Visualizing Code Smells: Tables or Code Cities? A Controlled Experiment

Falko Galperin
University of Bremen, Germany
falko1@uni-bremen.de

Rainer Koschke
University of Bremen, Germany
orcid.org/0000-0003-4094-3444

Marcel Steinbeck
University of Bremen, Germany
marcel@informatik.uni-bremen.de

***Abstract*—This paper presents a study in which we compared the visualization of code smells in Code Cities with classical tabular representations. We conducted a controlled experiment with 20 participants who had to solve six tasks in both environments. We evaluated the results of our experiment statistically and came to the following conclusions: In four tasks the completion time was significantly lower when using Code Cities (the remaining two tasks did not show any statistically significant differences for any of the two environments). Also, the perceived effort was significantly lower in three tasks when the participants used Code Cities over the tabular representation (again, the remaining tasks did not show any statistically significant differences). However, with regard to the perceived usability (which was measured across all tasks) and the correctness of the supplied answers, the tabular representation performed better. In particular, in two tasks the correctness was significantly better in the tabular environment and in one task the correctness was significantly better in the Code Cities environment. Based on our results, our verdict is as follows: Code Cities are better suited to get a quick overview of the code smells of a software, whereas tabular representations are better suited to analyze code smells in more detail.**

Video URL: https://youtu.be/dGY8RFn5WvA (see the video description for more details)

***Index Terms*—software visualization, code cities, tabular representation, code smells, controlled experiment**

I. Introduction

Software visualization is a tried and tested means of emphasizing the characteristics of a software visually. Researchers have developed a variety of visualization concepts in the last years, each focusing on certain aspects of software. A technique that has gained popularity in recent years—and which is still subject to current research—is the *Code City* visualization [1]–[26]. Code Cities are based on the *software-as-a-city* metaphor and depict the elements of a software (e.g., its source-code files, modules, components, etc.) as three-dimensional objects—usually blocks, but other kinds of shapes are also possible—on a two-dimensional plane. The metrics measured for the elements of a software to be visualized are mapped to the size of the corresponding blocks (width, height, and depth, respectively), creating the impression of a city with differently formed buildings. The spatial distribution of the depicted elements (and the metrics measured for them) that builds up in Code Cities is one of the key strengths of Code Cities, because it allows users to create a mental map of the visualized software. To this regard, specific traits in the arrangement of the blocks can be set through different kinds of layouts, such as in, for example, EvoStreets [27], [28], a Code Cities layout with a special emphasis on the visualization of software evolution.

By mapping metrics to visual properties, human beholders can easily identify peculiarities regarding these metrics. For example, if the metric *lines of code* is mapped to the height of the blocks of a city, particularly large and particularly small blocks immediately catch the eye of the users—the same is true for the width and depth of the blocks. Therefore, Code Cities are well suited to assist developers in software maintenance tasks [29], [30].

Software maintenance encompasses a wide range of activities. Our research focuses on the identification and elimination of code smells. Although code smells do not necessarily imply faulty code, they indicate bad practices which may have negative effects on the maintainability of software. Code smell management plays an increasingly important role in modern software development and so it is not surprising that a variety of tools for the automatic detection of code smells have been developed in the last years. At the same time, most tools present their findings by tables, a visualization form which, on the one hand, allows easy sorting and filtering, but, on the other hand, does not utilize the human visual perception to the same extent as Code Cities.

Contributions: We conducted a controlled experiment to answer the question whether tabular representations or Code Cities are better suited for the analysis of code smells in software systems. In our experiment, 20 participants were given a set of six tasks, of which one half of the tasks had to be solved in one environment (e.g., the tabular representation) and the other half of the tasks had to be solved in the other environment (e.g., by means of Code Cities)—the allocation of tasks and environments was chosen randomly, but was evenly distributed among the participants.

The question of whether Code Cities are better suited for analyzing software than other forms of presentations is not new in the field of research. Wettel et. al. conducted a similar experiment [29], [30] in which participants examined two software systems—*FindBugs* and *Azureus*—with regard to various maintenance concerns (split into *program comprehension* and *design quality assessment*). The study of Wettel et. al. is based on the Code Cities implementation *CodeCity* [31] developed by Wettel and Lanza [32], [33]. The baseline to that was a combination of the integrated development environment *Eclipse* and the spreadsheet software *Microsoft Excel*. The authors decided to go with Excel spreadsheets as a comparison system to *CodeCity* because they could not find a suitable Eclipse plug-in that “fit [their] requirements (including support for user defined metrics)” and did not want to “confer an unfair data advantage to the subjects in the experimental group” [30]. Our study represents an innovation in this research area as we compare Code Cities with additional decorations for code smells to a professional integrated dashboard application—the *Axivion Suite* [34]—which is specially designed for showing code smell data. For example, with the *Axivion Suite*, unlike with Excel spreadsheets, users can easily jump between the list of code smells findings in the tabular view and the source code containing the code smells (and vice versa). Thus, in our study two fully integrated systems for presenting code smell data are compared with each other.

Similar to the experiment of Wettel et. al., we measured the performance of the participants with regard to *correctness* and *task completion time*. However, we also measured how the participants personally felt about both systems using the *System Usability Scale* (SUS) [35] (a *post-study* questionnaire which measures the perceived usability) and the *After-Scenario Questionnaire* (ASQ) [36] (a *post-task* questionnaire which we used to measure perceived effort and perceived difficulty). Our study therefore comprises not only the measurement of performance data but also the participants’ subjective perception with respect to the compared environments.

Outline: The remainder of this paper is structured as follows. Section II presents related research. Section III delves into our design decisions on how to visualize code smells in our software visualization platform SEE (for **S**oftware **E**ngineering **E**xperience). Section IV details how the controlled experiment was structured and performed. Section V analyzes the results of the experiment. Section VI concludes.

II. Related Research

This paper focuses on the visualization of code smells in software systems using Code Cities. For a comprehensive overview on clone visualization (the code smell considered most harmful by Fowler et. al. [37]), we refer the interested reader to the work of Hamad et. al. [38]. In the following section, we present related research on the topic of Code Cities and the comparison of different Code City environments—that is, studies comparing different hard- and software setups.

A. Code Cities

In their simplest form, Code Cities are an extension of *Tree-maps* [39] in 3D space, which, in turn, are an early attempt to express quantitative data (i.e., metrics) over a hierarchy. With Tree-maps, a hierarchy (whatever it is composed of) is depicted by recursively subdividing a planar shape (e.g., a rectangle) into smaller nested planar shapes—that is, the hierarchy is expressed by spatial inclusion. The area of the innermost shapes corresponds to a certain metric (e.g., lines of code, complexity, etc. for software), expressing the metric as a proportion of the available space. This approach allows humans to easily identify atypical patterns in a special context, and so it is not surprising that quickly the idea came up to express an additional metric by mapping its value to the height of the shapes. If the underlying shape is a rectangle, resulting in cuboids in 3D space (which is generally the most commonly used shape), the obtained visualization reminds of North American downtowns with buildings arranged in grids of blocks. Accordingly, such three-dimensional Tree-maps were called *Code Cities* [40]. Researchers quickly adopted the idea of Code Cities and have been using them to visualize a variety of aspects of software [1]–[26].

In addition to the geometric extensions (width, height, and depth) of the buildings of a Code City, colors can be used to express yet another metric by mapping the metric values to color gradients (e.g., from green to red) and applying the gradients on the surface of the buildings [16], [19]. Sizing and coloring the representation of the elements of a software to be visualized is well suited to express several metrics—up to four—for individual elements at once. However, relations between elements (e.g., include dependencies) cannot be represented appropriately by these means. To depict relations, edges connecting the related elements are therefore often used [41]. To diminish the visual clutter that can occur when visualizing a lot of (overlapping) edges, hierarchical edge bundles have proven themselves suitable [42]–[44].

As already mentioned, Code Cities, in their simplest form, are Tree-maps in 3D space. That said, Code Cities today vary also in their layout. Examples for other Code City layouts include *EvoStreets* (a layout with a special emphasis on the visualization of software evolution) [27], *Circular Balloon* (a layout which, due to the more generous use of space, makes it easier to grasp the hierarchical structure of the visualized elements) [22], and *Circle Packing* [22] or *Rectangular Packing* [6]–[8]—two layouts where the hierarchy is expressed by nested circles or rectangles, respectively.

B. Comparing Different Environments

In Section I, we already discussed the controlled experiment of Wettel et. al. [29], [30]. Besides the comparison of Code Cities with tabular representations, there are also studies focusing on comparing Code Cities in two-dimensional, pseudo-3D (2.5D), and virtual reality (VR) environments [19]–[21]. There is already a great body of knowledge on 2.5D and VR visualization outside of the software visualization community [45]–[47]. Studies have shown that head-mounted displays may have a positive effect on the orientation of users in VR environments [48]. In the hope that these advantages also apply to software visualization, researchers have begun implementing Code Cities in VR and augmented reality (AR) environments [13], [23], [49], [50].

III. Design Decisions

The aim of our research is to create a multi-functional software visualization based on the software-as-a-city metaphor. To this end, we are developing the visualization platform SEE (for **S**oftware **E**ngineering **E**xperience). Our primary goal is to provide a consistent environment wherein Code Cities form the basis for at least the following use cases:

Visualization of software metrics: Metrics measured for the elements of a software can be depicted by mapping the metric values onto the width, height, depth, and colorization (color gradients, textures, and the like) of the blocks representing the elements. Relations between elements can be expressed using edges (and hierarchical edge bundles).

Monitoring software evolution: The source code of software is typically managed in version control systems (VCS), such as *git*, *subversion*, and *mercurial*. Such VCS can be analyzed using software repository mining techniques. SEE makes use of the open-source library *LibVCS4j* [51] (which is also developed by our research group) to create a linear change model of the source code of a software which can then be viewed as animated Code Cities in SEE. This allows users to examine the development history of a software and to get a better understanding when code smells were introduced and how they evolved.

Debugging of software: In order to fix errors in existing source code, it is often necessary to debug software. SEE assists developers in debugging software by visualizing program execution traces interactively. Function calls are depicted as edges connecting the source and target of the calls. The values of variables of the currently active frame (function) can be inspected in a dedicated code window (similar to how source code is debugged in modern integrated development environments). The program traces to be visualized are recorded in advance, allowing users to debug software both forwards and backwards.

Software architecture: Within the virtual environment of SEE, users can model the architecture of a software at component level by creating nodes (representing the expected components) and grouping them hierarchically. The nodes arranged this way build up a Code City. Afterwards, the actual implementation of the software can be imported as yet another Code City. With both Code Cities placed next to each other in the virtual environment, the users can then map implementation components from the implementation city onto architecture components in the architecture city (by dragging the nodes from one city to the other). Using the reflexion model [52], the specified architecture is continuously checked against the actual implementation [25]. Convergences, divergences, and absences are depicted by differently colored edges.

All these use cases should be supported by a uniform visualization. Non-uniform visualizations would require additional cognitive effort when a user switches from one use case to another one. Uniform here means that the meaning of a block in a Code City, its dimensions, the spatial enclosure, and the layout should be the same for those use cases. Because in all these use cases, components of the software arranged in a hierarchy are the central unit of thought, it is quite natural to depict those by means of blocks and spatial enclosure. Uniformity across the use cases also dictates that the metrics chosen to determine the dimensions of the blocks are the same; otherwise their sizes might change radically between use cases likely leading to disorientation of the beholder. Hence, information specific to a use case must be added by other means.

For these reasons, we refrained from mapping the code smell information onto the dimensions of a block. Neither are we using color to highlight code smells because color is generally used for other purposes such as the type of the node (e.g., a field versus a method), another node metric or simply a user selection. Accordingly, we decided to visualize code smells by counting the number of findings (which yields a metric) and mapping those counts as follows: Each block in the city represents a component (e.g., a file), which may be affected by code smells. The number of findings for each kind of smell is depicted as a dedicated icon and placed above the center of the corresponding block's roof. As icons we chose the same icons by which the professional tool that we use as a baseline in our experiment (the *Axivion Dashboard*) depicts the code smells it detects, for instance, a pen is used for code-style violations and a triangle ruler is used for exceeding metric values.

The red tint of an icon is proportional to the relative occurence of instances of that kind of code smell within its level, whereas a level simply consists of all nodes with the same distance to the root node. In other words, the redder an icon is, the more code smells of that type occur within that level of the Code City's underlying hierarchy. This visualization of stacked icons above affected nodes creates the impression of a rising cloud of smoke. Figure 1 shows an example. If a block is hovered over, the exact number of smell instances appears next to the smell type's icon to show details. This follows Shneiderman's mantra and allows users to get a quick overview of how many code smells of which kind were detected and then to dig into the details of the exact numbers.

Inner nodes of the hierarchy may also have code smells. They may simply aggregate the smells of their descendants but may even have their own additional smells (e.g., circular dependencies among packages). Inner nodes are drawn as either rectangular or circular areas depending upon the chosen layout. Uniformly to blocks representing leaves in the hierarchy, the smell icons will be placed at the center of their area.

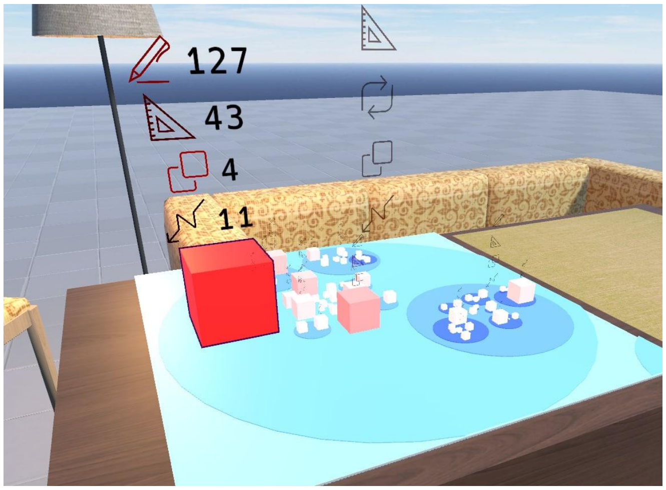
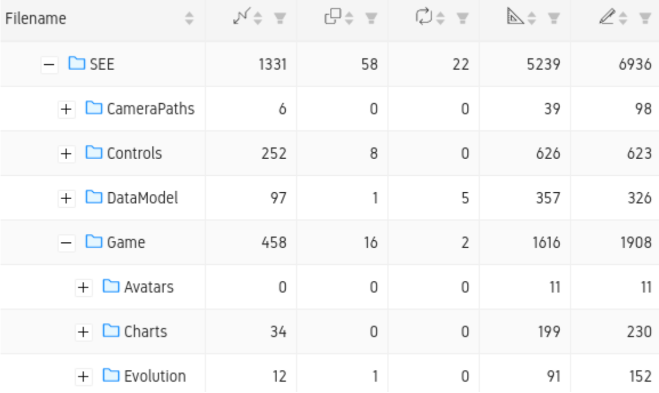


Fig. 1: Visualizing code smells in SEE.

IV. Controlled Experiment

We wanted to empirically compare the visualization of code smells in SEE to a more traditional tabular representation in terms of task accuracy (correctness), task efficiency (effort), and how satisfied the users were with the visualization (perceived usability). This section describes our controlled experiment in detail.

As a tool to gather the code smells we visualize in SEE, we already used the *Axivion Suite* [34], which presents its findings in a web-based dashboard using tabular tree views (cf. Figure 2). For these reasons, we decided to use the tabular views in the *Axivion Dashboard*—or simply *Dashboard* for short in the following—as the baseline in our experiment. That also means that the data provided by SEE and the tabular view are identical, only the presentation differs. The *Axivion Suite* groups its findings into six types of code smells: duplicated code, cyclic dependencies, dead code, architecture violations, code-style violations, and excessive metric values.



Filename					
− SEE	1331	58	22	5239	6936
+ CameraPaths	6	0	0	39	98
+ Controls	252	8	0	626	623
+ DataModel	97	1	5	357	326
− Game	458	16	2	1616	1908
+ Avatars	0	0	0	11	11
+ Charts	34	0	0	199	230
+ Evolution	12	1	0	91	152

Fig. 2: Visualization of code smells in the *Axivion Dashboard*.

A. Independent Variables

The independent variable that we vary in the experiment is the use of the visualization in SEE and the tabular representation of the *Dashboard*. In addition, we are using different tasks (see Section IV-D and Figure 3). Other independent variables that may influence the dependent variables that we do not control are the characteristics of the participants (see Section IV-F) and the type of system whose code smell data are visualized (see Section IV-D).

B. Dependent Variables

The dependent variables were gathered while our experimental subjects solved particular tasks (see Section IV-D). To measure the effort, we simply measured the time they needed to finish a task, referred to as **task completion time** in the following. **Correctness** was measured as the percentage of correct answers provided in a task. Task completion time and correctness are classic measures in controlled experiments in software visualization providing objective quantitative insights into the usability of a visualization regarding effectiveness and efficiency.

Complementary to that we also wanted to get a better understanding on the user experience of the visualizations, that is, how users perceived the usage of a visualization for a particular set of tasks. Perceived usability is subjective and more difficult to capture. Usability research has developed several ways how to gather it. One is to simply ask questions about the users' experience in using a given program. Research in this field has developed different questionnaires for this purpose. Of course, such questions target the conscious mind only and may be answered subjectively and within cognitive limits (e.g., short-term memory). As an alternative, usability research has also proposed to gather bio signals such as body temperature, blood pressure, or EEG sensing possible stress indicators. Measuring these is, however, quite invasive, requires special equipment, and is often difficult to interpret.

We neither have the necessary equipment nor the expertise in interpreting bio signals and foremost preferred to have a non-invasive way to gather our data, hence decided to opt for a questionnaire. Yet, there are many different usability questionnaires one can choose from. A quite comprehensive overview is provided by Assila et al. [53]. We do not have the room to present them all. For this reason, we just state which one we considered and then finally selected and summarize the reasons for this decision here. All our participants were Germans, hence, the questions were asked in German. For this reason, we favored questionnaires whose translation into German was validated, which further limited the number of available questionnaires. Among those we considered initially, namely, NASA-TLX [54], RSME [55], SMEQ [55], SEQ [56], ASQ [36], SUMI [57], QUIS [58], PSSUQ [59], and SUS [35], we selected ASQ and SUS.

ASQ (for after-scenario questionnaire) is a post-task questionnaire, that is, will be filled out after each task completed. It consists of three questions asking whether the user was satisfied with the simplicity, required time, and provided supporting information [36] on a Likert scale from 1 (strongly disagree) to 7 (strongly agree). Because neither SEE nor the *Dashboard* provides accessible supporting information immediately within the visualization itself (the latter provides this information in a separate online user manual; SEE does not even have a user manual yet) and also because supporting information is not in the scope of our evaluation, we dropped the third question regarding supporting information (as done by others in similar studies before [60]). In summary, the statements of ASQ we gathered opinions from the participants were: (1) "Overall, I am satisfied with the ease of completing the task in this scenario" and (2) "Overall, I am satisfied with the amount of time it took to complete the task in this scenario", where we actually used a translated German version by Funk et al. [61]. ASQ was chosen because it makes a distinction between the perceived cognitive difficulty (ease) and the perceived expenditure of time unlike most other questionnaires. In addition, it complements our objective measure of task completion time very well and asks only two questions so that an experimental subject can move quickly to the next task (each had to solve six tasks).

As a post-study questionnaire, that is, one that is asked when all tasks for one particular visualization have been completed, we used the *System Usability Scale (SUS)* [35], which asks ten generic, rather similar questions about the perceived usability, thereby balancing the number of positive (affirmative) and negative (dissenting) questions. SUS is widely used [62], allows to compare two visualizations [63], [64], robust with respect to validity, sensitivity, and reliability and widely applicable [63]–[65], free of charge [35], and has a validated German translation [66]. Respondents to the SUS questionnaire can state their degree of agreement to each of the ten questions on a Likert scale from 0 (strongly disagree) to 4 (strongly agree). These values can be summed up where the values for negatively asked questions are inverted. The total sum is multiplied by a factor 2.5 so that the final value, referred to as the *SUS score*, may be in range from 0 to 100. A SUS score above 68 would be considered above average [67].

In summary, our dependent variables are:

Correctness = the percentage C of correct answers of a task (the higher the better)
Task completion time = time t required to solve a task (the lower the better)
Overall perceived system usability = *SUS* score S (the higher the better)
Perceived task difficulty = value A^d for ASQ statement (1) (the lower the better)
Perceived task effort = value A^e for ASQ statement (2) (the lower the better)

C. Hypotheses

Given the measures for the dependent variables, we can now postulate our null hypotheses the experiment attempts to falsify where the index S relates to the respective value for SEE and the index D to the *Dashboard*. All of them are directed and stated in the favor of the tabular representation as follows:

$$H_{0,C} : C_S \leqslant C_D \qquad H_{0,t} : t_S \geqslant t_D \qquad H_{0,S} : S_S \leqslant S_D$$
$$H_{0,A^e} : A^e_S \geqslant A^e_D \qquad H_{0,A^d} : A^d_S \geqslant A^d_D$$

D. Tasks

To ensure the experiment's tasks are representative, *Axivion* employees have been asked for typical usage scenarios for the *Dashboard*. Based on their answers, three tasks have been designed around the general scenario of identifying god classes. The first two tasks (A and B) are about first finding god classes within a given directory and then determining how many instances of a certain code smell type exist for each of those god classes, while the third task (C) is about figuring out the most common code smell type for each of the five *given* biggest files within a directory. The project that was to be analyzed by participants was the source code of SEE, which consists of around 50,000 lines of C# code and 9,000 nodes in the dependency graph among which are 479 representing classes (others are interfaces, methods, fields, or namespaces).

The participants solved each of the three tasks once with the *Dashboard* and once with SEE, though the order in which the two systems are used has been randomized to counteract any influence the ordering may have. Additionally, while the tasks as described above stay the same for their second iteration, the directory that is to be analyzed will be different. Otherwise participants would already know the answer from the first iteration. The directories have been chosen such that they are comparable in their relevant attributes between the two iterations. We call the group that first worked with SEE and then with the *Dashboard* **group $\boldsymbol{\alpha}$**, and call the other one **group $\boldsymbol{\beta}$**. We use the subscript **1** (e.g., A_1) for the first iteration of a task and **2** for the second. Figure 3 illustrates the flow of the experiment from a participant's perspective.

We differentiate between architecture violations (Task A) and style violations (Task B_1) / metric violations (Task B_2), because the latter two correlate much more strongly with each other in their number of occurences ($r \approx 0.66$) compared to architecture and style violations ($r \approx 0.24$) or architecture and metric violations ($r \approx 0.13$). The other three code smell types were excluded due to how little they occurred in SEE's source code.

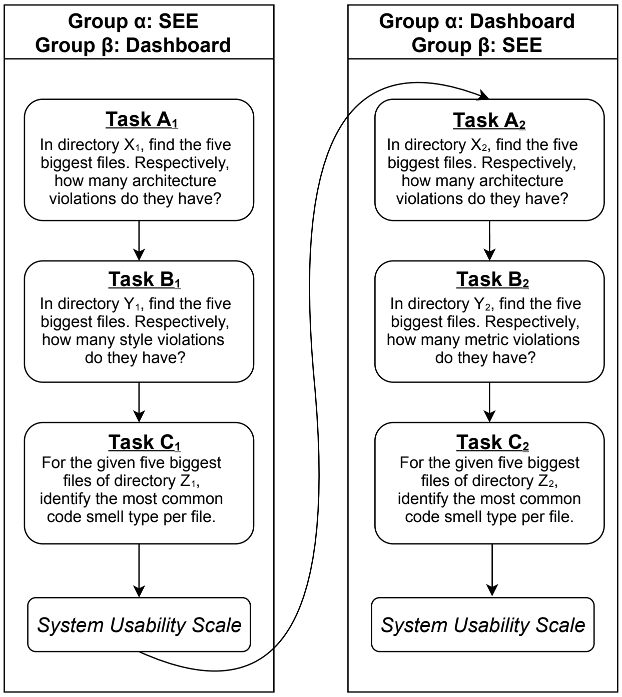

Group α: SEE
Group β: Dashboard
Task A₁
In directory X₁, find the five biggest files. Respectively, how many architecture violations do they have?
Task B₁
In directory Y₁, find the five biggest files. Respectively, how many style violations do they have?
Task C₁
For the given five biggest files of directory Z₁, identify the most common code smell type per file.
System Usability Scale
Group α: Dashboard
Group β: SEE
Task A₂
In directory X₂, find the five biggest files. Respectively, how many architecture violations do they have?
Task B₂
In directory Y₂, find the five biggest files. Respectively, how many metric violations do they have?
Task C₂
For the given five biggest files of directory Z₂, identify the most common code smell type per file.
System Usability Scale

Fig. 3: Flow of the tasks the participants worked on. A^e and A^d were gathered after each completed task.

E. Conducting the Experiment

The study was conducted online asynchronously—participation worked by simply opening a link which randomly assigned the participant to group α or β. Participants then had to answer a preliminary questionnaire asking about basic demographic attributes (such as age and gender) as well as prior experience with the two evaluated systems (as reasoned by Mclellan et al [68]) and with software development and video games in general because SEE uses paradigms of modern 3D video games, so experience may help. Afterwards, participants could download SEE or access the *Dashboard* and were presented with the tasks and corresponding questionnaires as shown in Figure 3. On average, participation took around 45 minutes in total.

Four people participated in a pilot study before we conducted our main study. As a result of feedback given by those people, a few complaints about SEE's controls were addressed and we switched from the *Single Ease Question (SEQ)* to the aforementioned *After Scenario Questionnaire (ASQ)* so that respondents can differentiate between *cognitive difficulty* and *effort*, as opposed to grouping the two concepts into the single question of SEQ.

F. Experimental Subjects

For the main study itself, we found our $n = 20$ participants—ten for group α, ten for group β—using simple convenience sampling (none of those participated in the pilot study). More specifically, we had three distinct types of participants:

1) ***Axivion* employees**, who are presumed[1] to have prior experience with the *Dashboard*.
2) **SEE students**, i.e., students who helped develop SEE and thus are presumed to have prior experience with it.
3) **CS students**, who are presumed to have no prior experience with any of the two systems.

In theory, a fourth partition is missing—namely, people who have experience with both SEE and the *Dashboard*—but the two people known to belong to this category have already participated in the pilot study and thus could not also participate in the actual study.

V. Results

In this section, we will compare demographic attributes measured by our preliminary questionnaire, as well as the previously mentioned aspects for each task—correctness, task completion time, usability, perceived effort, and perceived difficulty—between group α and group β. As such, we will always compare these aspects between the exact same task, the only difference being whether it was solved using SEE or the *Dashboard*. We will also analyze the effect of experience on these aspects and discuss threats to validity on the results.

To check for significant differences between the two groups, we use the Mann-Whitney-U test, which does not assume any specific distribution. Our significance level will be $\alpha = 0.05$. When visualizing any distribution of values, we are using an auto-generated [69] violin plot combined with a box plot, in which the median is shown and any (automatically detected [70]) outliers are pointed out in red.

A. Preliminary questionnaire

In comparing various demographic attributes—namely age, gender, highest attained degree, programming experience, experience in bigger software projects, and experience with video games—we found no significant differences between group α and β. Similarly, there were no significant differences in the levels of experience with SEE and the *Dashboard*. Thus, these confounding factors should not influence any comparison between the two groups.

B. Correctness

In this part, we analyze the correctness C—that is, the percentage of answers that are correct—for each task and compare it between SEE and the *Dashboard* to examine the null hypothesis $H_{0,C}$. The correctness was determined by comparing participants' answers with the solution for each task. This was done manually because we do not want to count typographical errors or subsequent errors (i.e., the correct number of code smells given for the wrong file). For the tasks C_1 and C_2 five numbers each had to be stated. Each of the other tasks asked for five filenames and their corresponding number of violations, summing up to ten data points requested from the respondents for that task. If a filename was correct, but the wrong number of violations was stated, this counted as one correct and one incorrect data point. If a filename was wrong, but the correct number of violations of that wrong filename was given, this counted as one incorrect and one correct data point.

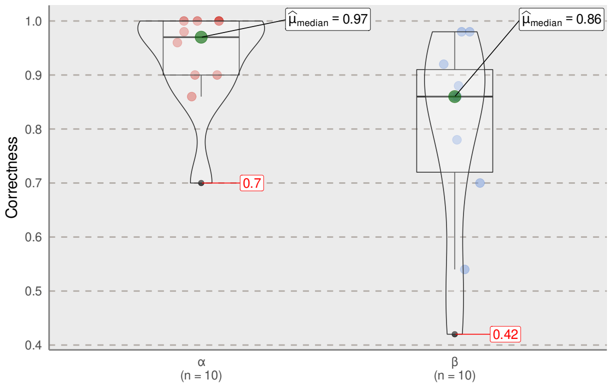


Fig. 4: Comparison of the correctness between the participants of the two groups (not between SEE and the *Dashboard*).

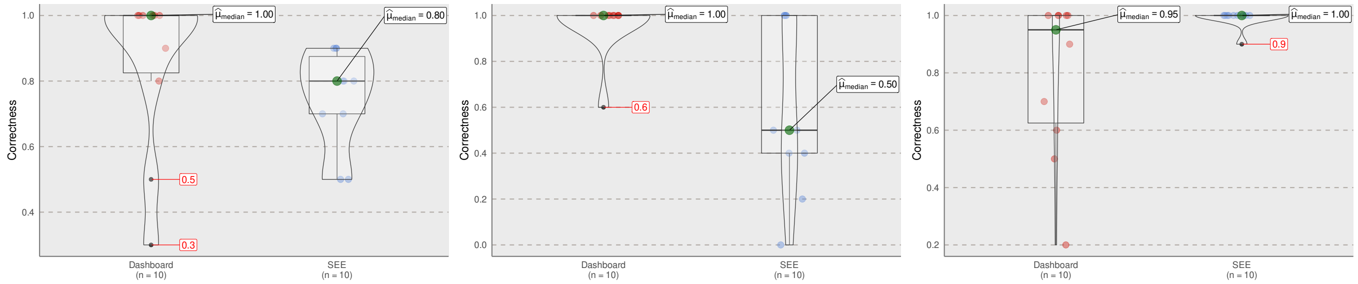
Before comparing the results of each task, we compared the average correctness of the participants across the two groups in total (see Figure 4), and found that group β had a significantly lower correctness (median 0.86) than group α (median 0.97) as measured by the Mann-Whitney-U test ($p \approx 0.03$). There are several possible reasons for this difference:

- Some participants in group α coincidentally performed better on the tasks than group β.
- The directories used in the first three tasks A_1, B_1, C_1 were easier to solve using SEE than the *Dashboard*, or vice versa for the tasks A_2, B_2, C_2.

Again using the Mann-Whitney-U test to compare the correctness for each task (thus comparing between participants using the *Dashboard* and participants using SEE), we find the following significant differences:

- **Task A_1:** Participants using the *Dashboard* achieved a higher correctness than those using SEE ($p \approx 0.03$).

[1] Note though that all participants were still asked about prior experience with both SEE and the *Axivion Dashboard* in the introductory questionnaire.



(a) Correctness for task A_1.

(b) Correctness for task B_1.

(c) Correctness for task A_2.

Fig. 5: Correctness compared between the two systems for each task. Only tasks with significant differences are included.

- **Task B_1:** Participants using the *Dashboard* achieved a higher correctness than those using SEE ($p \approx 0.01$).
- **Task A_2:** Participants using SEE achieved a higher correctness than those using the *Dashboard* ($p \approx 0.02$). However, due to getting too many ties (see Figure 5c) on this result, which is suboptimal for a rank-based measure such as the Mann-Whitney-U test, our U value is not below the critical value, even though the p-value is. This is because we are using *mid-ranks* to correct for the ties; the problem here is that, for percentages of ties higher than 15% (in our case, it is at 70%), permutation tests are recommended over mid-rank corrections [71, pp. 15–16].
Thus, we have repeated the comparison using an appropriate test—in our case the Fisher-Pitman permutation test [72, pp. 228–231]—which gives us the same result as the Mann-Whitney-U test: The participants using SEE achieved significantly higher correctness than those using the *Dashboard* ($p \approx 0.02$).

Note that tasks not listed did not bring out any significant difference. Overall, in two tasks the *Dashboard* achieved higher correctness while SEE did so in only one task. As such, we **cannot reject** $H_{0,C}$; this rather suggests the opposite, namely that the *Dashboard* leads to a slightly higher level of correctness than SEE ($C_S < C_D$). The results for correctness are visualized in Figure 5. Tasks A_x and B_x have two potential sources of error: (1) finding the biggest files and (2) stating the right numbers of violations. For the *Dashboard*, the overall correctness for (1) is 83.5 % and for (2) it is 90.5 %, whereas for SEE it is 76.0 % for (1) and 88.5 % for (2). These numbers indicate that the more likely source of error in SEE is finding the biggest files. Apparently, sometimes participants could not visually assess the file size from the block dimensions correctly.

C. Task completion time

In this part, we analyze the time, t, a participant required to solve a particular task to examine our null hypothesis $H_{0,t}$. We measured this time by having the participant click on a button which measures the current time and at the same time reveals the actual task so they can work on it. At the end of the task (i.e., when the user clicks the “Submit” button), the time is measured again and the user cannot change any submitted answer anymore. The *task completion time* t is simply the difference between these two points in time.

Again comparing between the two *groups* α and β we find no significant difference and move on to comparing the two *systems* for each task, only listing significant differences:

- **Task A_1:** Participants using SEE solved the task faster than those using the *Dashboard* ($p \approx 10^{-5}$).
- **Task B_1:** Participants using SEE solved the task faster than those using the *Dashboard* ($p \approx 0.008$).
- **Task A_2:** Participants using SEE solved the task faster than those using the *Dashboard* ($p \approx 0.007$).
- **Task B_2:** Participants using SEE solved the task faster than those using the *Dashboard* ($p \approx 0.003$).

Overall, we find a very significant effect ($p < 0.01$ for all differences): For four of the six tasks, it takes less time to solve them with SEE than with the *Dashboard*. For the remaining two tasks, no statistically significant difference has been found. These results are visualized in Figure 6. Thus, we can **reject** $\mathbf{H_{0,t}}$ in four out of six cases and conclude that those tasks were more quickly solvable with SEE than the *Dashboard* ($t_D > t_S$).

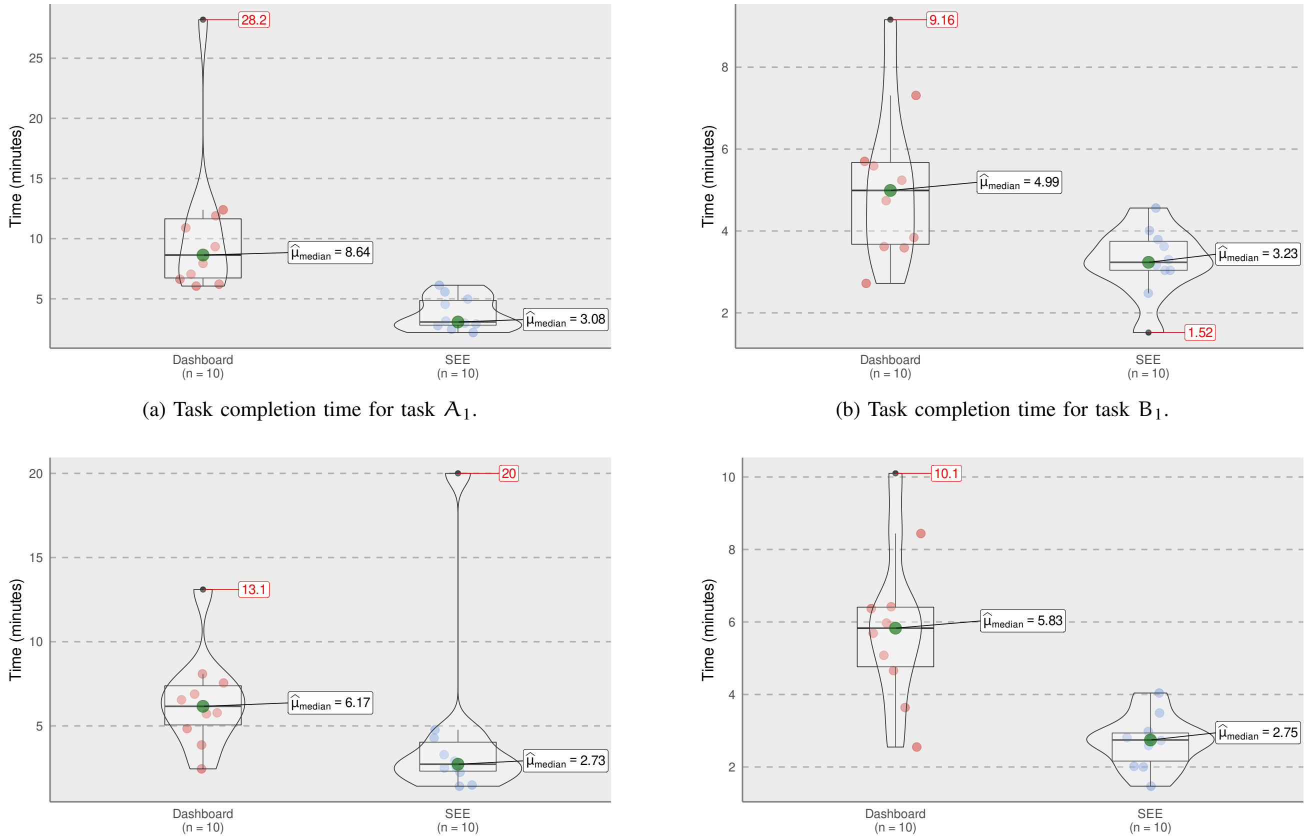
However, it must also be noted that (as said in Section V-B) the first two tasks (A_1 and B_1) were accompanied by a significantly lower correctness in SEE compared to the *Dashboard*. This is not the case for tasks A_2 and B_2, though, so our conclusion should still hold overall.

D. Usability

In this section, we will examine our final null hypotheses $H_{0,S}$ (overall usability), H_{0,A^e} (perceived effort) and H_{0,A^d} (perceived difficulty) by analyzing the answers to the post-study SUS and the post-task ASQ, respectively.

As before, we have also checked whether there is a significant difference between the two *groups* α and β in any of these variables to detect possible confounders, but have not found any.

1) SUS: To measure the overall perceived usability, we asked our participants to fill out the SUS questionnaire twice—once for the *Dashboard*, once for SEE—directly after having used each system. The resulting SUS scores for each participant can be found in Table I, and the results compared between the two systems can be seen in Figure 7.



(a) Task completion time for task A_1.

(b) Task completion time for task B_1.

(c) Task completion time for task A_2.

(d) Task completion time for task B_2.

Fig. 6: Task completion time compared between the two systems for each task. Only tasks with significant differences are included. *Note the differing scales for each task.*

Dashboard	55	95	93	65	55	80	85	85	68	75
	98	95	78	63	100	53	95	70	85	75
SEE	43	90	63	58	55	78	85	83	58	58
	88	60	93	65	78	45	35	68	63	88

TABLE I: SUS scores given by each of the twenty participants, rounded to integers.

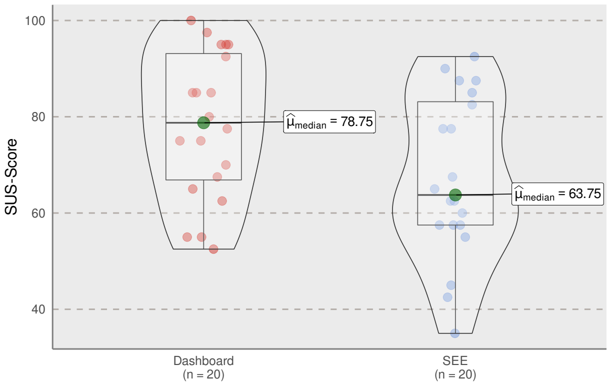


Fig. 7: SUS scores of the *Dashboard* and SEE.

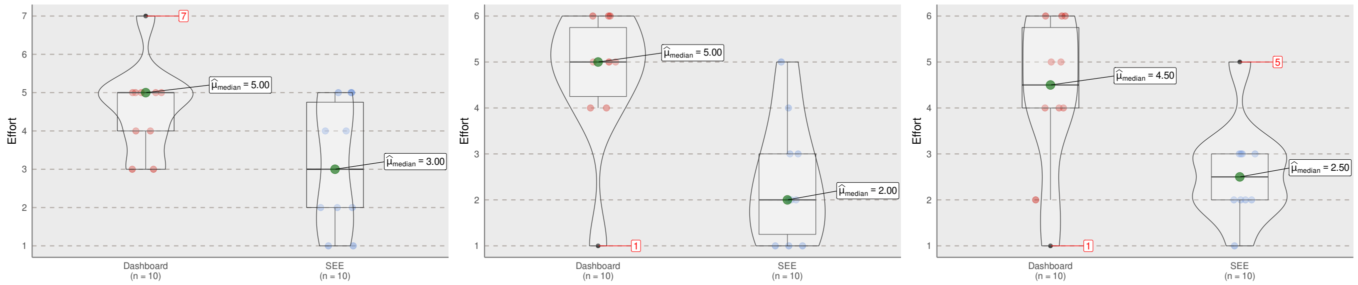
Using the Mann-Whitney-U test again to compare the two systems (though note that this time we have twenty data points for each side), SEE gets a significantly lower score than the *Dashboard* ($p \approx 0.028$). Thus, we **cannot reject** $\mathbf{H_{0,S}}$ and have reasons to believe in the opposite: SEE is perceived as being significantly less usable overall than the *Dashboard* ($S_S < S_D$).

2) ASQ—Perceived Effort: To measure the perceived effort, we have asked the participants after each task on how much effort they feel it took to complete that task.

The violin plots for these answers can be seen in Figure 8. We have found the following three significant differences here:

- **Task $\mathbf{A_1}$** Participants using SEE rated the task as taking less effort than those using the *Dashboard* ($p \approx 0.03$).
- **Task $\mathbf{A_2}$** Participants using SEE rated the task as taking less effort than those using the *Dashboard* ($p \approx 0.003$).
- **Task $\mathbf{B_2}$** Participants using SEE rated the task as taking less effort than those using the *Dashboard* ($p \approx 0.01$).

Hence, we can **reject** $\mathbf{H_{0,A^e}}$ in three tasks and conclude that participants perceived those tasks as taking less effort when using SEE rather than the *Dashboard* ($A^e_S < A^e_D$). Note that this may also relate to the fact that participants generally took less time solving the tasks with SEE ($t_S < t_D$).



(a) Perceived effort for task A_1.

(b) Perceived effort for task A_2.

(c) Perceived effort for task B_2.

Fig. 8: Perceived effort compared between the two systems for each task. Only tasks with significant differences are included.

3) ASQ—Perceived Difficulty: We have found no significant differences in the perceived difficulty of any task, neither when comparing the two groups α and β, nor when comparing the participants using SEE and those using the *Dashboard*. Based on this, we **cannot reject** $\mathbf{H_{0,A^{d}}}$ and—since no significant difference in any direction was found—cannot make a statement either way.

E. The Effect of Experience

Finally, we want to take a look at possible correlations between independent variables measured by our preliminary questionnaire (such as various kinds of experience) and dependent variables listed in Section IV-B (and analyzed in the sections directly preceding this one). Since this means we are comparing ordinal scaled, discrete variables with continuous variables, we use *Kendall's* τ *coefficient*. Because the independent variable *age* is continuous, we additionally provide *Pearson's* r as recommended by the literature [73, p. 158], which leverages the distance between data points and tests for a linear correlation, while Kendall's τ neglects distances and tests for a monotonic correlation. Yet, Pearson's r is used only informative here, not to determine statistical significance for *age*, because we cannot assume a normal distribution, which is a prerequisite for the statistical test associated with r. The p values below always relate to Kendall's τ.

Almost all significant correlations we found were negative, namely:

- The more experience with the *Dashboard* a participant has, the less perceived effort[2] is required to solve a task ($\tau_b \approx -0.4$, $p \approx 0.03$). There are multiple possible reasons for this; for example, experience with the *Dashboard* probably makes using it easier and thus less effortful, since those participants do not need to familiarize themselves with it. Another possibility is that experience with the *Dashboard* correlates with another factor—for example, this is the case for the highest attained degree ($\tau_b = 0.48$, $p \approx 0.01$)—which in turn explains this correlation.
- The more experience in bigger software projects participants have, the less time they take to solve a task ($\tau_b \approx -0.37$, $p \approx 0.04$). A possible conjecture for this is that participants with experience in bigger projects may also be familiar with tools *like* the *Dashboard*.
- The higher the attained degree of a participant is, the lower the SUS score for SEE ($\tau_b \approx -0.52$, $p \approx 0.005$) and the higher the perceived difficulty of a task solved with the *Dashboard* ($\tau_b \approx 0.57$, $p \approx 0.002$) becomes. This comes as a surprise to us, since we would have expected the opposite, namely, that a higher degree leads to being more familiar with "conventional" interfaces, such as the *Dashboard*.
- Finally, the higher the age of a participant is, the lower the SUS score for SEE ($r \approx -0.5$, $\tau_b \approx -0.37$, $p \approx 0.03$) and the higher the perceived difficulty of a task solved with the *Dashboard* ($r \approx 0.54$, $\tau_b \approx 0.36$, $p \approx 0.04$) becomes. These are in essence the same correlations as for the highest attained degree, and can perhaps be explained by the same reasons, due to a correlation between age and highest degree ($\tau_b \approx 0.44$, $p \approx 0.02$).

F. Threats to Validity

This section discusses potential threats to validity of our experiment.

Internal validity: In order to make sure that the effects observed in our dependent variable can solely be attributed to the independent variables, we took the following measures. Differences in prior experiences with either of the tools (SEE and the *Dashboard*) may affect the performance of the participants. To counter this, we created our sample ($n = 20$) of participants from three types of participants: i) *Axivion employees* who are presumed to have prior experience with the *Dashboard* ($n = 6$), ii) SEE *students* who are presumed to have prior experience with SEE ($n = 5$) , and iii) *CS students* who are presumed to have no prior experience with any of the two systems ($n = 9$). In addition, all participants went through a training session in which both systems were introduced by means of explainer videos and a short test to ensure that the explanations of the videos were understood by the participants—i.e., the participants had to answer questions about the system they would use in the following tasks. During the training sessions, the participants could freely use the tool to familiarize themselves with it. The training sessions did not have any time limits and the participants were offered all

[2] As a sidenote, the *actual* effort (measured by the task completion time) almost had the same significant correlation ($\tau_b = -0.35$, $p \approx 0.05$).

	A_1	B_1	C_1	A_2	B_2	C_2
Correctness C	D ($p \approx 0.03$)	D ($p \approx 0.01$)	–	S ($p \approx 0.02$)	–	–
Task completion time t	S ($p \approx 10^{-5}$)	S ($p \approx 0.008$)	–	S ($p \approx 0.007$)	S ($p \approx 10^{-4}$)	–
ASQ (effort) A^e	S ($p \approx 0.03$)	–	–	S ($p \approx 0.003$)	S ($p \approx 0.01$)	–
ASQ (difficulty) A^d	–	–	–	–	–	–

TABLE II: Significant differences between the variables in favor of SEE and the *Dashboard*.

available time to feel comfortable with the tool. That said, we cannot fully exclude that potential "fun factors" in SEE or the *Dashboard* caused the participants to spend more time than necessary to solve the tasks. To counter learning effects during the experiment, the assignment of the participants to the order of the tools used (and therewith the order of the training sessions) was randomized.

External validity: In the following, we discuss whether and how far our results are generalizable. We used convenience sampling to create our sample of participants because in software engineering research it is generally rather difficult to employ other (potentially more reliable) sampling strategies. Nevertheless, convenience sampling is subject to the risk that the collated sample of participants does not reflect the population well. For example, only male developers took part in our experiment. As such, our results may not be generalizable to other genders. Likewise, almost all professional developers of our sample were employees of Axivion. As one of Axivion's core businesses is the analysis of code-smell data, the majority of the participants with a professional background in software development have a greater basic understanding of this topic.

In SEE, nodes are arranged automatically using different layouts. The layouts supported by SEE are: *Tree-maps*, *Circular Balloon*, *Circle Packing*, *Rectangle Packing*, and *EvoStreets*. In our experiment we used the *Circle Packing* layout, which makes it easier to grasp the hierarchical structure of the underlying software project. Other layouts may arrange nodes significantly differently, though, resulting in an entirely different Code City. Accordingly, the layout used in our experiment may affect our results and different results may be observed if other layouts are used.

VI. Conclusions

We conducted a controlled experiment to compare a visualization of code smells as decorations in a Code City to a more traditional tabular tree view. The summary of the experimental results in Table II suggests a tendency towards a higher degree of correctness for the tabular view, yet an even stronger tendency towards a shorter task completion time for the Code City in SEE. In the experiment of Wettel et. al. [29], [30] the authors found that their Code City approach (*CodeCity* [32], [33]) leads to an improvement of both correctness and task completion time compared to the *Eclipse+Excel* baseline. The baseline of our experiment was a professional *Dashboard* application that is specifically designed for the presentation of code-smell data. Accordingly, our baseline might be a stronger competitor to Code Cities. Wettel et. al. also performed a detailed analysis of each task of their experiment and found that in tasks where very precise answers are required (*focused tasks*) *CodeCity* did not perform better than the *Eclipse+Excel* baseline—yet, *CodeCity* managed to be on a par with it in most of the tasks. Our findings agree with the observations of Wettel et. al. with regard to the following statement: Code Cities are better suited to get a quick overview of the code smells of a software. However, according to our results, tabular views *are better suited* to analyze code smells in more detail.

In our experiment, unlike Wettel et al., we also looked into perceived usability based on ASQ and SUS. The objective measure for efficiency (task completion time) mirrors in the perceived effort asked for in ASQ question (2), while the perceived ASQ difficulty asked in ASQ question (1) showed no difference among the two representations. Although the perceived effort was assessed lower for SEE and no difference was found for the perceived difficulty, the SUS score for SEE was statistically significantly lower than the one for the tabular tree view. This is somewhat surprising because the two ASQ questions resemble multiple questions of SUS (many SUS questions are similar to each other). Among the ten SUS questions, we found four to be statistically different between SEE and the *Dashboard*. Two of these (question 4 and 10) are aspects of learnability [74], not covered by the ASQ. As expected, the tabular view could be more readily used by the participants as it is a more commonly known representation. The other two statistically different questions 3 and 8 ask for the ease of use and the confidence while using the system and are again both to the advantage of the *Dashboard*. What exactly lead the participants to answer this way, was not captured in our study. Maybe the more difficult type of navigation in a virtual 3D world and/or the difficulty to precisely compare the dimensions of two blocks visually—in particular, if they are not closely together—may have compromised ease and confidence of use. We also highlight that the SEE students who were more familiar with SEE than the other participants neither gave better SUS scores for SEE than for the *Dashboard* overall: only one favored SEE, one was neutral, and three gave higher SUS scores to the *Dashboard*.

We plan to look into that in further detail by conducting long-term studies where developers will be given sufficient time to familiarize themselves with Code Cities and where we also capture more detailed observations of all the necessary steps to solve a task (e.g., navigating, finding and inspecting a file etc.) to better understand what needs to be optimized and to provide suitable in-game help. The ultimate goal is to improve Code Cities such that detailed information can be retrieved accurately as in tabular views while maintaining their strengths of providing a quick overview.

References

[1] C. Knight and M. Munro, "Virtual but visible software," in *International Conference on Information Visualization*. IEEE, 2000, pp. 198–205.

[2] S. M. Charters, C. Knight, N. Thomas, and M. Munro, "Visualisation for informed decision making; from code to components," in *International Conference on Software Engineering and Knowledge Engineering*, 2002, pp. 765–772.

[3] M. Balzer, A. Noack, O. Deussen, and C. Lewerentz, "Software landscapes: Visualizing the structure of large software systems," in *IEEE TCVG Symposium on Visualization*, 01 2004, pp. 261–266.

[4] T. Panas, R. Berrigan, and J. Grundy, "A 3d metaphor for software production visualization," in *International Conference Information Visualization*. IEEE, 2003, pp. 314–319.

[5] A. Marcus, L. Feng, and J. I. Maletic, "3D representations for software visualization," in *ACM Symposium on Software Visualization*, 2003, pp. 27–36.

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

[7] ——, "Codecity: 3d visualization of large-scale software," in *Companion of the 30th International Conference on Software Engineering*. ACM, 2008, pp. 921–922.

[8] ——, "Visual exploration of large-scale system evolution," in *IEEE Working Conference on Reverse Engineering*. IEEE, Oct. 2008, pp. 219–228.

[9] F. Fittkau, S. Roth, and W. Hasselbring, "ExplorViz: visual runtime behavior analysis of enterprise application landscapes," in *European Conference on Information Systems*, 2015, pp. 1–13.

[10] F. Fittkau, A. Krause, and W. Hasselbring, "Exploring software cities in virtual reality," in *Working Conference on Software Visualization*. IEEE Computer Society Press, 2015, pp. 130–134.

[11] G. o. Balogh, A. Szabolics, and A. Beszédes, "Codemetropolis: Eclipse over the city of source code," in *Conference on Source Code Analysis and Manipulation*, Sep. 2015, pp. 271–276.

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

[13] L. Merino, A. Bergel, and O. Nierstrasz, "Overcoming issues of 3d software visualization through immersive augmented reality," in *Working Conference on Software Visualization*. IEEE Computer Society Press, 2018, pp. 54–64.

[14] W. Scheibel, C. Weyand, and J. Döllner, "Evocells - A treemap layout algorithm for evolving tree data," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2018, pp. 273–280.

[15] L. Merino, M. Hess, A. Bergel, O. Nierstrasz, and D. Weiskopf, "Perfvis: Pervasive visualization in immersive augmented reality for performance awareness," in *ACM/SPEC International Conference on Performance Engineering*, 2019, pp. 13–16.

[16] D. Limberger, W. Scheibel, J. Döllner, and M. Trapp, "Advanced visual metaphors and techniques for software maps," in *International Symposium on Visual Information Communication and Interaction*, Sep. 2019, pp. 1–8.

[17] A. Schreiber, L. Nafeie, A. Baranowski, P. Seipel, and M. Misiak, "Visualization of software architectures in virtual reality and augmented reality," *IEEE Aerospace Conference*, pp. 1–12, 2019.

[18] C. Zirkelbach, A. Krause, and W. Hasselbring, "Hands-on: Experiencing software architecture in virtual reality," Christian-Albrechts-Universität zu Kiel, Research Report 1809, Jan. 2019.

[19] M. Steinbeck, R. Koschke, and M.-O. Rüdel, "Comparing the EvoStreet visualization technique in two- and three-dimensional environments—a controlled experiment," in *International Conference on Program Comprehension*. IEEE Computer Society Press, 2019, pp. 231–242.

[20] M. Steinbeck, R. Koschke, and M.-O. Rüdel, "Movement patterns and trajectories in three-dimensional software visualization," in *Conference on Source Code Analysis and Manipulation*, Sep. 2019, pp. 163–174.

[21] M. Steinbeck, R. Koschke, and M.-O. Rüdel, "How EvoStreets are observed in three-dimensional and virtual reality environments," in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2020, pp. 332–343.

[22] R. Koschke and M. Steinbeck, "Clustering paths with dynamic time warping," in *Working Conference on Software Visualization*, 2020, pp. 89–99.

[23] F. Jung, V. Dashuber, and M. Philippsen, "Towards collaborative and dynamic software visualization in vr," in *Proceedings of the International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 3: IVAPP*, INSTICC. SciTePress, 2020, pp. 149–156.

[24] V. Dashuber, M. Philippsen, and J. Weigend, "A layered software city for dependency visualization," in *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, vol. 3. SciTePress, 2021, pp. 15–26.

[25] R. Koschke and M. Steinbeck, "Modeling, visualizing, and checking software architectures collaboratively in shared virtual worlds," in *Workshop on Software Architecture and Architectural Consistency*, 2021.

[26] ——, "See your clones with your teammates," in *International Workshop on Software Clones*, 2021, pp. 15–21.

[27] F. Steinbrückner and C. Lewerentz, "Representing development history in software cities," in *ACM Symposium on Software Visualization*. ACM, 2010, pp. 193–202.

[28] F. Steinbrückner, "Consistent software cities: supporting comprehension of evolving software systems," Ph.D. dissertation, Brandenburgischen Technischen Universität Cottbus, Cottbus, 06 2013.

[29] R. Wettel, "Software systems as cities," Ph.D. Thesis, University of Lugano, Switzerland, 2010.

[30] R. Wettel, M. Lanza, and R. Robbes, "Software systems as cities: A controlled experiment," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 551–560. [Online]. Available: https://doi.org/10.1145/1985793.1985868

[31] Richard Wettel, "Codecity," 07.06.2022. [Online]. Available: https://wettel.github.io/codecity.html

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

[33] R. Wettel, M. Lanza, and R. Robbes, "Empirical validation of codecity: A controlled experiment," Università della Svizzera italiana, Tech. Rep., 2010.

[34] Axivion GmbH, "Axivion bauhaus suite," 08.05.2018. [Online]. Available: http://www.axivion.com/

[35] J. Brooke, *SUS: A 'Quick and Dirty' Usability Scale*. CRC Press, 1996, pp. 189–194.

[36] J. R. Lewis, "Psychometric evaluation of an after-scenario questionnaire for computer usability studies: the ASQ," *ACM SIGCHI Bulletin*, vol. 23, no. 1, pp. 78–81, 1991.

[37] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2000.

[38] M. Hammad, H. A. Basit, S. Jarzabek, and R. Koschke, "A systematic mapping study of clone visualization," *Computer Science Review*, vol. 37, p. 100266, 2020. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1574013719302679

[39] B. Johnson and B. Shneiderman, "Tree-maps: A space-filling approach to the visualization of hierarchical information structures," in *Proceedings of the Conference on Visualization*. IEEE Computer Society Press, 1991, pp. 284–291.

[40] K. Andrews, J. Wolte, and M. Pichler, "Information pyramids: A new approach to visualising large hierarchies," in *IEEE Conference on Visualization*. IEEE Computer Society Press, 1997, pp. 49–52.

[41] R. Koschke, "Software visualization in software maintenance, reverse engineering, and re-engineering: a research survey," *Journal on Software Maintenance and Evolution*, vol. 15, no. 2, pp. 87–109, 2003.

[42] D. H. R. Holten, "Visualization of graphs and trees for software analysis," Ph.D. dissertation, Technical University of Delft, 2009.

[43] ——, "Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data," *IEEE Transactions on Visualization and Computer Graphics*, vol. 12, no. 5, pp. 741–748, Sep. 2006.

[44] M. Steinbeck and R. Koschke, "Tinyspline: A small, yet powerful library for interpolating, transforming, and querying nurbs, b-splines, and bézier curves," in *IEEE International Conference on Software Analysis, Evolution and Reengineering*. IEEE Computer Society Press, 2021, pp. 572–576.

[45] D. A. Bowman, E. T. Davis, L. F. Hodges, and A. N. Badre, "Maintaining spatial orientation during travel in an immersive virtual environment," *Presence: Teleoper. Virtual Environ.*, vol. 8, no. 6, pp. 618–631, 1999.

[46] B. E. Riecke, D. W. Cunningham, and H. H. Bülthoff, "Spatial updating in virtual reality: the sufficiency of visual information," *Psychological Research*, vol. 71, no. 3, pp. 298–313, May 2007.

[47] J. W. Regian, W. L. Shebilske, and J. M. Monk, “Virtual reality: An instructional medium for visual-spatial tasks,” *Journal of Communication*, vol. 42, no. 4, pp. 136–149, 1992.

[48] S. S. Chance, F. Gaunet, A. C. Beall, and J. M. Loomis, “Locomotion mode affects the updating of objects encountered during travel: The contribution of vestibular and proprioceptive inputs to path integration,” *Presence: Teleoper. Virtual Environ.*, vol. 7, no. 2, pp. 168–178, 1998.

[49] M. Rüdel, J. Ganser, and R. Koschke, “A controlled experiment on spatial orientation in VR-based software cities,” in *Working Conference on Software Visualization*, Sep. 2018, pp. 21–31.

[50] D. Moreno-Lumbreras, R. Minelli, A. Villaverde, J. M. González-Barahona, and M. Lanza, “Codecity: On-screen or in virtual reality?” in *2021 Working Conference on Software Visualization (VISSOFT)*, 2021, pp. 12–22.

[51] M. Steinbeck, “Mining version control systems and issue trackers with libvcs4j,” in *IEEE International Conference on Software Analysis, Evolution and Reengineering*, 2020, pp. 647–651.

[52] G. C. Murphy, D. Notkin, and K. Sullivan, “Software reflexion models: Bridging the gap between source and high-level models,” in *ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM, 1995, pp. 18–28.

[53] A. Assila, K. M. de Oliveira, and H. Ezzedine, “Standardized usability questionnaires: Features and quality focus,” *Electronic Journal of Computer Science and Information Technology*, vol. 6, no. 1, pp. 15–31, Dec. 2016.

[54] S. G. Hart and L. E. Staveland, “Development of NASA-TLX (task load index): Results of empirical and theoretical research,” in *Advances in Psychology*, ser. Human Mental Workload, P. A. Hancock and N. Meshkati, Eds. North-Holland, 1988, vol. 52, pp. 139–183.

[55] F. Zijlstra and L. Doorn, “The construction of a scale to measure perceived effort,” *Department of Philosophy and Social Sciences*, 1985.

[56] J. Sauro and J. S. Dumas, “Comparison of three one-question, post-task usability questionnaires,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1599–1608. [Online]. Available: https://doi.org/10.1145/1518701.1518946

[57] J. Kirakowski, “The use of questionnaire methods for usability assessment,” *Unpublished manuscript. Recuperado el*, vol. 12, 1994.

[58] J. P. Chin, V. A. Diehl, and K. L. Norman, “Development of an instrument measuring user satisfaction of the human-computer interface,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’88. Association for Computing Machinery, 1988, pp. 213–218.

[59] J. Lewis, “Psychometric evaluation of the post-study system usability questionnaire: The PSSUQ,” in *Proceedings of the Human Factors Society*, vol. 2, 1992, pp. 1259–1263.

[60] D. Tedesco and T. Tullis, “A comparison of methods for eliciting post-task subjective ratings in usability testing,” *Usability Professionals Association (UPA)*, vol. 2006, pp. 1–9, 01 2006.

[61] W. Funk and B. Roegele, *Safety4Bikes. Assistenzsystem für mehr Sicherheit von fahrradfahrenden Kindern. Arbeitspaket 1: Usability-Evaluation des Gesamtsystems aus der Anforderungsperspektive von Kindern*. Nürnberg: Institut für empirische Soziologie, 2020. [Online]. Available: https://www.ifes.fau.de/files/2020/09/ifes-mat-3-2020-safety4bikes-arbeitspaket-1-usability-evaluation-1.pdf

[62] J. Sauro and J. R. Lewis, “Correlations among prototypical usability metrics: evidence for the construct of usability,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2009, pp. 1609–1618.

[63] S. C. Peres, T. Pham, and R. Phillips, “Validation of the system usability scale (SUS): SUS in the wild,” *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 57, no. 1, pp. 192–196, 2013.

[64] A. Bangor, P. T. Kortum, and J. T. Miller, “An empirical evaluation of the system usability scale,” *International Journal of Human–Computer Interaction*, vol. 24, no. 6, pp. 574–594, 2008.

[65] J. R. Lewis, “The system usability scale: Past, present, and future,” *International Journal of Human–Computer Interaction*, vol. 34, no. 7, pp. 577–590, 2018.

[66] B. Rummel and E. Ruegenhagen. System usability scale – jetzt auch auf deutsch. [Online]. Available: http://web.archive.org/web/20200607035254/https://experience.sap.com/skillup/system-usability-scale-jetzt-auch-auf-deutsch/

[67] J. Sauro, *A practical guide to the Systems Usability Scale: Background, Benchmarks & Best Practices*. CreateSpace Independent Publishing Platform, 2011.

[68] S. Mclellan, A. Muddimer, and S. Peres, “The effect of experience on system usability scale ratings,” *Journal of Usability Studies*, vol. 7, 2011.

[69] I. Patil, “Visualizations with statistical details: The ’ggstatsplot’ approach,” *Journal of Open Source Software*, vol. 6, no. 61, p. 3167, 2021.

[70] J. W. Tukey, *Exploratory data analysis*. Reading, Mass. : Addison-Wesley Pub. Co., 1977. [Online]. Available: http://archive.org/details/exploratorydataa00tuke_0

[71] M. McGee, “Case for omitting tied observations in the two-sample t-test and the wilcoxon-mann-whitney test,” *PLOS ONE*, vol. 13, no. 7, p. e0200837, 2018, publisher: Public Library of Science.

[72] J. Bortz, G. A. Lienert, T. Barskova, K. Leitner, and R. Oesterreich, “Testmethoden für kardinaldaten,” in *Kurzgefasste Statistik für die klinische Forschung: Leitfaden für die verteilungsfreie Analyse kleiner Stichproben*, ser. Springer-Lehrbuch. Springer, 2008, pp. 227–255.

[73] H. Khamis, “Measures of association: How to choose?” *Journal of Diagnostic Medical Sonography*, vol. 24, no. 3, pp. 155–162, 2008, publisher: SAGE Publications Inc STM.

[74] J. R. Lewis and J. Sauro, “The factor structure of the system usability scale,” in *Human Centered Design*, ser. Lecture Notes in Computer Science, M. Kurosu, Ed. Springer, pp. 94–103.