candidates for refactoring from a list of large classes. In the absence of other criteria, the number and complexity of methods can be used as a measure of the amount of functionality for solving this problem related to maintenance and quality assurance.
B1.1. Identify the package with the highest percentage of god classes.

*Rationale.* God classes are classes that tend to incorporate an overly large amount of intelligence. Their size and complexity often make them a maintainers nightmare. Keeping these potentially problematic classes under control is important. By maintaining the ratio of god classes in packages to a minimum, the quality assurance engineer keeps this problem manageable. For a project manager, in the context of the software process, packages represent work units assigned to the developers. Assessing the magnitude of this problem allows him to take informed decisions in assigning resources.

B1.2. Identify the god class containing the largest number of methods.

*Rationale.* It is difficult to prioritize candidates for refactoring from a list of god classes. In the absence of other criteria (e.g., the stability of a god class over its evolution), the number of methods can be used as a measure of the amount of functionality for solving this problem related to maintenance and quality assurance.

B2.1. Identify the dominant class-level design problem in the program.

*Rationale.* God class is only one of the design problems that can affect a class. A similar design problem is the brain class, which accumulates an excessive amount of intelligence, usually in the form of brain methods (i.e., methods that tend to centralize the intelligence of their containing class). Finally, data classes are just dumb data holders without complex functionality, but with other classes strongly relying on them. Gaining a big picture of the design problems in the system would benefit maintainers, quality assurance engineers, and project managers.

B2.2. Write an overview of the class-level design problems. Describe your most interesting or unexpected observations.

*Rationale.* The rationale and targeted user roles are the same as for task B2.1. However, while the previous one gives an overview of design problems in figures, this task provides qualitative details and has the potential to reveal the types of additional insights obtained with visualization over raw data.

There are two factors that underlie the definition of a replication: *the procedure* (i.e., the steps followed to conduct the experiment) and *the researchers* (i.e., who conducted the replication).

Replication procedure ranges between *close* and *conceptual*. In a close replication the procedure is close to the original procedures as much as possible [45]. A replication is conceptual if the research question is the same, but the experimental procedure is different from that of the baseline experiment.

As for the researcher factor, we distinguish between *internal* and *external*. 
Internal replications are executed by the same researchers as those in the baseline experiment [46]. External replications are executed by different experimenters. External replications should be preferred because the results are slightly be affected by experimenters’ biases [47, 45]. If the results of a baseline experiment are confirmed then they can be considered more generalizable and the research more credible. We conducted an external replication of the experiment by Wettel et al. [2], i.e., the original researcher were not involved in the execution. With respect to this baseline experiment we introduced some changes in the procedure, e.g., an additional treatment (i.e., Code2CityVR) and a variation in the design. Our experiment can also be considered an operational replication [48]. A discussion of the changes introduced to the protocol, the operationalization, and the population is presented in Section 4.9, where we also discuss the rationale behind the changes we introduced.

4. Experiment

In the execution of our replication, we exploited the guidelines by Wohlin et al. [49], and Juristo and Moreno [47]. To present this replication, we used the template by Jedlitschka et al. [50].

4.1. Goals

The goal of our experiment, formalized according to the Goal Question Metric (GQM) template [51], is the following:

**Analyze** program comprehension for the purpose of evaluating the city metaphor implemented in a 3D visualization tool and in a virtual reality visualization tool with respect to the correctness of solutions to program comprehension tasks and the completion time of these tasks from the point of view of the researcher in the context of Bachelor students in Computer Science and Master students in Computer Engineering.
Based on the goal of our experiment, we formalized and investigated the following Research Questions (RQs):

**RQ1.** Do city metaphor implementations lead to better correctness of solutions to program comprehension tasks, compared to non-visual exploration tools?

**RQ2.** Do city metaphor implementations lead to better completion time of program comprehension tasks, compared to non-visual exploration tools?

### 4.2. Experimental Units

The participation in the experiment was on a voluntary basis: We neither forced nor payed the participants to take part in the replication. We invited, via e-mail, bachelor students (third-year) in Computer Science and master students (first-year) in Computer Engineering to take part in that replication. Among the invited students (i.e., 70), 54 students initially accepted our invitation, of which 42 (34 where undergraduate students, while eight were graduate) took part in the experiment.

### 4.3. Experimental Material

In our study, we used the following two programs written in Java: (1) **Jmol** ([jmol.sourceforge.net](http://jmol.sourceforge.net)), an open-source viewer for chemical structures in 3D, and (2) **FindBugs** ([findbugs.sourceforge.net](http://findbugs.sourceforge.net)), an open-source program for finding bugs (i.e., code instances that are likely to be errors) in Java programs.

Table 3 provides some code metrics for these programs like the number of types, methods, and fields, and LOC (Lines of Code).

Table 3: Some information on the programs used for the experiment.

<table>
<thead>
<tr>
<th>Program</th>
<th># Types</th>
<th># Methods</th>
<th># Fields</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jmol</td>
<td>634</td>
<td>8,148</td>
<td>6,273</td>
<td>106,305</td>
</tr>
<tr>
<td>FindBugs</td>
<td>1,744</td>
<td>10,123</td>
<td>4,836</td>
<td>92,571</td>
</tr>
</tbody>
</table>
Jmol was the program on which the participants carried out the training session, i.e., the participants were asked to perform program comprehension tasks by using the procedure they would use in the experiment (see Section 4.7).

FindBugs was the program (i.e., the experimental object) on which they performed the experimental session. FindBugs was the same experimental object as Wettel et al.’s experiment [2]. We reused the experimental material (e.g., tasks) these researchers made available in their technical report [52]. As for Jmol, we created the material by taking as an example the one Wettel et al. created for FindBugs [2, 52].

The participants were provided with one of the following tools:

- **Code2City.** The city metaphor implemented in our 3D visualization tool.
- **Code2CityVR.** The city metaphor implemented in our virtual reality immersive 3D environment.
- **Eclipse.** It is an open-source IDE, which can be extended via plugins. In particular, we plugged Metrics & Smells into Eclipse. This plugin allows Eclipse’s users to compute and display code metrics and detect smells.

These tools exploit the same software component to compute the code metrics and to detect the smells. This allowed us to provide the participants with the same values for the chosen metrics and the same code smells.

### 4.4. Tasks

We asked the participants to fulfill a subset of the ten tasks defined by Wettel et al. for FindBugs [2, 52]. We discarded two of them, A4.2 and B2.2 (see Table 2), because these tasks were not included in the final quantitative data analysis by Wettel et al. [2]. The tasks they defined covered various concerns. In Table 2 we provide the rationale behind each task as Wettel et al. reported in their paper [2].
4.5. Hypotheses, Parameters, and Variables

Each participant performed the program comprehension tasks with either Code2City, Code2City\textsubscript{VR}, or Eclipse. Therefore, the independent variable or main factor in the experiment (i.e.,) is \textbf{Tool}. It is a nominal variable that assumes three values: Code2City, Code2City\textsubscript{VR}, and Eclipse.

To quantify the correctness of the solutions to program comprehension tasks, we evaluated the solutions as done by Wettel \textit{et al.} in their baseline experiment \cite{2}. For each task, we assigned a score ranging between 0 and 1, where 0 means that the participant’s solution to that task is wrong while 1 means that the solution is correct. We summed the scores each participant achieved over the eight tasks to obtain an overall score between 0 and 8. For the completion time of the tasks we measured the time to complete the assigned tasks. Therefore, the dependent variables of our experiment are \textbf{Score} and \textbf{Time}.

We tested the same null hypotheses as Wettel \textit{et al.} did:

\textbf{NH1}. Tool does not significantly impact the correctness of the solutions to program comprehension tasks.

\textbf{NH2}. Tool does not significantly impact the completion time of program comprehension tasks.

\textbf{NH1} and \textbf{NH2} allowed us to study RQ1 and RQ2, respectively. The alternative hypothesis for either \textbf{NH1} or \textbf{NH2} admits an effect of Tool on the studied constructs. For example, if \textbf{NH1} is rejected—the alternative hypothesis is accepted—, we can conclude that \textit{Tool significantly impacts the correctness of the solutions to program comprehension tasks}.

4.6. Experiment Design

The design of our replication is \textit{one factor with more than two treatments} \cite{19}, where the treatments (or groups, from here on) are: Code2City, Code2City\textsubscript{VR}, and Eclipse. The former two treatments are needed to understand if the considered two different implementations of the city metaphor affect the correctness
of the solutions to program comprehension tasks and their completion time. Eclipse was chosen because it is widely used in both academia and industry and it was used (although with some Excel spreadsheets) in the baseline experiment \[2]\ as control treatment.

We randomly assigned each participant to either the treatments or the control—13 participants were administered with Eclipse, while 12 and 17 participants were administered with Code2City and Code2City\textsubscript{VR}, respectively.

4.7. Procedure

The experimental procedure consisted of the following steps:

1. We asked the participants to fill in a pre-questionnaire to gather demographic information.
2. We randomly assigned the participants to one of the following groups: Code2City, Code2City\textsubscript{VR}, and Eclipse.
3. Depending on the treatment, we trained the participants in the use of Code2City, Code2City\textsubscript{VR}, or Eclipse. We sent the participants in the Code2City group (i.e., those participants assigned to the Code2City treatment) a tutorial on the Code2City’s features needed to fulfill the experimental tasks (see Section 4.4). We asked the participants to follow this tutorial and practise Code2City on their own. Successively, these participants were involved in a training session, i.e., a laboratory session where they were asked to accomplish tasks similar to those in the experiment, but on a different program (i.e., Jmol, see Section 4.3). This session had a twofold goal: improving their ability to use Code2City and practising the experiment steps. Similarly, the participants in the Code2City\textsubscript{VR} and Eclipse groups received a tutorial on the tool to be used in the experiment. They participated in a training session too. The goal of this session was to let participants exercises the tool they would have then used in the experiment tasks.
4. To accomplish the experiment tasks, we assigned each participant to a computer of the laboratory, where they found the tool (e.g., Code2City)
and the experimental object (i.e., FindBugs). The participants were informed that they had maximum ten minutes per task (as done by Wettel et al. in the baseline experiment [2]). Thus, we asked the participants to write down the time it took to fulfill a task if it required less than ten minutes.

5. We asked the participants to fill in a Positive and Negative Affect Schedule (PANAS) questionnaire [53] and a post-questionnaire. Thanks to the PANAS questionnaire, we measured the positive and negative affects of the participants just after the use of Code2City, Code2CityVR, or Eclipse (see Section 5.3). On the other hand, we used the post-questionnaire to collect information of the participants’ perception of the tools they used to perform the experiment tasks (see Section 5.4). To create this questionnaire, we inspired to the one by Ahn et al. [54].

4.8. Analysis Procedure

We first exploited descriptive statistics and box-plots to summarize the distributions of the values of the dependent variables. To test the null hypotheses (i.e., NH1 and NH2), we ran the one-way ANOVA test. It is a parametric test for analyzing data from experiments like ours (i.e., experiments with one factor and more than two treatments) [49]. For any ANOVA test, we ensured to not violate its assumptions: normality of data and homogeneity of variance. To check the assumption of normality, we ran a Shapiro-Wilk test [55] (Shapiro test, from here onwards) for each group (e.g., the Code2CityVR group). As for the assumption of homogeneity of variance, we ran a Bartlett’s test [56]. We also planned to execute a post-hoc analysis if a null hypothesis would be rejected. In particular, we executed a Tukey’s HSD test [57]. This test is run after an ANOVA test to determine which groups in the sample are significantly different [58].

We accepted a probability of 5% of committing Type-I error (i.e., $\alpha = 0.05$). That is, we reject a null hypothesis when the obtained $p$-value is less than 0.05.
4.9. Differences with Respect to the Baseline Experiment

There are some differences between our experiment and the baseline one. Table 4: Summary of the differences between our experiment and the baseline one.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Our Experiment</th>
<th>Baseline Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool</td>
<td>Code2City, Code2City&lt;sub&gt;VR&lt;/sub&gt;, Eclipse</td>
<td>CodeCity, Eclipse+Excel</td>
</tr>
<tr>
<td>Design</td>
<td>One factor with more than two treatments (i.e., between-subjects)</td>
<td>2x2 factorial (i.e., between-subjects)</td>
</tr>
<tr>
<td>Experimental Objects</td>
<td>FindBugs</td>
<td>FindBugs, Azureus</td>
</tr>
<tr>
<td>Number of Tasks</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>Kind of Participants</td>
<td>Students</td>
<td>Students, Professors, Developers</td>
</tr>
</tbody>
</table>

These differences, sketched in Table 4, are:

- **Tool.** Both Code2City and CodeCity implement the city metaphor in a 3D visualization tool. Given the goal of our experiment, we needed an implementation of the city metaphor in a virtual reality tool (i.e., Code2City<sub>VR</sub>). This is why three tools were used for program comprehension in our experiment (instead of two as in the baseline experiment). With respect to the baseline experiment, we decided to extend Eclipse via the Metrics & Smells plugin (instead of providing some Excel spreadsheets with both code metrics and smells of the program to be comprehended), to increase the realism of the experiment. The use of Eclipse with Excel would not have been representative of industrial practices, thus it would have implied a threat of interaction of setting and treatment.

- **Design.** In our case the treatments (i.e., tools) were three, and thus we conceived an experiment whose design was one factor with more than two treatments. Furthermore, Wettel et al. did not find any significant interaction between Tool (i.e., the main effect) and Object (i.e., the secondary factor): The effect of Tool on the dependent variables was the same for each level of the Object factor (i.e., FindBugs and Azureus). This allowed us to control only one factor (i.e., Tool) in our experiment.
• **Experimental Object.** Given the above-mentioned considerations, we randomly chose one experimental object used by Wettel et al.

• **Number of Tasks.** Since Wettel et al. did not include the tasks A4.2 and B2.2 in their quantitative data analysis, we did not provide the participants with these two tasks.

• **Kind of Participants.** Unlike Wettel et al. that only trained the participants with CodeCity, we trained any participant with the tool she had to use in the experiment (e.g., a participant administered with the Code2City treatment had received the training on Code2City, while a participant administered with Eclipse had received the training on Eclipse).

5. Results

We present descriptive statistics and exploratory data analysis and show results from our statistical inference. We conclude with results from the analyses conducted on the data gathered from the post- and PANAS questionnaires.

5.1. Descriptive Statistics and Exploratory Analysis

In Table 5, we provide descriptive statistics for the dependent variables (i.e., Score and Time).

Table 5: Some descriptive statistics for each tool and each dependent variable.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Statistic</th>
<th>Code2City</th>
<th>Code2CityVR</th>
<th>Eclipse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Score</td>
<td>Min</td>
<td>4.8</td>
<td>4.0</td>
<td>1.9</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>6.183</td>
<td>5.788</td>
<td>4.754</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>6.1</td>
<td>5.8</td>
<td>5.0</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>8.0</td>
<td>7.4</td>
<td>6.4</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>1.158</td>
<td>1.036</td>
<td>1.071</td>
</tr>
<tr>
<td>Time</td>
<td>Min</td>
<td>24</td>
<td>21</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>38.167</td>
<td>27.294</td>
<td>43.385</td>
</tr>
<tr>
<td></td>
<td>Median</td>
<td>36</td>
<td>27</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>58</td>
<td>38</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td>SD</td>
<td>10.053</td>
<td>5.205</td>
<td>5.994</td>
</tr>
</tbody>
</table>
The distributions for these variables are also graphically summarized by the boxplots in Figure 5.a and Figure 5.b, respectively.

![Boxplots for Score (a) and Time (b).](image)

The boxes for Score (see Figure 5.a) suggest that there is no difference between who was administered with Code2City and who was administered with Code2City\_VR. These boxes overlap and the descriptive statistic values (see Table 5) are similar (e.g., the mean values are 6.183 and 5.788 for Code2City and Code2City\_VR, respectively). By comparing the boxes of either Code2City or Code2City\_VR with that of Eclipse, we can notice that the box of Eclipse is much lower than the others. That is, it seems that who was administered with Eclipse attained a Score value worse than who was administered with either Code2City or Code2City\_VR (on average the Score values was 4.754 for Eclipse while 6.183 and 5.788 for the other two treatments).

As for Time, we can observe noticeable differences among the boxes for Code2City, Code2City\_VR, and Eclipse since they do not overlap (see Table 5.b). It seems that the participants provided with Code2City\_VR spent on average less time than others (the mean value for Code2City\_VR was 27.294 while it was 38.167 for Code2City and 43.385 for Eclipse).
5.2. Hypotheses Testing (and Post-hoc Analysis)

In Table 6, we report the p-values returned by the ANOVA test for Score and Time and the results of the post-hoc analysis. [⋆]: p-value less than α.

Table 6: Results (i.e., p-values) from the ANOVA and post-hoc (i.e., HSD) tests.

<table>
<thead>
<tr>
<th>Variable</th>
<th>ANOVA</th>
<th>Post-hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Code2City_VR-Code2City</td>
</tr>
<tr>
<td>Score</td>
<td>0.005*</td>
<td>0.601</td>
</tr>
<tr>
<td>Time</td>
<td>&lt;0.001*</td>
<td>0.001*</td>
</tr>
</tbody>
</table>

Before applying the ANOVA test for Score, we verified the required assumptions. As for the normality assumption, the Shapiro test indicated that the data were all normally distributed since the returned p-values were equal to: 0.22 for Code2City, 0.202 for Code2City\_VR, and 0.079 for Eclipse. The data had also the same variance (the p-values returned by the Bartlett’s test was 0.923). Summarizing, all the assumptions of the ANOVA test were satisfied.

As shown in Table 6, the ANOVA test for Score returned a p-value (i.e., 0.005) less than α. This allows us to reject NH1 and accept the alternative hypothesis: Tool significantly impacts the correctness of the solutions to program comprehension tasks. The results of the HSD test suggest significant differences between the data distributions of Code2City and Eclipse (p-value = 0.035), and between those of Code2City\_VR and Eclipse (p-value = 0.006).

For Time, the assumptions of the ANOVA test were all verified. In particular, the Shapiro test returned the following p-values: 0.173 for Code2City, 0.44 for Code2City\_VR, and 0.67 for Eclipse. As for the Bartlett’s test, it returned a p-value equal to 0.052. The results of the ANOVA test (see Table 6) suggest that Time is significant (i.e., p-value less than 0.001). Therefore, we can reject NH2, namely: Tool significantly impacts the completion time of program comprehension tasks. We can also observe that there are significant differences between the distributions of Code2City\_VR and the others (p-value is equal to 0.001 when comparing Code2City\_VR with Code2City, while less than 0.001 when comparing Code2City\_VR with Eclipse).

25
5.3. Results on PANAS – Positive And Negative Affects

As a further analysis, we investigated if the tool used for the program comprehension tasks had influenced the feelings and emotions of the participants. We asked them to fill in the PANAS questionnaire [53] just after they had completed the experiment task. PANAS is a self-report questionnaire consisting of two scales of ten items each: the positive affect scale measuring positive affects and the negative affect one measuring negative affects. Each item in the questionnaire represents a feeling/emotion. Thus, a subject is asked to rate (from 1=“not at all” to 5=“extremely”) the extent to which she feels each of these feelings/emotions.

We then computed the Positive Affect Score (PAS) and the Negative Affect Score (NAS) by summing the scores in the positive affect and negative affect scales [53]. Both PAS and NAS range in between 10 and 50. A high value of PAS indicates high levels of positive affect, thus the higher the value the better it is. A high value of NAS indicates high levels of negative affect, that is the lower the value the better it is. The interested reader can find the PANAS questionnaire in [53].

In Figure 6, we report the boxplots arranged by Tool that summarize the PAS and NAS values.

![Boxplots for PAS and NAS](image)

Figure 6: Boxplots for PAS (a) and NAS (b).
We can observe that independently from the values of the Tool variable, the distributions for PAS are higher than the ones for NAS. This seem to indicate that after using Code2City, Code2CityVR, and Eclipse the participants felt more positive affects than negative affects.

By looking at Figure 6.a, we can observe that there is a noticeable difference between the box of Code2CityVR and the one of Eclipse. In particular, the participants administered with Code2CityVR seem to have PAS values (mean=31.412, SD=6.032) higher than those administered with Eclipse (mean=24.231, SD=6.496). A similar outcome, but less pronounced, can be observed by comparing the boxes of Code2CityVR and Code2City. That is, it seems that the use of Code2CityVR lead to better PAS values with respect to the use of Code2City (mean=28.167, SD=9.713). To understand if these differences in the PAS values are significant, we ran an ANOVA test (as done for Score and Time). Before running this test, we verified the assumptions for the applicability of this test. The normality assumption was satisfied (0.77, 0.456, and 0.137 were the p-values the Shapiro tests returned for the Code2City, Code2CityVR, and Eclipse distributions, respectively) as well as the assumption of homogeneity of variance (the Bartlett’s test returned 0.183 as p-values).

In Table 7, we report the p-value of the ANOVA test for PAS and the results of the post-hoc analysis (i.e., HSD test).

<table>
<thead>
<tr>
<th>Variable</th>
<th>ANOVA (Kruskal)</th>
<th>Post-hoc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Code2CityVR-Code2City</td>
<td>Code2CityVR-Eclipse</td>
</tr>
<tr>
<td>PAS</td>
<td>0.04*</td>
<td>0.48</td>
</tr>
<tr>
<td>NAS</td>
<td>0.164</td>
<td>–</td>
</tr>
</tbody>
</table>

The result of ANOVA test justifies a post-hoc analysis since the p-value is less than $\alpha = 0.05$, namely: Tool significantly influences the positive affects of the participants.
The HSD test indicates a significant difference between the distributions of Code2City\textsubscript{VR} and Eclipse: The participants provided with Code2City\textsubscript{VR} felt significantly better positive affects than those provided with Eclipse.

As far as NAS is concerned, the distributions of Code2City, Code2City\textsubscript{VR}, and Eclipse (see Figure 6) are quite similar one another. On average, the values of NAS are 11.667 for Code2City, 13.235 for Code2City\textsubscript{VR}, and 13.538 for Eclipse. The SD values are, respectively, equal to: 4.271, 3.784, and 5.425.

To confirm that there was no difference in the NAS values among the distributions of Code2City, Code2City\textsubscript{VR}, and Eclipse, we could not apply an ANOVA test because the assumption of normality was not met (the Shapiro tests returned p-values less than \(\alpha\) for Code2City, Code2City\textsubscript{VR}, and Eclipse). Therefore, we ran a Kruskal-Wallis test, which represents the non-parametric alternative to ANOVA [49]. As shown the p-value reported in Table 7, the Kruskal-Wallis test did not indicate a significant difference in the NAS values (p-value is 0.164). Therefore, we could not perform any post-hoc analysis.

5.4. Answers to the Post-Questionnaire

We summarize the participants’ answers to the post-questionnaire. We do not report the analyses for all the questions in the post questionnaire for space reasons. We selected all those questions whose answers are more interesting. Most of the questions were formulated as statements, where the participants had to rate (from 1=“I strongly disagree” to 7=“I strongly agree”) how much they agreed with such statements. To summarize the participants’ answers, we employed barplots—one for each tool. Since the questions were grouped, in the post-questionnaire, by theme (e.g., playfulness), we show the barplots and summarize the participants’ answers according to these themes.

5.4.1. Playfulness

In Figure 7, we show the barplots representing the answers to the questions on the playfulness of Code2City, Code2City\textsubscript{VR}, and Eclipse.
Answers to P1 (When interacting with Tool, I did not realize the time elapsed)

Answers to P2 (I had fun when fulfilling the tasks with Tool)

Answers to P3 (Using Tool to fulfill the tasks made me happy)

Answers to P4 (Using Tool stimulated my curiosity to explore the program)

Figure 7: Barplots representing the answers to the questions P1 (a), P2 (b), P3 (c), and P4 (d) on the playfulness of Code2City, Code2CityVR, and Eclipse.

Most participants administered with Code2City and Code2CityVR did not realize the passage of time (see Figure 7a). The participants administered with Eclipse seems to agree less with the statement of the question P1. Those who used Code2City and Code2CityVR had more fun than who used Eclipse (See Figure 7b). The use of Code2City and Code2CityVR to fulfill the tasks made the participants happier (see Figure 7c and Figure 7d). The barplots suggest that Code2City and Code2CityVR stimulated more the curiosity of their users to explore the program than Eclipse (see Figure 7d).
Summing up, it seems that the playfulness of Code2City and Code2CityVR is comparable, while that of Eclipse is worse.

5.4.2. Perceived Usefulness

In Figure 8, we summarize the answers to the questions on the perceived usefulness of Code2City, Code2CityVR, and Eclipse.

The Code2CityVR’s users believed to accomplish the tasks faster than the Code2City’s users and Eclipse’s users (See Figure 8a). For question PU2 (see Figure 8b), we did not observe huge differences between the tools.
The participants provided with Code2City, Code2CityVR, or Eclipse agreed that the tool assigned to them increased their productivity. The barplots in Figure 8.c indicate that the Code2CityVR’s users were mostly convinced to improve the correctness of the program comprehension tasks because they used Code2CityVR. This pattern is not noticeable for Code2City nor Eclipse. The participants provided with Code2City and Code2CityVR believed that the tasks were easier (see Figure 8.d). Both Code2City and Code2CityVR are perceived as more useful than Eclipse when carrying out program comprehension tasks. The perceived usefulness of Code2CityVR is slightly better than that of Code2City.

5.4.3. Behavioral Intention to Use

In Figure 9, we report the barplots of the questions about the behavioral intention to use Code2City, Code2CityVR, and Eclipse.

![Barplots representing the answers to the questions B1 (a) and B2 (b) on the behavioral intention to use Code2City, Code2CityVR, and Eclipse.](image)

The barplots in Figure 9.a indicate that most of the Code2CityVR’s users would like to use Code2CityVR on a regular basis. This outcome is not noticeable for Code2City nor Eclipse. Moreover, it seems the participants administered with Code2City and Code2CityVR were mostly favorable to use these tools, rather than an IDE, to carry out program comprehension tasks (see Figure 9.b).
6. Discussion

We discuss the results according to the defined research questions and the implications of the obtained results and possible threats to their validity.

6.1. Answering the Research Questions

**RQ1.** Both boxplots and descriptive statistics for the Score variable suggest that there is a difference in the correctness of solutions to program comprehension tasks when participants use the city metaphor implementations (i.e., Code2City and Code2City\textsubscript{VR}), rather than Eclipse (with the Metrics & Smells plugin), to fulfill such tasks. This difference is significant (as suggested by the testing of NH1) and in favour of the city metaphor implementations, which are comparable (based on the results of the post-hoc analysis). Therefore, we can positively answer RQ1: *the city metaphor implementations lead to better correctness of solutions to program comprehension tasks with respect to Eclipse.* This outcome is coherent with that of the Wettel et al.’s experiment [2].

**RQ2.** For the Time variable, the results of our data analysis indicate a significant difference in the completion time when the participants carried out the program comprehension tasks by means of Code2City, Code2City\textsubscript{VR}, or Eclipse. The participants using Code2City\textsubscript{VR} spent significantly less time to accomplish the assigned tasks with respect to who used Code2City or Eclipse. Therefore, we answer RQ2 as follows: *the city metaphor implementations (the one based on virtual reality, in particular) lead to better completion time of program comprehension tasks with respect to Eclipse.* Again, this outcome is consistent with the results by Wettel et al. [2].

6.2. Overall Discussion

The city metaphor, independently from its implementation, seems to support the execution of comprehension tasks better than a popular IDE, i.e., Eclipse. Both tools implementing the city metaphor seem to aid the participants in carrying out program compression tasks better than Eclipse. In addition, Code2City and Code2City\textsubscript{VR} are perceived more useful than Eclipse.
The participants in our controlled experiment also found these tools more pleasant/enjoyable. Code2CityVR (more than Code2City) positively affects the feelings and emotions of the participants.

The participants seem to find Code2CityVR more useful than Code2City. The results did not show a huge difference between Code2CityVR and Code2City from a qualitative perspective, i.e., playfulness, behavioural intention to use, and positive and negative affects. The obtained quantitative results suggest that the use of Code2CityVR (with respect to Code2City) allows the participants to spend less time to accomplish program comprehension tasks. To summarize, providing users with tools implementing city metaphor, regardless of the time to accomplish a comprehension task, might be considered equivalent. However, we have to mention that the use of any VR display might induce side effects such as headache, nausea, and visual annoyance.

Our results strengthen the validity of the results of the experiment by Wettel et al. [2]: the use of the city metaphor leads to better correctness of solutions to program comprehension tasks in less time as compared to the use of Eclipse.

6.3. Implications and Future Extensions

We focus on the researcher and the practitioner perspectives for the discussion of the implications and future extensions for our research.

The use of VR seems to be a viable means in the software visualization field. Our results justify future research in this respect and have practical implications from both the researcher and the practitioner perspectives.

As compared to Eclipse, the use of Code2CityVR positively affects the feelings and emotions of the developers and the correctness of the solutions to program comprehension tasks. This finding is relevant for the developer interested in the adoption of VR to perform comprehension tasks. The researcher could be interested in studying the correlation between feelings and emotions and the correctness of program comprehension tasks. In addition, it could be interesting from a researcher perspective to verify if the benefits from the use of VR (delineated just before) are still valid in a long run. In other words,
it could be possible that the novelty of the used technology could lead to the observed quantitative (e.g., less time to accomplish a comprehension task) and qualitative benefits (e.g., happiness and positive affects while accomplishing a comprehension task) and they could decrease with time.

Code2CityVR is designed to be easily adapted to object-oriented programming languages a different from Java. Although our results cannot be generalized to programs written in different programming languages, the researcher could be interested in studying the application of Code2CityVR in the context of programs written in other languages.

6.4. Threats to Validity

We discuss the threats the could affect the validity of our results with respect to internal, external, construct, and conclusion validity.

**Internal Validity** concerns uncontrolled factors that may alter the effect of the treatments on the dependent variables:

- **Selection.** Allowing volunteers to take part in an experiment (as it is in our case) may influence the results since volunteers might be more motivated than the whole population [49].

- **Resentful demoralization.** A participant receiving a less desirable treatment might may not behave as good as she generally does. For example, a participant receiving the Eclipse treatment could be less motivated.

**External Validity** regards the ability to generalize the results:

- **Interaction of selection and treatment.** Students took part in our experiment, thus generalizing the obtained results to the population of professional developers poses a threat to the external validity. However, involving students in software engineering experiments has a number of advantages (e.g., having preliminary empirical evidence) [59]. Moreover, we trained the participants in performing the program comprehension tasks with the provided tools in order to make them expert users of these tools.
We believe the use of students in our experiment is appropriate as the
literature suggests [59, 60]. In addition, the observed outcomes confirm
with stronger evidence those by Wettel et al. [2].

**Construct Validity** concerns the relation between theory and observation:

- **Mono-method bias.** Using a single method to measure a given construct
  might lead to misleading results in the case there was a measurement
  bias. However, we used the same dependent variables as the experiment
  by Wettel et al. [2] since our experiment is an external replication of such
  an experiment.

- **Hypothesis guessing.** The participants in our experiment might guess
  the experiment goal and thus behave on the basis of their guesses. This
  kind of threat has been mitigated by the used experimental design (i.e.,
  each participant experimented only one treatment).

- **Evaluation apprehension.** Some participants might be afraid of being
  evaluated. For example, their apprehension to be evaluated might lead
  them to perform poorly. Since participation in the experiment was on vol-
  untary basis, the effect of the evaluation apprehension should be marginal
  on the observed results. In addition, the participants were aware that
  their data would be treated anonymously.

**Conclusion Validity** regards treatments vs. outcomes:

- **Reliability of measures.** The participants were asked the write down
  the time needed to carry our the program comprehension tasks. This
  might affected the validity of the results if they did not report this infor-
  mation truthfully.

- **Hypotheses guessing.** The participants were aware of being part of
  a study regarding the use of software visualization tools. However, the
  participants were not aware of the specific constructs being investigated
  in the experiment.
7. Conclusion

We presented the results of a controlled experiment to compare two different implementations of the city metaphor and a popular IDE (i.e., Eclipse). The two implementations of the city metaphor were: 3D- and VR- based. These tools have been compared with respect to the support they provide in the comprehension of Java source code. Our experiment is an external replication of the (baseline) experiment by Wettel et al. [2].

We summarize the most important take-away results of our experiment as follows. The use of city metaphor implementations leads to an improvement in both correctness and completion time of program comprehension tasks, over the Eclipse. This outcome is coherent with that of the baseline experiment and increases the generalizability of the results. We observed that developers that use the VR implementation of the city metaphor spent significantly less time to complete the experimental task. The use of the VR-based implementation also leads to a higher satisfaction of the participants. These two outcomes suggest that VR might represent a viable tool in software visualization. Although we cannot draw definitive conclusions, the results from our experiment seem to justify future research on the use of VR in software visualization.

References


