On the use of virtual reality in software visualization: The case of the city metaphor

Simone Romano[a,*], Nicola Capece[b], Ugo Erra[b], Giuseppe Scanniello[b], Michele Lanza[c]

[a] *University of Bari, Bari, Italy*
[b] *University of Basilicata, Potenza, Italy*
[c] *Software Institute – Università della Svizzera italiana (USI), Lugano, Switzerland*

A R T I C L E   I N F O

Keywords:
Code city
Software visualization
Virtual reality
Experiment

A B S T R A C T

Background: Researchers have been exploring 3D representations for visualizing software. Among these representations, one of the most popular is the *city metaphor*, which represents a target object-oriented system 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 displayed on a standard computer screen and in an immersive virtual reality with respect to the support provided 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)* an integrated development environment (Eclipse) with a plugin for gathering code metrics and identifying bad smells; and *(ii)* a visualization tool of the city metaphor displayed on a standard computer screen and in an immersive virtual reality.

Results: The use of the city metaphor displayed on a standard computer screen and in an immersive virtual reality significantly improved the correctness of the solutions to program comprehension tasks with respect to Eclipse. Moreover, when carrying out these tasks, the participants using the city metaphor displayed in an immersive virtual reality were significantly faster than those visualizing with the city metaphor on a standard computer screen.

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

1. Introduction

Software visualization refers to the visualization of different aspects of software [1]. It is considered an effective means for program comprehension, which 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. Researchers have been exploring 3D representations for visualizing software [3]. Among these 3D representations one of the most popular is the city metaphor [2,4–8]. For example, a target object-oriented software system is visualized as a city whose buildings represent its classes and whose districts depict its 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]. Later, the city metaphor was adapted to analyze the evolution of software systems throughout their versions [9] and to identify design issues [10]. The city metaphor has been recently implemented in interactive software visualization tools that use virtual reality in an immersive 3D environment medium (*e.g.,* [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 might be 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 evaluations [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 3D visualization tool displayed

* Corresponding author.
E-mail addresses: simone.romano@uniba.it (S. Romano), nicola.capece@unibas.it (N. Capece), ugo.erra@unibas.it (U. Erra), giuseppe.scanniello@unibas.it (G. Scanniello), michele.lanza@usi.ch (M. Lanza).

https://doi.org/10.1016/j.infsof.2019.06.007

on a standard computer screen and in an immersive virtual reality. 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 is based on the one by Wettel et al. [2]. With respect to this experiment, we deliberately introduced changes to some parameters. For example, we considered a further treatment—an implementation of the city metaphor displayed in an immersive virtual reality.

In this paper, we make the following contributions:

- A controlled experiment to assess the use of virtual reality in software visualization with a high number of participants (*i.e.,* 42). This is valuable because it provides evidence on the benefits related to the use of an emerging technology (*e.g.,* Oculus Rift, a head-mounted display) in the software engineering field. The practitioner could take advantage from this evidence to support adoption decisions of such a kind of technology.
- From a scientific perspective, our experiment allows collecting further evidence on the benefits deriving from the use of the city metaphor for program comprehension, so bringing credibility to both software visualization and city metaphor.

The paper is organized as follows. In Section 2, we present work related to our research. 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 Sections 5 and 6, respectively. We conclude the paper in Section 7.

2. Related work

In the software visualization field, there is a high number of metaphors that use a synthetic natural environment to represent a subject system [7,14–16]. These metaphors use well-understood elements of the world to provide insights about a system [1]. In this context, the city metaphor is one of the most explored [4,17–20]. Wettel and Lanza [4] proposed the use of this metaphor to allow a large- and small-scale understanding of object-oriented software systems. In their proposal, classes are represented as buildings and packages as districts. The authors named their visualization tool CodeCity.[1] To assess their tool and the underlying metaphor, the authors conducted a controlled experiment with participants from both academia and industry [2]. The results suggest that the use of CodeCity leads to a significant improvement in terms of task correctness and significantly reduces task completion time. Our experiment is based on that by Wettel et al. [2]. In Section 3, we provide further details on both city metaphor and empirical assessment by Wettel et al.

To date, the city metaphor is the most implemented metaphor in immersive 3D environment media [11–13,21–23]. We can speculate that this is due to the fact that the city metaphor is very suitable to transmit a good sense of habitability and locality to the users [5], for example, when dealing with program comprehension tasks. For example, Fittkau et al. [22] proposed an approach for the exploration and visualization of software systems as cities through Oculus Rift and Microsoft Kinect. The authors implemented their approach in ExplorViz. This tool provided different gestures for the interaction between the user and the visualized city. Then they conducted 11 structured interviews to investigate the usability of these gestures. The results indicated that the gestures for translation, rotation, and selection were considered highly usable, while the zooming gesture was not.

Souza et al. [21] proposed a visualization of a target system through augmented reality using the city metaphor to represent its evolution. Their system, called SkyscrapAR, represents packages as districts, subpackages as stacked districts, and classes as buildings positioned in their respective packages. The authors did not conduct any empirical assessment of SkyscrapAR.

Merino et al. [11] proposed CityVR—an interactive visualization tool that implements the city metaphor in an immersive virtual reality medium. CityVR targets the software maintainer, who has to perform software comprehension tasks to correct and evolve a software system. The authors present the results of a 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 implementation of code city displayed in an immersive virtual reality is inspired by CityVR. Our proposal is based on the use of Oculus Rift (costumer version), which provides the feeling of total immersion in the visualized software system.

Merino et al. [24] conducted a controlled experiment with 27 participants. Each of them experimented: 3D visualizations of the city metaphor across a standard computer screen, an immersive 3D environment, or a physical 3D printed model. The participants were asked to visualize software systems to solve comprehension tasks. The authors found that: *(i)* the participants that visualized code cities using a physical 3D printed model required the least time to identify outliers (*e.g.,* classes with the highest number of lines of code); *(ii)* those who visualized code cities using an immersive virtual reality obtained the highest recollection; and *(iii)* when using a standard computer screen to visualize code cities, the participants perceived the least difficulty to identify outliers.

Later, Merino et al. [13] conducted a controlled experiment with nine participants where they compared the city metaphor displayed on a standard computer screen with the same metaphor displayed in an immersive augmented reality in the context of program comprehension tasks. To perform this comparison, the authors also considered data points from nine participants of a previous experiment [24]. They also conducted a qualitative investigation where nine participants experimented the space-time cube visualization technique displayed in immersive augmented reality. The main findings that emerged from this experiment are: *(i)* developers who visualized code cities on a standard computer screen require the least time to identify outliers and achieve the highest correctness, although they perceive the most difficulty in identifying outliers and *(ii)* the use of an immersive augmented reality medium allow recollecting many details of systems while offering a good level of engagement.

In this paper, we present the results of a controlled experiment to assess the city metaphor displayed on a standard computer screen and in an immersive virtual reality when carrying out program comprehension tasks. Our research contributes with those by Merino et al. [11,13,24] to increase the body of knowledge on the impact of the medium on the effectiveness of 3D software visualizations (the city metaphor, in particular). However, there are a number of differences between our research and those by Merino et al. For example, we conducted a controlled experiment, while the empirical investigation by Merino et al. [11] consisted of semi-structured interviews (similarly, our research differs from that by Fittkau et al. [22]). In Table 1, we summarize the main differences between our experiment and those by Merino et al. [13,24]. First, we considered different treatments (*e.g.,* Eclipse, a state-of-the-art IDE). Second, we involved a higher number of participants (*i.e.,* 42)—the number of participants can impact the conclusion validity of an experiment (*e.g.,* due to low statistical power) [25]. Third, to mitigate experimenter bias [25], we used the experimental materials of the study by Wettel et al. [2]. This design choice also allowed us to compare our results with those by Wettel et al.—if we confirmed their results, we

[1] https://wettel.github.io/codecity.html.

Table 1
Summary of the differences between our experiment and those by Merino et al. [13,24].

Characteristic	Our experiment	Merino et al.'s experiment [13]	Merino et al.'s experiment [24]
Treatments	Eclipse, code city displayed across standard computer screen and immersive virtual reality media	Code city displayed across standard computer screen and immersive augmented reality media	Code city displayed across standard computer screen, immersive virtual reality, and physical 3D printed model media
# Participants	42	9 (+9 from Merino et al. [24])	27
# Tasks	10 (from Wettel et al. [2])	6 (from Merino et al. [24])	9
Investigated Aspects	Completion time, correctness, feelings/emotions, perceived usefulness (including perceived task difficulty), playfulness, behavioral intention to use, and side effects	Completion time, correctness, feelings/emotions, perceived task difficulty, and recollection	Completion time, correctness, feelings/emotions, perceived task difficulty, and recollection

would increase the evidence on the effectiveness of the city metaphor when carrying out comprehension tasks. Fourth, we investigated different aspects like playfulness.

3. Background and motivations

In this section, we first introduce the city metaphor (Section 3.1) and then we discuss the experiment by Wettel et al. [2] together with the motivations behind ours (Section 3.2).

3.1. The city metaphor

The city metaphor relies on the similarities between software constructs and cities. Researchers have instantiated the city metaphor in different ways (*e.g.,* [2,5,11,26,27]) 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. [27] 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 system 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.

3.2. Assessing the city metaphor and motivations for our experiment

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 [2]. In Table 2, we summarize the main characteristics of that experiment.

The authors involved 45 participants from both industry and academia (*e.g.,* students or professors). Each participant was asked to perform comprehension tasks (see Table 3 for a description of the tasks) on Java programs (either FindBugs or Azureus).

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: NH1 and NH2 (Table 2). NH1 aimed to investigate 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., [28–31]), where a replication has to be intended as a repetition of a baseline experiment [32]. Unfortunately, there is no agreement yet on terminology, typology, purposes, operation, and other replication issues [29,30,33]. Regarding the lack of shared terminology, authors use different definitions for the same kind of replication and use the same definition to refer to different kinds of replication [34]. Recently, Gómez et al. [33] proposed a different classification for the types of replications based on: *protocol, operationalizations, populations*, and *experimenters*. Based on changes to these four dimensions, the authors established three types of replications: *literal*, where the aim is to run as exact a replication of the baseline experiment as possible; *operational*, where the aim is to vary some (or all) of the dimensions of the baseline experiment configuration; and *conceptual*, where experimenters have "nothing more than a clear statement of the empirical fact". Our study can be considered an operational replication of the experiment by Wettel et al. [2] because we introduced several changes in the four dimensions by Gómez et al. [33]. For example, Wettel et al. [2] were not involved in the design and the execution of our experiment, but one of these researchers shared with us the experimental material and discussed with us the obtained experimental results. A deeper 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 experiment, we exploited the guidelines by Wohlin et al. [25] and Juristo and Moreno [35]. To present this experiment, we used the template by Jedlitschka et al. [36]. The experimental materials are available on the web[2] as well as the raw data. We also made available a technical report where we provide details on the tools implementing the city metaphor and used in our experiment.

4.1. Goals

We formalized and investigated the following Research Questions (RQs):

RQ1. Do city metaphor implementations displayed on a standard computer screen and in an immersive virtual reality lead to better

[2] http://graphics.unibas.it/Code2City/index.md.html.

Table 2
Main characteristics of the Wettel et al.'s experiment.

Participants	45 (41 after data-cleaning) from industry and academia
Experimental Objects	FindBugs, Azureus
Main Factor	Tool—CodeCity vs. Eclipse+Excel
Secondary Factor	Object—FindBugs vs. Azureus
Design	2×2 factorial—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.

Table 3
Task description as reported in the Wettel et al.'s paper [2].

Task	Concern
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.	Structural understanding
A2.1. Look for the term *T1* in the names of types and their fields and methods, and describe the spread of these types in the system. *Rational.* 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 domain knowledge. Each of these tasks starts with locating a term or set of terms in the system and assess its dispersion.	Concept location
A2.2. Look for the term *T2* 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 *T2* was chosen such that it had a different type of spread than *T1*.	Concept location
A3. Evaluate the change impact of the class *C* by considering its caller types. The assessment is done in terms of both intensity (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.	Change impact analysis
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 classs 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.	Metric-based analysis
A4.2. Find the 3 types with the highest mean 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.	Metric-based analysis
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.	Focused design assessment
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.	Focused design assessment
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.	Holistic design assessment
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.	Holistic design assessment

correctness of solutions to program comprehension tasks, compared to an IDE with a plugin for gathering code metrics and identifying bad smells?

RQ2. Do city metaphor implementations displayed on a standard computer screen and in an immersive virtual reality lead to better completion time of solutions to program comprehension tasks, compared to an IDE with a plugin for gathering code metrics and identifying bad smells?

4.2. Experimental units

The participation in the experiment was on a voluntary basis: we neither forced nor paid the participants to take part in our experiment. We invited, via e-mail, bachelor students (third-year) in Computer Science and master students (first-year) in Computer Engineering (at the University of Basilicata, Potenza, Italy) by asking them if they wanted to participate in our experiment. Among 70 invited students, 54 students initially accepted our invitation, of which 42 (34 where undergraduate students, while eight were graduate) took part in the experiment.

4.3. Tasks

We asked the participants to fulfill a subset of the ten tasks defined by Wettel et al. for FindBugs [2,37]. We discarded two of them, A4.2 and B2.2 (Table 3), because these tasks were not included in their final quantitative data analysis by Wettel et al. [2]. In particular, they discarded the task A4.2 because, to carry out this task, the CodeCity's users had to have a deep knowledge of the CodeCity's customization features as well as experience with the underlying Smalltalk programming language. As for the task B2.2, it was conceived to collect qualitative data. That is to say that the tasks considered in our quantitative data analysis are the same as the quantitative data analysis by Wettel et al. [2].

The tasks covered various concerns. In Table 3, we provide the rationale behind each task as reported in the Wettel et al.'s paper [2].

Table 4
Some information on the programs used for the experiment.

Program	NOT	NOM	NOA	LOC
Jmol	634	8148	6273	106,305
FindBugs	1744	10,123	4836	92,571

4.4. *Experimental materials and tools*

In our study, we used the following two programs written in Java: (1) **Jmol**,[3] an open-source viewer for chemical structures in 3D, and (2) **FindBugs**,[4] an open-source program for finding bugs (*i.e.,* code instances that are likely to be errors) in Java programs. In Table 4, we provide some code metrics for these programs like the NOT (Number Of Types), NOM, NOA, and LOC.

Jmol was the program on which the participants carried out the training session. We chose this program because it was previously used in studies on software visualization [9,10] and because it was large enough and not obvious to justify the use of software visualization. The participants were asked to perform program comprehension tasks by using the procedure they would use in the experiment (Section 4.7). FindBugs was the program (or experimental object) on which they performed the experimental session. The participants did not have any knowledge on the FindBugs codebase. 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 [37]. As for Jmol, we created the material by taking as an example the one Wettel et al. created for FindBugs [2,37].

The participants were provided with one of the following tools:

- **Code2City.** An implementation of the city metaphor displayed on a standard computer screen.
- **Code2City$_{VR}$.** An implementation of the city metaphor displayed displayed in an immersive virtual reality.
- **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 identify bad smells in a Java codebase.

In the following, we provide further details on these tools. In particular, we provide some implementation details and sketch those features useful to accomplish the experimental tasks described in Section 4.3. Further details on the implementations of Code2City and Code2City$_{VR}$ can be found in the paper by Capece et al. [12] and in our on-line technical report.

4.4.1. *Code2City*

The user interacts with Code2City by using mouse and keyboard. These devices allow moving inside a 3D scene like in first-person games. Indeed, the user can fly over the city (bird's-eye view), climb on top of the buildings, and look at the city from the highest buildings. The user can also move and observe the city from the perspective of a pedestrian who walks inside the city. Code2City has been developed so that the size of the displayed scene is proportional to the size of the user.

To move the user, we used the WASD based video game control method. The W and S keys control forward and backward movements, respectively. On the other hand, the A and D keys control left and right *strafing,* respectively. We opted for the WASD keys because the arrow keys are not ergonomic together with a right-handed mouse. To move upwards, the user exploits the space key, while the Ctrl key is used to move downwards. The mouse allows rotations with respect to the normal and transverse axes.

In addition, Code2City offers also various features useful to understand the subject system. In particular, the user can identify an object of interest (building or district) and select it through a viewfinder. When an object is selected, 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 to. If the object corresponds to a package, its full name and the contained classes are shown to the user.

Code2City allows searching the types of a subject program by specifying their names. Indeed, the user writes down a search string (by using the keyboard) that should match either the entire name of the type or a part. Types that respect the search string are highlighted in red (Fig. 1). This feature could be used to accomplish the following experiment tasks: A1, A2.1, and A2.2. Code2City also allows searching types by callers. This feature allows identifying the types, which perform calls to methods of the type provided by the user. Types that respect the search criterion are also highlighted in red (Fig. 1). This feature could be used to perform the task A3. On the other hand, the task A4.1 could be accomplished by looking for the highest three buildings shown in Code2City.

Code2City supports the identification of possible design problems using different colors to distinguish among the following smells: brain classes are shown in light green, data classes are shown in heavenly, and god classes are shown in violet. Fig. 2 shows FindBugs in Code2City as well as the brain, data, and god classes it contains. This feature should help the accomplishment of the following tasks: B1.1, B1.2, and B2.1.

We did not use CodeCity (*i.e.,* the tool by Wettel et al. [5]) because, according to our experimental goals, we would have needed two versions of CodeCity: *(i)* one displaying the city metaphor on a standard computer screen; and *(ii)* one displaying the same metaphor in an immersive virtual reality. Unfortunately, CodeCity was incompatible with an immersive virtual reality medium (*e.g.,* it was not devised to work with Oculus Rift).

4.4.2. *Code2City$_{VR}$*

Code2City$_{VR}$ implements the same features as Code2City. That is, it provides the same support as Code2City when carrying out the experiment tasks. On the other hand, the interaction and visualization modes in Code2City$_{VR}$ are different from those in Code2City. In fact, the user is provided with a controller—Xbox One Wireless Controller—and a head-mounted display—Oculus Rift—that allows the immersion in the 3D environment. Fig. 3 shows how a city is shown on Oculus Rift.

The user interacts with the Code2City$_{VR}$ through head movements that allow rotations with respect to the normal and transverse axes. Differently from the mouse, head-mounted display also allows a rotation with respect to the longitudinal axis. The user can select the virtual component (*e.g.,* a building) using the direction in front of his head and a controller button. In this way, the user has the feeling of being really in a city. The user can change her movement speed by pressing a key button on the controller. Since a physical keyboard is not present, searching types (by name and callers) are supported through the visualization and the interaction with a virtual keyboard, which can be activated and deactivated by a key of the controller. To interact with the virtual keyboard, the user exploits the controller (*i.e.,* the right stick). To speed up the writing time for text queries, Code2City$_{VR}$ provides an automatic text completion feature.

Code2City and Code2City$_{VR}$ share the same software architecture. They also exploit the software component to compute the code metrics and to detect smells. Code2City and Code2City$_{VR}$ differ on the medium used to display the city metaphor: a standard computer screen vs. an immersive virtual reality. This guaranteed that participants that experimented Code2City and Code2City$_{VR}$ dealt with the same information (*e.g.,* the same smells) when accomplishing the experimental tasks. To implement Code2City and Code2City$_{VR}$, we used Unreal Engine (version 4.15.3). We used this game engine because it provides a good trade-off

[3] jmol.sourceforge.net.
[4] findbugs.sourceforge.net.

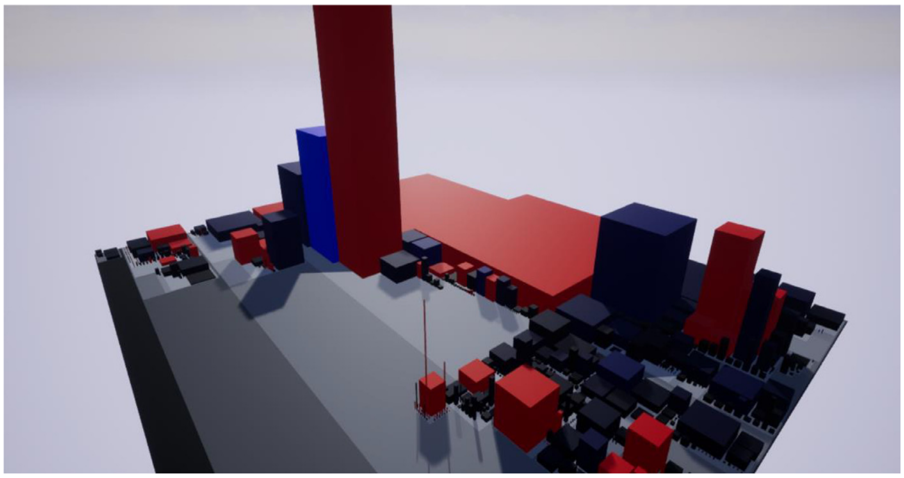


Fig. 1. Buildings in red are the types satisfying a given search criterion. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

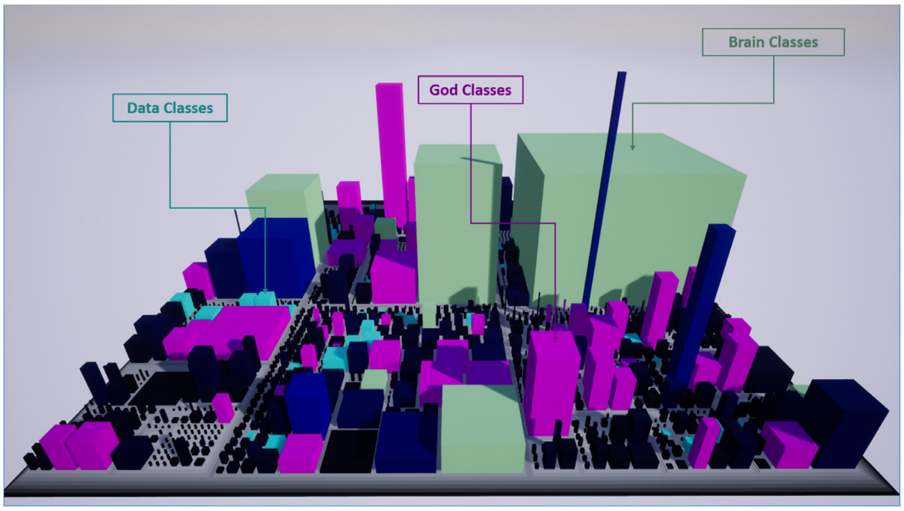


Fig. 2. Code smells: brain classes (light green), data classes (heavenly), god classes (violet). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

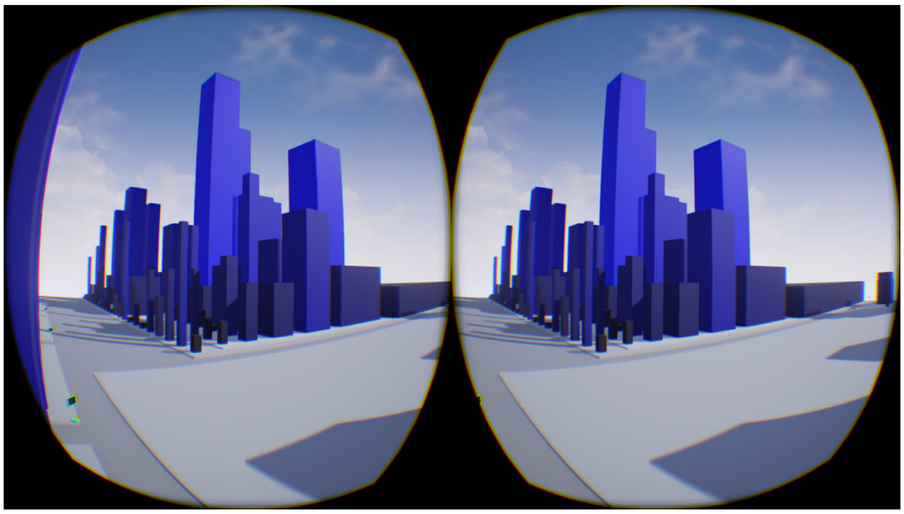
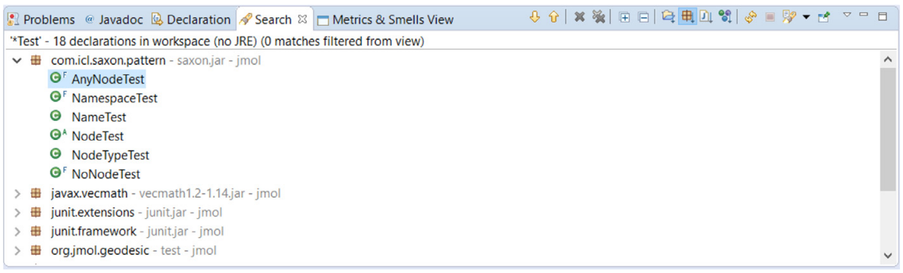


Fig. 3. Visualization of the city through Oculus Rift.

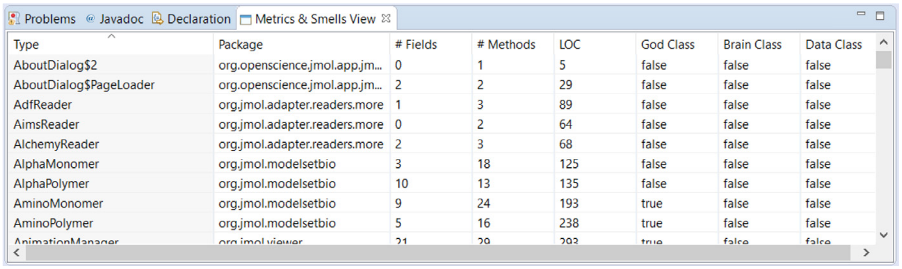
between rendering of the scenes, ease of use, and performance of immersive virtual reality (*e.g.,* 45 frames per seconds on FindBugs).

In this paper, we restricted the description of Code2City and Code2City$_{VR}$ to the essential because our main goal was not to propose these solutions, but their effectiveness in the execution of program comprehension tasks. Further details on the implementation of both Code2City and Code2City$_{VR}$ can be found in the paper by Capece et al. [12] and in our on-line technical report. For example, the interested reader can find details on how we improved the photo-realism of the scene by adding two lights: ambient light and directional light.



```
'*Test' - 18 declarations in workspace (no JRE) (0 matches filtered from view)
com.icl.saxon.pattern - saxon.jar - jmol
    AnyNodeTest
    NamespaceTest
    NameTest
    NodeTest
    NodeTypeTest
    NoNodeTest
javax.vecmath - vecmath1.2-1.14.jar - jmol
junit.extensions - junit.jar - jmol
junit.framework - junit.jar - jmol
org.jmol.geodesic - test - jmol
```

Fig. 4. View reporting the results of a search in Eclipse.



Type	Package	# Fields	# Methods	LOC	God Class	Brain Class	Data Class
AboutDialog$2	org.openscience.jmol.app.jm...	0	1	5	false	false	false
AboutDialog$PageLoader	org.openscience.jmol.app.jm...	2	2	29	false	false	false
AdfReader	org.jmol.adapter.readers.more	1	3	89	false	false	false
AimsReader	org.jmol.adapter.readers.more	0	2	64	false	false	false
AlchemyReader	org.jmol.adapter.readers.more	2	3	68	false	false	false
AlphaMonomer	org.jmol.modelsetbio	3	18	125	false	false	false
AlphaPolymer	org.jmol.modelsetbio	10	13	135	false	false	false
AminoMonomer	org.jmol.modelsetbio	9	24	193	true	false	false
AminoPolymer	org.jmol.modelsetbio	5	16	238	true	false	false
AnimationManager	org.jmol.viewer	21	29	293	true	false	false

Fig. 5. View reporting both metric values and bad smells in Eclipse.

We did not use CityVR by Merino et al. [11], because it does not provide the same features as CodeCity [9]. Therefore the use of CityVR would have biased the validity of the results from our experiment.

4.4.3. Eclipse

Eclipse (Mars)[5] provides functionality to support developers during their daily work activities. For example, it allows developers to surf their source code, search in source code, run their programs, *etc.* Thanks to the search functionality, developers can carry out the tasks from A1 to A3 (Table 3). In particular, developers can search for any kind of software entities (*e.g.,* methods or classes) or simple text. It is also possible to customize the search in several ways. For example, when searching for software entities, developers can specify if the search matches will be declarations or references (*e.g.,* callers). The results of a search are reported in a *tree view* showing the location of each search match as well as the total number of matches. Fig. 4 shows how Eclipse reports the results of a research.

To support the tasks from A4.1 to B2.2, Wettel et al. provided the participants in their experiment with some Excel spreadsheets reporting NOM, NOA, and LOC of each Java class. The spreadsheets also reported the brain, data, and god classes. Instead of providing this information in spreadsheets, we developed *Metrics & Smells*, a plugin for Eclipse. It allows gathering code metrics (NOM, NOA, and LOC) and identifying bad smells (brain, data, and god classes), afterward, it reports this information in a *table view*. This functionality provides support for the tasks A4.1, B1.1, B1.2, and B2.1. In Fig. 5, we show how the Metrics & Smells plugin reports both the values for the considered code metrics and the presence or the absence of the supported bad smells.

It is worth mentioning that the Metrics & Smells plugin exploits the same software component as Code2City and Code2City$_{VR}$ to compute software metrics and detect smells. This is to show the same information (code metric values and bad smells) whatever the tool was.

4.5. Hypotheses, parameters, and variables

Each participant performed the program comprehension tasks with either Code2City, Code2City$_{VR}$, or Eclipse. Therefore, the independent variable or main factor in the experiment is **Tool**. It is a nominal variable that assumes three values: *Code2City, Code2City*$_{VR}$, and *Eclipse*.

To quantify the correctness of the solutions to program comprehension tasks, we evaluated the solutions as done by Wettel et al. [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. As for the completion time of the tasks, we measured the time to complete the assigned tasks. Therefore, the dependent variables of our experiment are **Score** and **Time**.

We tested the same null hypotheses as Wettel et al. [2] did:

NH1. Tool does not significantly impact the correctness of the solutions to program comprehension tasks.
NH1. Tool does not significantly impact the completion time of program comprehension tasks.

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

4.6. Experiment design

The design of our experiment is *one factor with more than two treatments* [25], where the treatments are: Code2City, Code2City$_{VR}$, and Eclipse. This design is a kind of between-subjects designs exploited to avoid carryover effects that can plague within-subjects designs. The former two treatments are needed to understand if the considered different kinds of visualization 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 Wettel et al.'s experiment [2] as control treatment.

We randomly assigned each participant to the treatment groups, namely 18 students for each group. All the 54 participants took part in the training sessions, while 42 of them participated in the actual experimental sessions—13 participants were administered with Eclipse, while 12 and 17 participants were administered with Code2City and Code2City$_{VR}$, respectively. It is worth mentioning that we did not rebalance the number of participants in the treatment groups after the training sessions because each of these sessions were conceived to train the participants on the tool to be used in the actual experimental sessions. Indeed, re-training a few participants of the group with the highest number of participants (Code2City$_{VR}$) and moving them to the other two groups could not guarantee a balance among the treatment groups

[5] https://www.eclipse.org/downloads/packages/release/mars/2.

and possibly it would affect results in an undesirable way. For example, the following kinds of internal validity threats could be introduced: compensatory rivalry or resentful demoralization. We also recall that the unbalance of the number of participants in the treatment groups is not a major issue when performing statistical inferences [25].

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. Given the experiment design, we randomly assigned the participants to the treatment groups: Code2City, Code2City$_{VR}$, or Eclipse.
3. Depending on the treatment group, we trained the participants in the use of Code2City, Code2City$_{VR}$, 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 (Section 4.3). We informed them that they had to carefully read the tutorial before the training session took place (in a laboratory at the University of Basilicata). Before starting the training session, we asked these participants to follow the tutorial and practice Code2City on their own without any time limit. Successively, these participants were involved in a training session, where they were asked to accomplish tasks similar to those of the experiment, but on a different program. For each task, they had ten minutes at most (as in the experimental session). Similarly, each participant in the Code2City$_{VR}$ and Eclipse groups received a tutorial on the tool to be used in the experiment, practiced the tool on their own, and then took part in the training session. The training sessions had a twofold goal: improving the participants' ability to use a certain tool (*e.g.,* Code2City) and practicing the experiment steps. The training sessions took place the days before the experimental ones. This is to avoid as much as possible fatigue effects during the experiment.
4. To accomplish the experiment tasks, we assigned each participant of the Code2City and Eclipse groups to a computer of the laboratory (it is the same laboratory as the training sessions), where they found the tool and the experimental object. We arranged experimental sessions in which two/three participants at a time accomplished the tasks. The computer configuration was the same for any participant, namely: Windows 10 Pro; 16 GB of RAM; Intel Core i7-3820 (3.6GHz); and NVIDIA Titan XP GPU. In each session, we avoided any interaction among the participants by monitoring them. The participants of the Code2City$_{VR}$ group accomplished the tasks in the same laboratory as the participants of the Code2City and Eclipse groups, but one at a time. We introduced this difference in the experiment execution because we had available only one Oculus Rift device. The computer configuration for the Code2City$_{VR}$ group was the same as the other groups. The participants (regardless of the treatment group) were informed that they had a maximum of ten minutes per task (as done by Wettel et al. [2]). Thus, we asked the participants to write down the time it took to fulfill a task if it required less than ten minutes. Supervisors (the first or the second author) took care that the participants correctly wrote down that information.
5. We asked the participants to fill in a Positive and Negative Affect Schedule (PANAS) questionnaire [38] 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, Code2City$_{VR}$, or Eclipse (Section 5.3). On the other hand, we used the post-questionnaire to collect information on the participants' perception of the tools they used to perform the experiment tasks (Section 5.4). To create this questionnaire, we inspired to the one by Ahn et al. [39]. An extended analysis of both PANAS and post-questionnaire data can be found in the paper by Romano et al. [40].

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 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) [25]. 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 [41] (Shapiro test, from here onwards) for each group of participants. In case the assumption of normality was met, we opted for Bartlett's test [42] to check the assumption of homogeneity of variance. That is, homogeneity of variance was verified only when the assumption of normality was met. We used Bartlett's test because it is recommended when the underlying populations are normally distributed [43]. In case the ANOVA assumptions would be violated, we planned to use the Kruskal–Wallis test, which represents the nonparametric alternative to ANOVA [25].

We also planned to execute a post-hoc analysis if a null hypothesis would be rejected. To this end, we planned to execute Tukey's HSD test [44] or Dunn's test [45]. Tukey's HSD test is run after an ANOVA test to determine which groups in the sample are significantly different [46]. Similarly, Dunn's test [45] is run after a Kruskal–Wallis test.

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. Summary of the differences

There are some differences between our experiment and that by Wettel et al. [2]. These differences, sketched in Table 5, are:

- **Tool.** Both Code2City and CodeCity implement the city metaphor in a 3D visualization tool displayed on a standard computer screen. Code2City also implements the same functionality as CodeCity to carry out the experimental tasks. Given our research questions, we needed the same visualization tool in an immersive virtual reality (Code2City$_{VR}$). With respect to the experiment by Wettel et al., 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 and then mitigate external validity threats. Finally, by implementing Code2City, Code2City$_{VR}$, and the Metrics & Smells plugin, we could provide the same information to the participants whatever the treatment group was. This is because different metric analysis tools can return values of code metrics that differ from one another [47]. Similarly, the technique implemented in a smell analysis tool can affect the detected smells [48].
- **Design.** In our case, the treatments 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 factor) 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 (FindBugs and Azureus). This allowed us to control only one factor (Tool) in our experiment so simplifying the complexity of the design, reducing risks of failure, and mitigating conclusion validity threats.
- **Experimental object.** Given the above-mentioned considerations, we randomly chose one experimental object used by Wettel et al.. For example, the use of a single experimental object

Table 5
Summary of the differences between our experiment and the Wettel et al.'s one [2].

Characteristic	Our experiment	Wettel et al.'s experiment
Tool	Code2City, Code2City$_{VR}$, Eclipse	CodeCity, Eclipse+Excel
Design	One factor with more than two treatments—between-subjects	2×2 factorial—between-subjects
Experimental Objects	FindBugs	FindBugs, Azureus
Number of Tasks	10	12
Kind of Participants	Students	Students, Professors, Developers

Table 6
Some descriptive statistics for each tool and each dependent variable.

Variable	Statistic	Code2City	Code2City$_{VR}$	Eclipse
Score	Min	4.8	4.0	1.9
	Mean	6.183	5.788	4.754
	Median	6.1	5.8	5.0
	Max	8.0	7.4	6.4
	SD	1.158	1.036	1.071
Time	Min	24	21	32
	Mean	38.167	27.294	43.385
	Median	36	27	45
	Max	58	38	52
	SD	10.053	5.205	5.994

allowed us to mitigate possible threats related to data analysis and number of participants in the groups (statistical power). The selection of the object in a random fashion allowed to mitigate researcher bias.

- **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. That is, the tasks considered in the quantitative data analysis by Wettel et al. are the same we considered in ours.
- **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). It allowed us to mitigate threats to the reliability of treatment implementation, a kind of conclusion validity threat. That is, the implementation of the different treatments has to be as standard as possible over different participants and occasions [25].

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 6, we provide some descriptive statistics for Score and Time dependent variables. The distributions for these variables are also graphically summarized by the boxplots in Fig. 6.a and .b, respectively.

The boxes for Score (Fig. 6.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 (Table 6) 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$_{VR}$ (mean 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 (Table 6.b). It seems that the participants provided with Code2City$_{VR}$ spent on 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 7, we report the p-values returned by the ANOVA test for Score and Time and the results of the post-hoc analysis. 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 Bartlett's test was 0.923). Summarizing, all the assumptions of the ANOVA test were satisfied.

As shown in Table 7, 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 Bartlett's test, it returned a p-value equal to 0.052. The results of the ANOVA test (Table 7) 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).

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 [38] 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 [38]. 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 [38].

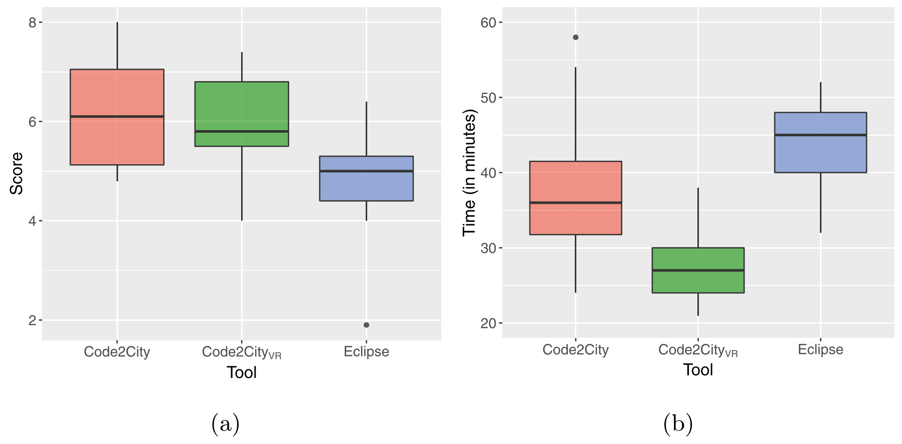


Fig. 6. Boxplots for Score (a) and Time (b).

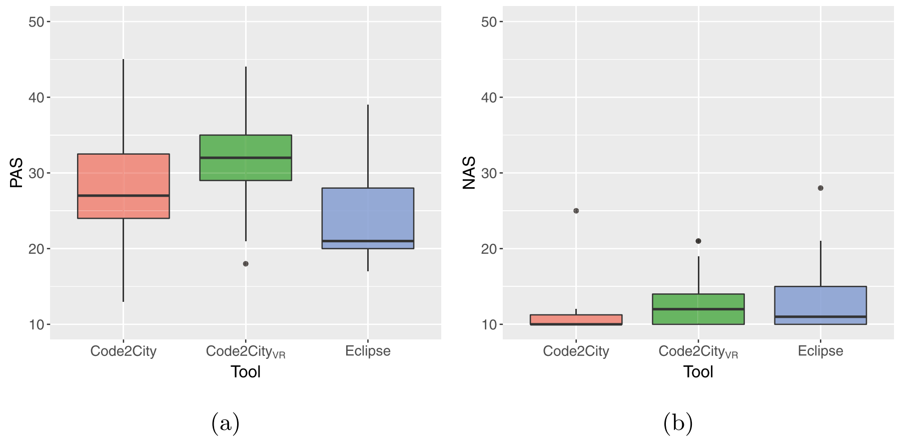


Fig. 7. Boxplots for PAS (a) and NAS (b).

Table 7
p-values from the ANOVA and the HSD tests. *p*-values less than α are highlighted with [*].

Variable	ANOVA	Post-hoc		
		Code2City$_{VR}$-Code2City	Code2City$_{VR}$-Eclipse	Code2City-Eclipse
Score	0.005*	0.601	0.035*	0.006*
Time	<0.001*	0.001*	<0.001*	0.171

In Fig. 7, we report the boxplots arranged by Tool that summarize the PAS and NAS values. We can observe that independently from the values of the Tool variable, the distributions for PAS are higher than the ones for NAS. This seems to indicate that after using Code2City, Code2City$_{VR}$, and Eclipse the participants felt more positive affects than negative affects (Fig. 7.a vs. Fig. 7.b).

By looking at the PAS results depicted in Fig. 7.a, we can observe that there is a noticeable difference between the box of Code2City$_{VR}$ and the one of Eclipse. In particular, the participants administered with Code2City$_{VR}$ 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 Code2City$_{VR}$ and Code2City. That is, it seems that the use of Code2City$_{VR}$ 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 were significant, we ran a Kruskal–Wallis test. We did not run an ANOVA test (as done for Score and Time) because PAS in an ordinal variable. In case of significant differences, we planned to execute Dunn's test [45] to determine which groups in the sample were significantly different (post-hoc analysis).

In Table 8, we report the *p*-value of the Kruskal–Wallis test for PAS (and NAS) and the results of the post-hoc analysis. The result of the Kruskal–Wallis 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.* Dunn's test indicates a significant difference between the distributions of Code2City$_{VR}$ and Eclipse. Therefore, the results of this post-hoc analysis suggest that: *The participants provided with Code2City$_{VR}$ felt significantly better positive affects than those provided with Eclipse.*

As far as NAS is concerned, the boxplots for Code2City, Code2City$_{VR}$, and Eclipse (Fig. 7) overlap one another, although the boxplot for Code2City is shorter than the others. The mean values of NAS are 11.667 for Code2City, 13.235 for Code2City$_{VR}$, and 13.538 for Eclipse. The SD values are, respectively, equal to: 4.271, 3.784, and 5.425. Summarizing, we can observe no huge difference among the distributions of the NAS values for Code2City, Code2City$_{VR}$, and Eclipse, although the NAS values for Code2City are slightly better.

To test the presence of significant differences in the NAS values, we ran a Kruskal–Wallis test (as done for the PAS values). It did not indicate any significant difference in the NAS values (*p*-value is 0.164). Therefore, we could not perform any post-hoc analysis.

Table 8
p-values from the Kruskal–Wallis and Dunn's tests [49][a]. p-value less than α are highliogheted with [*].

Variable	ANOVA	Post-hoc		
		Code2City$_{VR}$-Code2City	Code2City$_{VR}$-Eclipse	Code2City-Eclipse
PAS	0.021*	0.258	0.008*	0.307
NAS	0.164	–	–	–

[a] The p-values returned by the Dunn's test are adjusted by means of the Bonferroni adjustment.

Table 9
Answers to the questions P1, P2, P3, and P4 on the playfulness of Code2City, Code2City$_{VR}$, and Eclipse.

Qn	Tool	Answer						
		1	2	3	4	5	6	7
P1	Code2City	8% ($\frac{1}{12}$)	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	8% ($\frac{1}{12}$)	8% ($\frac{1}{12}$)	17% ($\frac{2}{12}$)	50% ($\frac{6}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	6% ($\frac{1}{17}$)	12% ($\frac{2}{17}$)	6% ($\frac{1}{17}$)	24% ($\frac{4}{17}$)	18% ($\frac{3}{17}$)	35% ($\frac{6}{17}$)
	Eclipse	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	0% ($\frac{0}{13}$)	31% ($\frac{4}{13}$)	23% ($\frac{3}{13}$)	31% ($\frac{4}{13}$)	8% ($\frac{1}{13}$)
P2	Code2City	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	8% ($\frac{1}{12}$)	8% ($\frac{1}{12}$)	33% ($\frac{4}{12}$)	42% ($\frac{5}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	12% ($\frac{2}{17}$)	18% ($\frac{3}{17}$)	18% ($\frac{3}{17}$)	24% ($\frac{4}{17}$)	29% ($\frac{5}{17}$)
	Eclipse	8% ($\frac{1}{13}$)	8% ($\frac{1}{13}$)	8% ($\frac{1}{13}$)	15% ($\frac{2}{13}$)	46% ($\frac{6}{13}$)	15% ($\frac{2}{13}$)	0% ($\frac{0}{13}$)
P3	Code2City	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	17% ($\frac{2}{12}$)	17% ($\frac{2}{12}$)	42% ($\frac{5}{12}$)	17% ($\frac{2}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	12% ($\frac{2}{17}$)	6% ($\frac{1}{17}$)	24% ($\frac{4}{17}$)	24% ($\frac{4}{17}$)	24% ($\frac{4}{17}$)	12% ($\frac{2}{17}$)
	Eclipse	23% ($\frac{3}{13}$)	8% ($\frac{1}{13}$)	8% ($\frac{1}{13}$)	15% ($\frac{2}{13}$)	38% ($\frac{5}{13}$)	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)
P4	Code2City	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$	0% ($\frac{0}{12}$)	25% ($\frac{3}{12}$)	17% ($\frac{2}{12}$)	50% ($\frac{6}{12}$)
	Code2City$_{VR}$	6% ($\frac{1}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	6% ($\frac{1}{17}$)	18% ($\frac{3}{17}$)	18% ($\frac{3}{17}$)	53% ($\frac{9}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	15% ($\frac{2}{13}$)	31% ($\frac{4}{13}$)	46% ($\frac{6}{13}$)	0% ($\frac{0}{13}$)

5.4. *Answers to the post-questionnaire*

In this section, we report the participants' answers to some questions of the post-questionnaire. 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. Since the questions were grouped, in the post-questionnaire, by theme (*e.g.,* playfulness), we present the participants' answers according to these themes.

5.4.1. *Playfulness*

In Table 9, we report how the participants answered to the questions on the playfulness of Code2City, Code2City$_{VR}$, and Eclipse.

The answers to the question P1—*When interacting with Tool, I did not realize the time elapsed*—indicate that most of the participants administered with Code2City and Code2City$_{VR}$ did not realize the passage of time. In particular, 75% and 76% of the participants in the Code2City and Code2City$_{VR}$ groups agreed with the statement of the question P1 (*i.e.,* they gave an answer from 5 to 7). The participants administered with Eclipse seems to agree less with this statement (62% of them gave an answer from 5 to 7).

As for the question P2—*I had fun when fulfilling the program comprehension tasks with Tool*—it seems that those who used Code2City and Code2City$_{VR}$ had more fun than who used Eclipse: 83% and 71% of the Code2City's and Code2City$_{VR}$'s users, respectively, gave a score from 5 to 7, while 62% of the Eclipse's users gave a score from 5 to 7 (no one gave the maximum score).

The use of Code2City and Code2City$_{VR}$ to fulfill the tasks made the participants happier (question P3—*Using Tool to fulfill the program comprehension tasks made me happy*). In particular, 75% and 59% of those who used Code2City and Code2City$_{VR}$, respectively, agreed with the statement of the question P3, while 46% of those administered with Eclipse agreed with this statement.

The answers to the question P4—*Using Tool stimulated my curiosity to explore the program*—suggest that Code2City and Code2City$_{VR}$ stimulated more the curiosity of their users to explore the program with respect to Eclipse. 92% and 88% of the Code2City's and Code2City$_{VR}$'s users, respectively, positively rated this question. 77% of the Eclipse's users positively rated this question but no one gave the maximum score.

Summing up, it seems that the playfulness of Code2City and Code2City$_{VR}$ is mostly comparable, while that of Eclipse is worse.

5.4.2. *Perceived usefulness*

In Table 10, we summarize the answers to the questions on the perceived usefulness of Code2City, Code2City$_{VR}$, and Eclipse.

Based on the answers to PU1—*Using Tool enabled me to accomplish the program comprehension tasks quickly*—the Code2City$_{VR}$'s users believed to accomplish the tasks faster than the Code2City's users and Eclipse's users. In particular, all the Code2City$_{VR}$'s users agreed with the statement of the question PU1 and 47% of them fully agreed; they gave the maximum score. As for the Code2City's users and Eclipse's users, 75% and 92% agreed with this statement. They gave a score from 5 to 7.

The answers to the question PU2—*Using Tool to fulfill the program comprehension tasks increased my productivity*—did not highlight huge differences between the tools. The participants provided with Code2City, Code2City$_{VR}$, or Eclipse agreed that the tool assigned to them increased their productivity. The participants in the Code2City, Code2City$_{VR}$, and Eclipse groups who provided a score from 5 to 7 were 92%, 100%, and 92%, respectively.

As for the question PU3—*Using Tool to fulfill the program comprehension tasks improved their correctness*—the answers indicate that the Code2City$_{VR}$'s users were convinced to improve the correctness of the program comprehension tasks because they used Code2City$_{VR}$ (all the scores were from 5 to 7). This pattern is noticeable neither for Code2City nor Eclipse. Only 67% and 77% of the Code2City's and Eclipse's users, respectively, provided a score from 5 to 7.

The participants provided with Code2City and Code2City$_{VR}$ believed that the tasks were easier (question PU4—*Using Tool made the program comprehension tasks easy*). 83% and 100% of those who used Code2City and Code2City$_{VR}$, respectively, agreed with the statement of the question P4. 77% of the Eclipse's users agreed with this statement and only one participant gave the maximum score (*i.e.,* fully agree).

Table 10
Answers to the questions PU1, PU2, PU3, and PU4 on the perceived usefulness of Code2City, Code2City$_{VR}$, and Eclipse.

Qn	Tool	Answer						
		1	2	3	4	5	6	7
PU1	Code2City	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	25% ($\frac{3}{12}$)	17% ($\frac{2}{12}$)	25% ($\frac{3}{12}$)	33% ($\frac{4}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	24% ($\frac{4}{17}$)	29% ($\frac{5}{17}$)	47% ($\frac{8}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	62% ($\frac{8}{13}$)	23% ($\frac{3}{13}$)
PU2	Code2City	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	8% ($\frac{1}{12}$)	25% ($\frac{3}{12}$)	42% ($\frac{5}{12}$)	25% ($\frac{3}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	24% ($\frac{4}{17}$)	47% ($\frac{8}{17}$)	29% ($\frac{5}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	0% ($\frac{0}{13}$)	15% ($\frac{2}{13}$)	62% ($\frac{8}{13}$)	15% ($\frac{2}{13}$)
PU3	Code2City	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	25% ($\frac{3}{12}$)	25% ($\frac{3}{12}$)	33% ($\frac{4}{12}$)	8% ($\frac{1}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	24% ($\frac{4}{17}$)	53% ($\frac{9}{17}$)	24% ($\frac{4}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	15% ($\frac{2}{13}$)	31% ($\frac{4}{13}$)	38% ($\frac{5}{13}$)	8% ($\frac{1}{13}$)
PU4	Code2City	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	0% ($\frac{0}{12}$)	17% ($\frac{2}{12}$)	8% ($\frac{1}{12}$)	42% ($\frac{5}{12}$)	33% ($\frac{4}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	24% ($\frac{4}{17}$)	35% ($\frac{6}{17}$)	41% ($\frac{7}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	8% ($\frac{1}{13}$)	0% ($\frac{0}{13}$)	15% ($\frac{2}{13}$)	23% ($\frac{3}{13}$)	46% ($\frac{6}{13}$)	8% ($\frac{1}{13}$)

Table 11
Answers to the questions B1 and B2 on the behavioral intention to use Code2City, Code2City$_{VR}$, and Eclipse.

Qn	Tool	Answer						
		1	2	3	4	5	6	7
B1	Code2City	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	17% ($\frac{2}{12}$)	17% ($\frac{2}{12}$)	25% ($\frac{3}{12}$)	8% ($\frac{1}{12}$)	25% ($\frac{3}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	0% ($\frac{0}{17}$)	12% ($\frac{2}{17}$)	24% ($\frac{4}{17}$)	12% ($\frac{2}{17}$)	35% ($\frac{6}{17}$)	18% ($\frac{3}{17}$)
	Eclipse	0% ($\frac{0}{13}$)	0% ($\frac{0}{13}$)	23% ($\frac{3}{13}$)	23% ($\frac{3}{13}$)	31% ($\frac{4}{13}$)	15% ($\frac{2}{13}$)	8% ($\frac{1}{13}$)
B2	Code2City	8% ($\frac{1}{12}$)	8% ($\frac{1}{12}$)	0% ($\frac{0}{12}$)	25% ($\frac{3}{12}$)	0% ($\frac{0}{12}$)	17% ($\frac{2}{12}$)	42% ($\frac{5}{12}$)
	Code2City$_{VR}$	0% ($\frac{0}{17}$)	6% ($\frac{1}{17}$)	18% ($\frac{3}{17}$)	6% ($\frac{1}{17}$)	12% ($\frac{2}{17}$)	29% ($\frac{5}{17}$)	29% ($\frac{5}{17}$)

Summarizing, both Code2City and Code2City$_{VR}$ are perceived as more useful than Eclipse when carrying out program comprehension tasks. The perceived usefulness of Code2City$_{VR}$ is slightly better than that of Code2City.

5.4.3. *Behavioral intention to use*

In Table 11, we report the answers to the questions on the behavioral intention to use Code2City, Code2City$_{VR}$, and Eclipse.

The answers to the question B1—*I would like to use Tool on a regular basis in the future*—indicate that most of the Code2City$_{VR}$'s users would like to use Code2City$_{VR}$ on a regular basis in the future. In particular, 65% of them agreed with the statement of the question B1; they gave a score from 5 to 7. This outcome is less noticeable for Code2City and Eclipse. 58% and 54% of the Code2City's and Eclipse's users gave a score from 5 to 7.

According to the answers to the question B2—*I would like to use Tool rather than an IDE to carry out program comprehension tasks*—the participants administered with Code2City and Code2City$_{VR}$ were mostly favorable to use these tools, rather than an IDE, to carry out program comprehension tasks. In particular, 58% of the Code2City's users and 71% of the Code2City$_{VR}$'s ones agreed with the statement of the question B2.

5.4.4. *Side effects of Code2City$_{VR}$*

In Fig. 8, we summarize the side effects observed by the participants when using Code2City$_{VR}$ (a few participants observed more than one side effect). Only 3 participants (out of 17) did not experience side effects. Most of them started to have headache, nausea, or visual annoyance after using Code2City$_{VR}$.

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$_{VR}$), 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). We can postulate that the difference between the city metaphor implementations and Eclipse is due to how the information needed to accomplish the tasks is provided. In Eclipse the information is shown through views (tables or tree views). This could have penalized the correctness of comprehension tasks of the participants administered with Eclipse. According to the observed results, 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$_{VR}$, or Eclipse. The participants finding Code2City$_{VR}$ spent significantly less time to accomplish the assigned tasks (although many of them experienced some side effects) with respect to who used Code2City or Eclipse. Therefore, we answer RQ2 as follows: *the city metaphor implementation (and Code2City$_{VR}$, 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 seems to support the execution of comprehension tasks better than Eclipse. Both Code2City and Code2City$_{VR}$ seem to aid

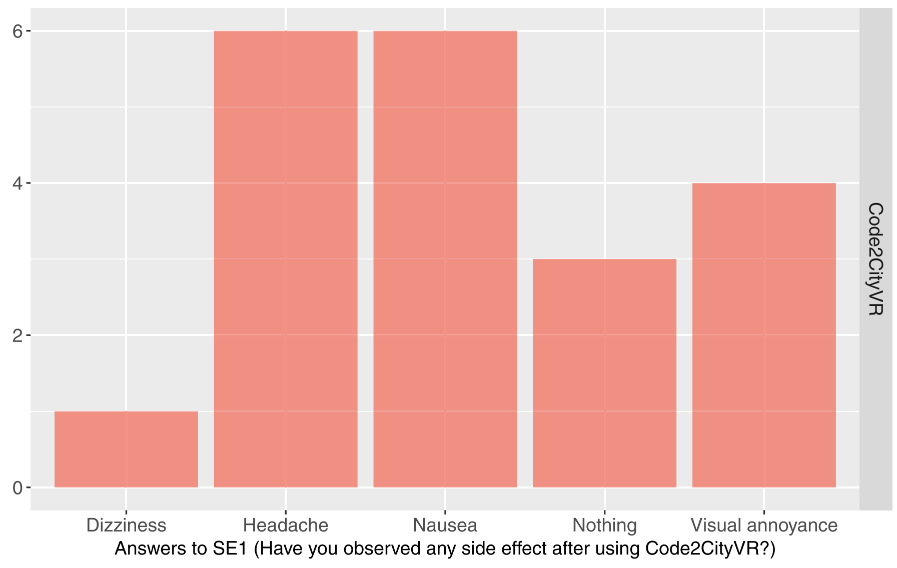


Fig. 8. Answers to the question SE1 on the side effects of Code2City$_{VR}$.

the participants in carrying out program compression tasks better than Eclipse. In addition, the city metaphor implementation displayed on a standard computer screen and in immersive virtual reality is perceived more useful than Eclipse.

The participants in our controlled experiment also found Code2City and Code2City$_{VR}$ more pleasant/enjoyable than Eclipse. Code2City$_{VR}$ (more than Code2City) positively affects the feelings and emotions of the participants. Indeed, the difference between Code2City$_{VR}$ and Eclipse was statistically significant, while between Code2City and Eclipse was not as well as between Code2City$_{VR}$ and Code2City. We can speculate that this outcome could be due to the novelty of the used technology and that, in general, the city metaphor could lead to better feelings and emotions independently from the used medium.

The participants seem to find Code2City$_{VR}$ more useful than Code2City. The results did not show a huge difference between Code2City$_{VR}$ and Code2City from a qualitative perspective, namely playfulness, behavioral intention to use, and positive and negative affects. The obtained quantitative results suggest that the use of Code2City$_{VR}$ (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 a virtual reality display might induce side effects such as headache, nausea, and visual annoyance. That is, a word of warning is needed when using immersive virtual reality in software visualization even because our participants experienced such a kind of side effects (Section 5.4.4). The outcomes of our experiment seem to suggest future research on this point.

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 virtual reality seems to be a viable means in software visualization. Our results confirm the outcomes from past research [11,24] and then our body of knowledge is increased on the usefulness of virtual reality in the software visualization field. Therefore, future research is justified in this respect. Our experiment and past research have then practical implications from both the researcher and the practitioner perspectives.

As compared to Eclipse, the use of Code2City$_{VR}$ 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 virtual reality 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 virtual reality (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.

Code2City$_{VR}$ is designed to be easily adapted to object-oriented programming languages 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 Code2City$_{VR}$ in the context of programs written in other languages.

6.4. Threats to validity

We discuss the threats that could affect the validity of our results with respect to internal, external, construct, and conclusion validity. Despite our efforts to avoid or to mitigate as many threats to validity as possible, some were still unavoidable.

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 [25].
- **Resentful demoralization.** A participant receiving a less desirable treatment might not behave as good as she generally does. For example, a participant receiving the Eclipse treatment could be less motivated. To mitigate this kind of threat, we randomly assigned the participants to the treatment groups and, in addition, the participants were not informed about these groups. Despite our efforts to mitigate this kind of threat, it is still possible that the assignment to the Eclipse group could be viewed negatively.

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 external validity. However, involving students in software engineering experiments has a number of advantages (*e.g.,* having preliminary empirical evidence) [50]. 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 [50,51]. 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. To mitigate this threat, we used the same dependent variables as the experiment by Wettel et al. [2].
- **Hypothesis guessing.** The participants in our experiment might guess the experiment goals and thus behave on the basis of their guesses, although we did not inform them about the goals of our investigation.
- **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 voluntary 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 the ability to draw the correct conclusion about relations between treatments and outcomes:

- **Reliability of measures.** The participants were asked the write down the time needed to carry out the program comprehension tasks. This might affect the validity of the results if they did not report this information truthfully. To mitigate this kind of threat, supervisors took care that the participants correctly wrote down the time to carry out program comprehension tasks that required less than ten minutes (see Section 4.7).

7. Conclusion

We presented the results of a controlled experiment to compare Eclipse (equipped with a plugin to gather code metrics and to identify bad smells) and the city metaphor displayed on a standard computer screen and in an immersive virtual reality with respect to the support provided in the comprehension of Java software systems. 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 a previous investigation and increases the generalizability of the results. We observed that developers that use the implementation of the city metaphor in an immersive virtual reality spent significantly less time to complete the experimental task. The use of the immersive virtual reality medium also leads to higher satisfaction of the participants. These two outcomes suggest that virtual reality 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 virtual reality in software visualization.

Conflicts of interest

We do not have conflicts of interest to declare.

Acknowledgment

Lanza gratefully acknowledges the financial support of the Swiss National Science Foundation for the project “Exploratory Visual Analytics for Interaction Graphs” (SNF NRP Project No. 407540_167173).

Appendix A. Results by task

In Table A.1, we report, for each task, some descriptive statistics for the dependent variables Score and Time.

Table A.1
Some descriptive statistics for each tool and each dependent variable grouped by task.

Task	Statistic	Score			Time		
		Code2City	Code2City$_{VR}$	Eclipse	Code2City	Code2City$_{VR}$	Eclipse
A1	Mean	0.333	0.353	0.154	5.75	3.882	5.846
	Median	0	0	0	5	4	6
	SD	0.492	0.493	0.376	2.989	1.616	2.911
A2.1	Mean	0.883	0.518	0.462	6.75	4.647	6.385
	Median	0.9	0.6	0.6	7	5	6
	SD	0.134	0.425	0.411	1.913	1.367	2.694
A2.2	Mean	1	1	0.923	5.083	3.059	3.769
	Median	1	1	1	4.5	3	3
	SD	0	0	0.277	2.968	1.638	2.279
A3	Mean	0.758	0.688	0.215	6.833	6.412	8.615
	Median	0.95	1	0	6	6	9
	SD	0.342	0.433	0.279	1.586	1.734	1.66
A4.1	Mean	0.875	1	1	4.5	2.706	4.077
	Median	1	1	1	4	3	4
	SD	0.294	0	0	1.977	0.772	1.605
B1.1	Mean	0.417	0.347	0.231	5.083	4.765	8.077
	Median	0	0	0	5	5	9
	SD	0.515	0.485	0.439	0.996	1.48	2.216
B1.2	Mean	0.917	1	0.923	2.583	1.118	2.769
	Median	1	1	1	2	1	2
	SD	0.289	0	0.277	2.575	0.332	2.386
B2.1	Mean	1	0.882	0.846	1.667	0.941	4
	Median	1	1	1	1	1	3
	SD	0	0.332	0.24	1.614	0.243	2

References

[1] A.R. Teyseyre, M.R. Campo, An overview of 3D software visualization., IEEE Trans. Vis. Comput. Graph. 15 (1) (2009) 87–105.

[2] R. Wettel, M. Lanza, R. Robbes, Software systems as cities: a controlled experiment, in: Proceedings of the International Conference on Software Engineering, ACM, New York, NY, USA, 2011, pp. 551–560, doi:10.1145/1985793.1985868.

[3] R. Koschke, Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey, J. Softw. Maint. 15 (2) (2003) 87–109.

[4] R. Wettel, M. Lanza, Visualizing software systems as cities, in: Proceedings of International Workshop on Visualizing Software for Understanding and Analysis, 2007, pp. 92–99.

[5] R. Wettel, M. Lanza, Program comprehension through software habitability, in: Proceedings of International Conference on Program Comprehension, 2007, pp. 231–240, doi:10.1109/ICPC.2007.30.

[6] A. Marcus, L. Feng, J.I. Maletic, 3D representations for software visualization, in: Proceedings of the International Symposium on Software Visualization, ACM, New York, NY, USA, 2003, pp. 27–ff.

[7] T. Panas, R. Berrigan, J. Grundy, A 3D metaphor for software production visualization, in: Proceedings of International Conference on Information Visualization, IEEE Computer Society, 2003, pp. 314–319.

[8] T. Panas, T. Epperly, D.J. Quinlan, A. Sæbjørnsen, R.W. Vuduc, Communicating software architecture using a unified single-view visualization, in: Proceedings of International Conference on Engineering of Complex Computer Systems, 2007, pp. 217–228.

[9] R. Wettel, M. Lanza, Visual exploration of large-scale system evolution, in: Proceedings of Working Conference on Reverse Engineering, IEEE Computer Society, 2008, pp. 219–228.

[10] R. Wettel, M. Lanza, Visually localizing design problems with disharmony maps, in: Proceedings of International Symposium on Software Visualization, ACM, 2008, pp. 155–164.

[11] L. Merino, M. Ghafari, C. Anslow, O. Nierstrasz, CityVR: Gameful software visualization, in: Proceedings of International Conference on Software Maintenance and Evolution, 00, 2017, pp. 633–637, doi:10.1109/ICSME.2017.70.

[12] N. Capece, U. Erra, S. Romano, G. Scanniello, Visualising a software system as a city through virtual reality, in: AVR (2), in: Lecture Notes in Computer Science, vol. 10325, Springer, 2017, pp. 319–327.

[13] L. Merino, A. Bergel, O. Nierstrasz, Overcoming issues of 3D software visualization through immersive augmented reality, in: Proceedings of Working Conference on Software Visualization, 2018, pp. 54–64.

[14] U. Erra, G. Scanniello, M. Caulo, Software systems as archipelagos of atolls, in: Proceedings of the International Conference on Information Visualisation, 2015, pp. 171–176.

[15] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, L. Nafeie, Islandviz: a tool for visualizing modular software systems in virtual reality, in: Proceedings of IEEE Working Conference on Software Visualization, 2018, pp. 112–116, doi:10.1109/VISSOFT.2018.00020.

[16] M. Misiak, D. Seider, S. Zur, A. Fuhrmann, A. Schreiber, Immersive exploration of osgi-based software systems in virtual reality, in: Proceedings of IEEE Conference on Virtual Reality and 3D User Interfaces, 2018, pp. 637–638, doi:10.1109/VR.2018.8446057.

[17] C. Knight, M. Munro, Virtual but visible software, in: Proceedings of International Conference on Information Visualisation, London, England, UK, July 19–21, 2000, 2000, pp. 198–205, doi:10.1109/IV.2000.859756.

[18] S.M. Charters, C. Knight, N. Thomas, M. Munro, Visualisation for informed decision making; from code to components, in: Proceedings of the International Conference on Software Engineering and Knowledge Engineering, ACM, 2002, pp. 765–772.

[19] T. Panas, R. Berrigan, J.C. Grundy, A 3D metaphor for software production visualization, in: Proceedings of International Conference on Information Visualization, 2003, pp. 314–319, doi:10.1109/IV.2003.1217996.

[20] J.I. Maletic, J. Leigh, A. Marcus, G. Dunlap, Visualizing object-oriented software in virtual reality, in: Proceedings of International Workshop on Program Comprehension, 2001, pp. 26–35, doi:10.1109/WPC.2001.921711.

[21] R. Souza, B. Silva, T. Mendes, M. Manoel, SkyscrapAR: an augmented reality visualization for software evolution, in: Brazilian Workshop on Software Visualization, 2012, pp. 17–24.

[22] F. Fittkau, A. Krause, W. Hasselbring, Exploring software cities in virtual reality, in: Proceedings of Working Conference on Software Visualization, 2015, pp. 130–134, doi:10.1109/VISSOFT.2015.7332423.

[23] M. Rudel, J. Ganser, R. Koschke, A controlled experiment on spatial orientation in VR-based software cities, in: Proceedings of Working Conference on Software Visualization, 2018, pp. 21–31.

[24] L. Merino, J. Fuchs, M. Blumenschein, C. Anslow, M. Ghafari, O. Nierstrasz, M. Behrisch, D.A. Keim, On the impact of the medium in the effectiveness of 3D software visualizations, in: Proceedings of International Working Conference on Software Visualization, 00, 2017, pp. 11–21, doi:10.1109/VISSOFT.2017.17.

[25] C. Wohlin, P. Runeson, M. Hst, M.C. Ohlsson, B. Regnell, A. Wessln, Experimentation in Software Engineering, Springer Publishing Company, Incorporated, 2012.

[26] G. Balogh, Á. Beszédes, Codemetropolis – code visualisation in minecraft, in: Proceedings of International Working Conference on Source Code Analysis and Manipulation, IEEE CS Press, 2013, pp. 136–141, doi:10.1109/SCAM.2013.6648194.

[27] A. Bacchelli, F. Rigotti, L. Hattori, M. Lanza, Manhattan-3D city visualizations in eclipse, ECLIPSE IT 2011 (2011) 307.

[28] F.Q.B. da Silva, M. Suassuna, A.C.C. França, A.M. Grubb, T.B. Gouveia, C.V.F. Monteiro, I.E. dos Santos, Replication of empirical studies in software engineering research: a systematic mapping study, Empir. Softw. Eng. 19 (3) (2014) 501–557.

[29] F. Shull, J.C. Carver, S. Vegas, N.J. Juzgado, The role of replications in empirical software engineering, Empir. Softw. Eng. 13 (2) (2008) 211–218.

[30] B. Kitchenham, The role of replications in rmpirical software engineering – a word of warning, Empir. Softw. Eng. 13 (2) (2008) 219–221.

[31] J.C. Carver, N.J. Juzgado, M.T. Baldassarre, S. Vegas, Replications of software engineering experiments, Empir. Softw. Eng. 19 (2) (2014) 267–276.

[32] V. Basili, F. Shull, F. Lanubile, Building knowledge through families of experiments, IEEE Trans. Soft. Eng. 25 (4) (1999) 456–473.

[33] O.S. Gómez, N.J. Juzgado, S. Vegas, Understanding replication of experiments in software engineering: a classification, Information & Software Technology 56 (8) (2014) 1033–1048.

[34] M.T. Baldassarre, J. Carver, O. Dieste, N. Juristo, Replication types: towards a shared taxonomy, in: Proceedings of International Conference on Evaluation and Assessment in Software Engineering, ACM, 2014, pp. 18:1–18:4.

[35] N. Juristo, A. Moreno, Basics of Software Engineering Experimentation, Kluwer Academic Publishers, 2001.

[36] A. Jedlitschka, M. Ciolkowski, D. Pfahl, Guide to Advanced Empirical Software Engineering, Springer London, London, pp. 201–228. doi:10.1007/978-1-84800-044-5_8.

[37] R. Wettel, M. Lanza, R. Robbes, Empirical validation of CodeCity: a controlled experiment, Technical Report 2010/05, University of Lugano, 2010.

[38] D. Watson, L.A. Clark, A. Tellegen, Development and validation of brief measures of positive and negative affect: the panas scales, J. Pers. Soc. Psychol. 54 (6) (1988) 1063–1070.

[39] A. Tony, R. Seewon, H. Ingoo, The impact of web quality and playfulness on user acceptance of online retailing, Inf. Manage. 44 (3) (2007) 263–275, doi:10.1016/j.im.2006.12.008.

[40] S. Romano, N. Capece, U. Erra, G. Scanniello, M. Lanza, The city metaphor in software visualization: feelings, emotions, and thinking, Multimed. Tools Appl. (2019), doi:10.1007/s11042-019-07748-1.

[41] S.S. Shapiro, M.B. Wilk, An analysis of variance test for normality (complete samples), Biometrika 52 (3/4) (1965) 591–611.

[42] M.S. Bartlett, Properties of sufficiency and statistical tests, Proc. R. Soc. Lond. A 160 (901) (1937) 268–282, doi:10.1098/rspa.1937.0109.

[43] H. Arsham, M. Lovric, Bartlett's Test, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 87–88. doi:10.1007/978-3-642-04898-2_132.

[44] J.W. Tukey, Comparing individual means in the analysis of variance, Biometrics 5 (2) (1949) 99–114.

[45] O.J. Dunn, Multiple comparisons using rank sums, Technometrics 6 (3) (1964) 241–252.

[46] H. Abdi, L.J. Williams, Encyclopedia of Research Design, Sage, Thousand Oaks. doi:10.4135/9781412961288.n181.

[47] R. Lincke, J. Lundberg, W. Löwe, Comparing software metrics tools, in: Proceedings of the 2008 International Symposium on Software Testing and Analysis, in: ISSTA '08, ACM, New York, NY, USA, 2008, pp. 131–142, doi:10.1145/1390630.1390648.

[48] F.A. Fontana, E. Mariani, A. Mornioli, R. Sormani, A. Tonello, An experience report on using code smells detection tools, in: Proceedings of the 2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops, in: ICSTW '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 450–457, doi:10.1109/ICSTW.2011.12.

[49] O. Dunn, Multiple comparisons among means, J. Am. Stat. Assoc. 56 (293) (1961) 52–64, doi:10.2307/2282330.

[50] J. Carver, L. Jaccheri, S. Morasca, F. Shull, Issues in using students in empirical studies in software engineering education, in: Proceedings of International Symposium on Software Metrics, in: METRICS '03, IEEE, Washington, DC, USA, 2003, p. 239.

[51] M. Höst, B. Regnell, C. Wohlin, Using students as subjects—a comparative study of students and professionals in lead-time impact assessment, Empir. Softw. Eng. 5 (3) (2000) 201–214, doi:10.1023/A:1026586415054.