SEE Your Clones With Your Teammates

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

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

2021 IEEE 15th International Workshop on Software Clones (IWSC) | 978-1-6654-4287-9/21/$31.00 ©2021 IEEE | DOI: 10.1109/IWSC53727.2021.00009

***Abstract*—We discuss how collaborative clone analysis in distributed teams can be supported by modern software-visualization technology. We present our multi-user multi-purpose software visualization platform SEE that is based on the popular code-as-a-city metaphor. We discuss how clone data can be visualized in SEE without compromising the other use cases of SEE. Our design decisions are based on principles of cognitive psychology, namely, the laws of Gestalt.**

***Index Terms*—clone analysis, software visualization, virtual and augmented reality, code cities, distributed development**

I. Introduction

Code-Cities are a popular type of software visualization that can be used in many different tasks in software development [1]–[20]. They can also be used to support clone analysis [21]. Clone management often requires the expertise of many stakeholders: project managers, architects, developers and others. That is, the results of clone detectors must be assessed not only because they may contain false positives, but also to make a decision on how to handle the found clones. There may be code fragments that are deliberately duplicated for technical (e.g., loop unrolling for the sake of run-time optimization) or organizational reasons [22]. Because many aspects play a role, the decision to remove or tolerate clones is often a team effort. Such joint team efforts are impeded if the team is not in the same location. Indeed there has been a sustained trend towards distributed software development long before the current pandemic situation [23]. No matter whether other team members are working offshore or just in a different office in the same building, if they are not physically present in the same room, spatial gaps must be bridged. The existing means to bridge these gaps in practice are chats, video conferencing, and screen sharing, which have no particular support for software visualization and are often cumbersome to use. Moreover, the team generally needs to view the clones in the overall context of a system. For instance, the refactoring of clones in an old subsystem not changed for a long time and soon to be replaced is hardly wanted. Modern software visualization can assist in both aspects by providing an overall context and enable remote collaboration.

Contributions: In this paper, we present our multi-user multi-purpose software-visualization platform *SEE* (for Software Engineering Experience) and show how it can be used for collaborative investigation of software clones. SEE allows multiple developers to work together on comprehending their software in shared virtual worlds at the same time even if they are not physically in the same place. The visualization is based on the well known *software-as-a-city* metaphor and can be used to depict a large set of metrics—including but not limited to cloning, to view the implementation in the context of the architecture [24], to trace its evolution, and to analyze run-time behavior. That said, not only the Code-Cities can be viewed, but all participants can also see and share the source code of the visualized code entities. The users, represented as avatars (cf. Figure 1), can see each other, discuss the relevance of the clones via voice chat, and interact naturally with the Code-City and its associated source code, where SEE makes sure that they all have a consistent view no matter where they are located and what kind of device they use (desktop, tablet, VR hardware). SEE is therefore well suited to address the aspects *collaboration* and *overall context* in clone management.

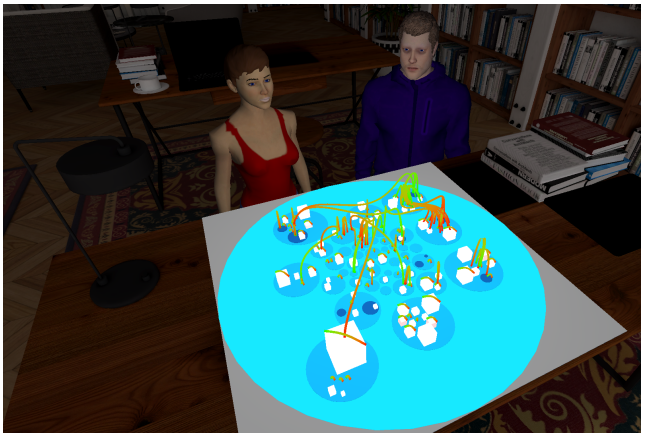

Fig. 1. Virtual room of a Code-City for subsystem Linux/Net with clone data

We discuss our design decisions on how to visualize code clones using Code-Cities. Rather than most other existing research on visualizing clone data (see Hamad et. al. [25] for a recent comprehensive survey), our visualization is not aimed at showing only clone data. Instead we discuss how Code-Cities that are used for a general overview of software already can be adapted to also show clone data, that is, how the aspect of cloning can be integrated into these without compromising their other purposes. These decisions are based on the laws of Gestalt [26], which are a set of principles in cognitive psychology accounting for how humans naturally perceive objects as organized patterns.

Outline: The remainder of this paper is structured as follows. Section II presents related research. Section III delves

978-1-6654-4287-9/21/$31.00 ©2021 IEEE
DOI 10.1109/IWSC53727.2021.00009

into our design decisions on how to visualize clone data in SEE. Section IV concludes.

II. Related Research

Our research focuses on the visualization of software and clone data. Based on the *software-as-a-city* metaphor, we develop a visualization platform with a special emphasis on collaboration in shared virtual worlds, remotely accessible from different hardware devices. There are therefore different topics of related research presented in the following. For a comprehensive recent overview on clone visualization, we refer to Hamad et. al. [25].

A. Code-Cities

Code cities are a visual language to express quantitative—in other words, metrics—and hierarchical properties of software at the same time. They can be viewed as a three-dimensional extension of *Tree-maps* [27], an early attempt to visualize both kinds of properties at once. The basic idea of *Tree-maps* is to recursively subdivide a rectangular shape into smaller nested rectangles. That is, the hierarchy is expressed by spatial inclusion. The area of the innermost rectangles is proportional to a selected quantitative attribute—this way expressing a metric as a proportion of the available space. This approach turned out to be quite advantageous (as it allows to identify atypical patterns in a spatial context) and quickly the idea came up to visualize an additional metric by scaling the height of the rectangles. Due to the impression of North American downtowns with building blocks arranged in grids, such three-dimensional Tree-maps were called *Code-Cities* [28]. Researchers quickly adopted the idea of Code-Cities and used them—and still do so—to visualize a variety of aspects of software [1]–[20]. By mapping colors and textures onto the surface of the three-dimensional blocks of a Code-City, yet another metric can be visualized in an intuitive way [16]. Code-Cities today vary also in their layout. There are not only different variants of Tree-map layouts but also very different hierarchical layouts used, such as rectangle or circle packing or the *EvoStreets* layout [29]. In addition to the mapping of metrics for individual elements, a visualization of relations between elements (e.g., cloned fragments in source-code-files) is often also required. To visualize relations, edges are frequently used [30]. In order to diminish the visual clutter that can occur when visualizing a lot of edges (especially when edges are overlapping), hierarchical edge bundles, as proposed by Holten [31], [32], have proven suitable. Code-Cities are not just an object of research but have also arrived in practice. Examples of use in practice include the tools by *Hello2morrow* and *Serene* [33] as well as *SonarQube*'s plug-in *SoftVis3D*.

B. Code-Cities in 3D Environments

With advancing technical progress in the area of virtual reality (VR), the attempt to represent Code-Cities in VR emerged as the next step towards a visualization that engages a larger spectrum of human perception. Considerations in this direction were made as early as 2000 [1], [2]. Studies outside of computer science observed that VR might be advantageous [34]–[37] and so researchers in the area of software visualization hoped that these observations also apply in visual software analytics. There are a multitude of studies with the aim of visualizing Code-Cities in pseudo 3D (desktop computer monitors) and VR environments: [4], [9], [10], [15], [17], [38]–[46]—to list only a few. Some studies have shown that head-mounted displays (HMDs) may have a positive effect on the orientation [34] and navigation speed [47] of users. That said, there are also studies where a positive effect could not be found [48] or where using HMDs had only little effect [49]. These conflicting results suggest that more research is needed.

C. Collaborative Software Visualization

Co-located collaborative software visualization where people interact with each other physically *in the same place* was studied by Isenberg et al. [50], [51] and Anslow et al. [52], [53]. The authors developed a multi-touch display mounted onto a table that can be used by several people (primarily pairs of users) at the same time. One of the key findings of these studies was that working face-to-face around the table (i.e., sharing individual findings and communicating throughout) was a successful way for the pairs to solve complex problems, and that collaborative software visualization should support multiple users. An early attempt of implementing a distributed collaborative visualization (i.e., users do not have to be physically in the same place) in the context of software comprehension can be found in the work of Kot et al. [54]. Based on the *Quake 3* game engine, the authors developed a shared three-dimensional world where users can view, move, and arrange source-code files interactively. Further works in the area of collaborative software visualization are [38], [55]–[58]—in a broader sense also [59]. Collaborative software visualization with focus on Code-Cities in shared virtual words was recently studied by Zirkelbach et al. [60] and Jung et al. [19]. In both studies users were represented as abstract avatars in the virtual world, allowing them to perceive each other and to support their verbal communication with a *simple form* of gesture. This aspect was in particular received positively in the study by Zirkelbach et al. [60], yet, the participants would have preferred a more realistic, human representation.

III. Design Decisions

In this section, we will explain and justify our design decisions on how to adapt SEE to provide clone information. The decisions will derive from a set of principles in cognitive psychology accounting for how humans naturally perceive objects as organized patterns, known as the laws of Gestalt [26]. We are using SEE for multiple purposes beyond just showing clone data. Code-Cities are used to support architecture conformance checking, debugging, performance analysis, as well as quality assessment in general. To support all these use cases, it is important to maintain a human beholder's mental map across uses cases and to provide uniform means to visualize data. In other words, we cannot arbitrarily change a Code-City for another use case, but need to integrate the

additional clone information into our existing visualization intelligently. This goal imposes constraints that we need to take into account in our design decisions.

We will describe those constraints first, then briefly introduce the concept of Gestalt principles, and after that discuss each principle in greater depth for Code-Cities in general and our Code-Cities in SEE in particular.

A. Code-Cities in General

The visual language of Code-Cities in general has already been introduced in Section II-A. The hierarchy of a program depicted by them is often formed by syntactic nesting (e.g., namespaces, packages, classes, etc.), physical structures (i.e., files nested in directories), or class hierarchies. The leaves of these hierarchies are generally syntactic (e.g., methods) or physical (i.e., files) entities. They are most often represented as blocks, such that they resemble real-world buildings in cities, although they may also take on other shapes such as cylinders, pyramids, etc. The kind of shape chosen may represent the type of the entity visualized. Because nesting is expressed by spatial inclusion, the representation of inner nodes must be visual objects that can contain other visual objects. The exact type of shape for inner nodes depends partly on the automated layout. In the original Tree-maps, rectangles were used. Circle packing and balloon layouts suggest to use circles. The EvoStreets use streets to depict inner nodes. All of these are essentially two-dimensional because their contained objects should be seen, but there are also Code-City variants where inner nodes have a three-dimensional shape such as a transparent half sphere [61]. Relations among these entities are visualized as edges—often hierarchically bundled.

B. Code-Cities in SEE

SEE (*Software Engineering Experience*) is a multi-user multi-purpose software visualization platform built upon the *Unity* game engine. The underlying data model visualized by SEE is a hierarchical graph with attributed nodes and edges—*hierarchical* means that nodes can be nested. SEE does not make any assumptions about what nodes, edges, and attributes represent and thus could be used to visualize anything that can be encoded as hierarchical graph.

Graphs can be imported from *GXL* files, a standard file format for exchanging arbitrary graphs that is used both in academia and industry [62]. The nodes and edges of an imported graph are visualized as three-dimensional blocks (leaf nodes), two-dimensional surfaces enclosing blocks (inner nodes), and splines connecting blocks or surfaces (edges) that can be hierarchically bundled. The visual facets of the blocks (width, height, depth, and color), surfaces (shape and color), and splines (thickness, color gradient, and bundling strategy) can be freely configured based on the node and edge attributes. Using one of the built-in layout engines, blocks (and thus implicitly also surfaces and splines) can be arranged automatically. At the moment of writing, SEE supports *Circular Balloon*, *Circle Packing*, *Rectangle Packing*, *Tree-map*, and *EvoStreets* layouts.

Our envisioned users are software architects, project managers, and developers who need to assess the software architecture or internal code quality, find performance bottlenecks or debug a program as a team. They can enter a virtual room—in which they can interact with Code-Cities—remotely using different types of devices, namely, regular desktop computers with 2D monitor, keyboard, and mouse, tablets with pens, as well as VR hardware connected over the Internet. We are currently also working on supporting AR devices. The users can see each others as avatars and communicate via voice chat.

The different use cases, in which those team members meet, are all supported by the same uniform means of Code-Cities. It is important that a Code-City does not change radically from use case to use case so as to maintain the beholders' mental maps. That means if the users have decided to let a particular metric determine the height of a block, for instance, the same metric should determine this visual attribute across all use cases. This need for consistency limits the degree of freedom we have to integrate additional clone data. Moreover, the input graphs in SEE typically model higher-level implementation (e.g., methods, classes, packages) or architectural concepts (e.g., layers and components) and their relations. This underlying logical data model and its mapping onto visual objects should neither change from use case to use case. A Code-City must capture the same kind of graph in a consistent way across all use cases. That is, clone information must be superimposed on an existing visualization of this graph.

C. Principles of Gestalt

In 1923, Wertheimer postulated six principles on how humans group together single visual or acoustic items [26]. These principles hold independently of cultural and even interpersonal differences [63]. This set was later extended by other researchers. While Wertheimer originally used the terms *principles* and *factors* to describe them, they are now often referred to as the *laws of Gestalt* (*Gestalt* is the German word for shape, figure, form, or outline but also exposure and effect), although the term *laws* is rather overstretching their significance. Among other aspects of cognitive psychology, these principles should be taken into account when designing human-computer interfaces. For instance, MacNamara has explored the effect of the Gestalt principles in VR [63], but only for 2D GUI panels, not for true 3D objects. It is worthwhile to consider and evaluate them also in metaphoric visualizations such as Code-Cities. Code-Cities aim at supporting visual analytics which leverages human visual perception and pattern detection to facilitate reasoning. Consequently, the principles of Gestalt need to be kept in mind when devising the details of visualizaton to foster this visual reasoning and to avoid false conclusions due to misconception.

In the following, we describe the original principles by Wertheimer and additional ones observed by other researchers relevant to our context and relate their relevance specifically to Code-Cities. Each principle is discussed in more detail by (1) a brief description, (2) discussion of its relevance for

Code-Cities in general, and then (3) justification of our design decisions on how to integrate clone data in SEE.

D. Principle of Similarity

Clones are similar code entities, thus, it is the obvious thing to visualize them as similar objects. In fact, the *principle of similarity* suggests just that. It states that similar things tend to appear grouped together. Leaf entities in the code hierarchy are typically represented by a solid mesh in Code-Cities. Those meshes could be formed according to the code of these entities. If two code entities have similar code, they would have similar meshes. Alternatively, other visual attributes of the representation of leaves in the code hierarchy such as size, color, or texture could be used to let them look alike. The similarity of inner nodes is more difficult to achieve if they are just two-dimensional containers such as rectangles or circles. They will at least have a line with a color and a texture that may highlight their similarity. Yet, since the reason for their similarity is normally the similarity of their contained elements, the similarity in representation of these will also implicitly suggest the similarity of the inner nodes themselves. This can be further emphasized if the positions of the inner elements are aligned alike such that similar inner elements are at similar relative positions within their container.

In SEE, the user can select the type of shape for inner nodes and leaves of the node hierarchy so the mesh of objects is already fixed. Yet, because leaf nodes are blocks in SEE we can use their sides to show their similarity. Drawing on the idea of SeeSoft [64], we suggest to put the shape of the code (possibly pretty printed to abstract from differences in code layout) implementing an entity depicted as block on each side. After all, it is the similarity of the code that makes them clones (unless they are type-4 clones). In other words, we downscale the code such that it fits into the available area of each side. Only the sides of a block are "imprinted", but the block remains a block. Obviously, this idea has its limitations for small objects with a lot of code such that the shape of the code on the block's size cannot be deciphered. Even though SEE supports zooming to increase focused objects, zooming can solve this problem at most partially because the "bigger picture" is lost in zooming.

Likewise to the shape of objects, the user selects which metrics determine the height, width, and depth of the objects for leaf nodes. If they were changed just to show other information on clones, the dimensions of the nodes may change radically, which may effect spatial relations because objects becoming larger must receive more space as overlap must be avoided. Significantly sized objects (e.g., tall blocks or large flat areas) serve as a landmark and if their sizes change so that they can no longer be recognized, a beholder may become disoriented. If, however, similar code entities (i.e., clones) have similar values for the metrics chosen to determine the dimensions, they will also look similar. For these reasons, the metrics determining the dimensions of the blocks should not be changed just to show clone information.

Changes in color are less invasive and that is why many researchers using Code-Cities adjust colors to convey more flexible information. For the same reasons, colors are often chosen differently in cartography to highlight different aspects while the size and positions of objects on a map are maintained. So it would be acceptable to let color express clone-specific information such as the clone rate if a Code-City in SEE is to be used to assess cloning of a program.

We refrain from changing the positions of the objects just to align nested similar elements to make the similarity of their container elements more obvious as this would make a previous mental imprint of the spatial relations useless. As we will discuss in Section III-E, we can use edges to align similar leaves instead, which also helps to identify similar inner nodes.

E. Principle of Uniform Connectedness

The *principle of uniform connectedness* states that elements that are connected to each other by uniform visual properties such as colors, lines, frames, or shapes are perceived as a single unit when compared with other elements not connected in the same manner. Frames are the typical means to visualize inner nodes, that is, containers of other elements. It is also common to visualize relations as lines (i.e., edges) conntecting the related elements [30]. The problem with lines is that they may cross other lines or objects and if there are too many of them, they create visual clutter. Instead of explicitly connecting the related elements with lines, one can also use colors. For instance, Yoshimura et al. color similar files (represented as dots) together forming a clone class by the same color rather than connecting them by lines [65]. However, there is only a limited number of colors humans are able to distinguish and the same entity may even be contained in different clone classes so that it would need to have multiple colors.

In SEE, we use edges—drawn as splines—to model code entities sharing similar code, where edges can be bundled to remove some of the visual clutter. The thickness of the splines can depict a clone metric such as the size of shared code as proposed originally by Rieger et al. [66]. The use of edges to model and visualize relations is consistent with the use of splines to show other code dependencies in SEE and the spline color is reserved for the edge type. Edges can be hidden by users on demand (e.g., by selection or by edge type) to get a better view.

F. Principle of Common Fate

The *principle of common fate* states that humans tend to perceive items moving in the same direction as being more related than items that move in different directions or are stationary. For instance, SEE animates the version history. All entities deleted from one revision to the next one, sink to the ground and fade out, whereas all new entities fade in and raise up. Yet, this principle has relevance not only for animations but also for hierarchical edge bundling in Code-Cities. Here, all splines for edges staying in the same subtree of the node hierarchy are routed through a unique control point related to this subtree (technically, the nearest common ancestor of the

source and target of an edge in the node hierarchy); this way, they take on the same fate of a shared static route.

G. Principle of Proximity

According to the *principle of proximity*, objects that are closer together appear to be more related than objects that are spaced farther apart. Where objects are placed in Code-Cities is generally decided by an automated layout algorithm, whose optimization criteria differ based on the nature of the layout (e.g., Tree-maps optimize the use of space) but generally do not include semantic relatedness. Yet, this principle suggests to put *related* entities closer together and to spread unrelated entities. If edges are used to describe relations, force-directed layouts can be used to group related objects together visually. For instance, Yoshimura et al. use a force-directed layout algorithm to cluster similar files [65]. We note, however, that force-directed layouts can take only those relations into account that are explicitly represented as edges and in general, there are many different kinds of relations beyond just clone relations. Moreover, Code-Cities express the hierarchy as spatial inclusion which puts an additional constraint on the automatic layouts not to move the objects arbitrarily: they must remain in their container.

SEE offers multiple hierarchical layouts including a force-directed layout, which can take into account clone relations—yet, these are generally not the only type of edge in the underlying dependency graph as SEE supports multiple use cases. Moreover, users are allowed to move objects around. This way even semantic relations not modelled explicitly as edges can be accommodated. Once the positions are determined, they should not change just because the Code-City is used in a different use case, in particular, if the users have arranged the objects. For these reasons, proximity is not used in SEE to show clone information specifically.

H. Principle of Common Region

The *principle of common region* states that objects located in the same closed region are perceived as belonging together. It is similar to the principle of proximity, but here an explicit rendering of a region, e.g., a line enclosing objects, is suggesting grouping; not all objects in such a region are necessarily very close to each other. This principle is leveraged by most Code-Cities; they generally have an explicit rendering of regions by enclosing lines or colored planes representing inner nodes of the node hierarchy. EvoStreets have a more distinct approach to show inner nodes. Here the contained objects are not visually nested in their container, but the containers are drawn as bars in analogy to streets at which the leaves are lined up as buildings and nested inner nodes are drawn as orthogonal side streets. At the center is the street representing the root of the node hierarchy. The width of the streets decreases along with their level in the node hierarchy. Even though the original inventors of the EvoStreets layout [29] have not done that, all nodes with the same parent in the node hierarchy could be easily enclosed by a rectangular line because the branches of the node hierarchy projected onto the plane do not overlap (unless they are nested in each other, in which case they are fully contained) [18], [67], [68].

SEE includes different layouts with explicit regions, but also EvoStreets in their original form. As noted in Section III-B, the node hierarchy in SEE is given by the underlying hierarchical dependency graph, so common explicit regions are used solely to express the node hierarchy, not any clone information. For the same reasons already discussed in Section III-D, we refrain from changing existing regions for the purpose of just highlighting clones. One could still render regions superimposed on the existing layout chosen as suggested by Byelas and Telea in form of their so called *areas of interest* [69]. This idea could be used for leveraging the principle of common region, for instance, for showing all objects together forming a clone class—at least if there are not too many such clone classes.

I. Principle of Prägnanz

The *principle of prägnanz* (also sometimes referred to as the principle of good figure or the principle of simplicity; the German word *prägnanz* means brief, precise, and accurate) asserts that a set of ambiguous or complex objects is interpreted in the simplest way. For instance, when we look at the Olympic logo with its six rings, we see overlapping circles rather than an assortment of curved, connected lines. To avoid misinterpretations because a human beholder falsely follows this principle, most Code-Cities, including those in SEE, use simple shapes such cuboids or cylinders to visualize leaf nodes and rectangles or circles for inner nodes—which can be considered *prägnant*—and they are placed with sufficient space in between to avoid any overlap. SEE uses the game engine *Unity*, which offers—as many other game engines—a rendering mechanism called *level of detail (LOD)*, which allows to draw different variants of the same object depending upon how much space the objects occupies in the visible area. That is, when an object is small because it is soon from afar, a simplified (i.e., a more abstract) shape can be used. If it is viewed closely and occupies a larger area, additional details can be rendered. SEE uses this mechanism to show additional decorations, which could also be used to fade in clone data, e.g., the miniaturized code put on the sides of a cuboid as discussed in Section III-D.

J. Principle of Continuity

The *principle of continuity* predicts points that are connected by lines are seen in a way that follows the smoothest path, that is, are continued in their established direction. This principle is particularly relevant for drawing edges. Suppose a cross of lines in angles of 90 degrees can be seen. This shape can be explained by two straight edges crossing each other on their half way or as two bending edges in an orthogonal edge layout whose way points happen to be at the same point in space. The principle of continuity would predict that this situation will be interpreted in the former way because the lines continue in their established direction.

Although edge crossing can be fully avoided in the 3D space used for Code-Cities, there may still be perspectives in which this situation could occur. Only if the beholder moved around, she or he would be able to disambiguate the scenery. The risk of misunderstandings of this kind is somewhat mitigated in SEE by the hierarchical bundling, which leads edges through different layers of height. Moreover, SEE colors the edges based on a color gradient from their source to their target. Colors have a base color that represents the type of the edge, but they are actually rendered from a lighter version of this base color to a darker version in the direction of the edge. When two edges meet, the additional color information may help to interpret the situation correctly. This may work even if the edges have the same type because depending on the proportion where they meet, they may have a different range of the color gradient at the area where they cross. Yet, if two edges of they same type meet both half ways, the situation is still ambiguous. However, because we use splines, which do not have sudden changes of their direction, the principle of continuity generally yields the accurate interpretation. If in doubt, the user can select an edge to highlight it so that it can be clearly distinguished from other edges.

IV. Conclusions

In this paper, we have described our software visualization platform SEE and how it can be used to support remote collaborative clone analysis in distributed teams. We explained our most important design decisions for the visualization with Code-Cities based on the laws of Gestalt. Our discussion emphasizes the relevance of these universal principles for this approach of software visualization. We elaborated how those principles are already leveraged by Code-Cities both in SEE and in general and what other opportunities exist on how Code-Cities could be even better perceived by human beholders. Ultimately, all design decision are trade-offs between multiple partly conflicting requirements. In our case, most of them are due to our attempt to use Code-Cities consistently for multiple use cases. That is why we cannot follow all principles of Gestalt, yet we discussed how clone data can nevertheless be visualized effectively by falling back on alternatives principles of Gestalt. As a next step, we plan to evaluate our design decisions empirically.

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 on Information Visualization*. IEEE, 2003, pp. 314–319.

[5] A. Marcus, L. Feng, and J. I. Maletic, "3D representations for software visualization," in *ACM International 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*, June 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*, 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 *IEEE Working Conference on Software Visualization*, 2015, pp. 130–134.

[11] G. o. Balogh, A. Szabolics, and A. Beszédes, "Codemetropolis: Eclipse over the city of source code," in *IEEE International Working 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 *IEEE 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 *IEEE Working Conference on Software Visualization*, 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] 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.

[19] 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.

[20] 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.

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

[22] J. R. Cordy, "Comprehending reality-practical barriers to industrial adoption of software maintenance automation," in *International Conference on Program Comprehension*. IEEE, 2003, pp. 196–205.

[23] C. Ebert, M. Kuhrmann, and R. Prikladnicki, "Global software engineering: evolution and trends," in *International Conference on Global Software Engineering*, 2016, pp. 144–153.

[24] 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, submitted for publication.

[25] 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

[26] M. Wertheimer, *Festschrift für Carl Stumpf*, ser. Psychologische Forschung; Zeitschrift für Psychologie und ihre Grenzwissenschaften. Verlag von Julius Springer, 1923, vol. 4, ch. Untersuchungen zur Lehre von der Gestalt, pp. 301–350.

[27] B. Johnson and B. Shneiderman, "Tree-maps: A space-filling approach to the visualization of hierarchical information structures," in *Proceed-*

ings of the Conference on Visualization. IEEE Computer Society Press, 1991, pp. 284–291.

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

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

[30] 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.

[31] D. H. R. Holten, "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.

[32] ——, "Visualization of graphs and trees for software analysis," Ph.D. dissertation, Technical University of Delft, 2009.

[33] J. Bohnet, "Visualization of execution traces and its application to software maintenance," Dissertation, Hasso-Plattner-Institut, Universität Potsdam, Oct. 2010.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

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

[39] J. Waller, C. Wulf, F. Fittkau, P. Döhring, and W. Hasselbring, "Synchrovis: 3d visualization of monitoring traces in the city metaphor for analyzing concurrency," in *IEEE Working Conference on Software Visualization*, 2013, pp. 1–4.

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

[41] K. Ogami, R. G. Kula, H. Hata, T. Ishio, and K. Matsumoto, "Using high-rising cities to visualize performance in real-time," in *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*. IEEE, 2017, pp. 33–42.

[42] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, "Communicating software architecture using a unified single-view visualization," in *IEEE International Conference on Engineering Complex Computer Systems*. IEEE, 2007, pp. 217–228.

[43] P. Khaloo, M. Maghoumi, E. Taranta, D. Bettner, and J. Laviola, "Code park: A new 3d code visualization tool," in *IEEE Working Conference on Software Visualization*. IEEE, 2017, pp. 43–53.

[44] L. Merino, J. Fuchs, M. Blumenschein, C. Anslow, M. Ghafari, O. Nierstrasz, M. Behrisch, and D. A. Keim, "On the impact of the medium in the effectiveness of 3d software visualizations," in *IEEE Working Conference on Software Visualization*. IEEE, 2017, pp. 11–21.

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

[46] F. Fernandes, C. S. Rodrigues, and C. Werner, "Dynamic analysis of software systems through virtual reality," in *Symposium on Virtual and Augmented Reality*, Nov. 2017, pp. 331–340, in Spanish.

[47] R. A. Ruddle, S. J. Payne, and D. M. Jones, "Navigating large-scale virtual environments: what differences occur between helmet-mounted and desk-top displays?" *Presence: Teleoperators & Virtual Environments*, vol. 8, no. 2, pp. 157–168, 1999.

[48] B. Sousa Santos, P. Dias, A. Pimentel, J.-W. Baggerman, C. Ferreira, S. Silva, and J. Madeira, "Head-mounted display versus desktop for 3d navigation in virtual reality: A user study," *Multimedia Tools and Applications*, vol. 41, no. 1, pp. 161–181, Jan. 2009.

[49] R. A. Ruddle and P. Péruch, "Effects of proprioceptive feedback and environmental characteristics on spatial learning in virtual environments," *International Journal of Human-Computer Studies*, vol. 60, no. 3, pp. 299–326, 2004.

[50] P. Isenberg, D. Fisher, M. R. Morris, K. Inkpen Quinn, and M. Czerwinski, "An exploratory study of co-located collaborative visual analytics around a tabletop display," *IEEE Symposium on Visual Analytics Science and Technology*, pp. 179–186, 2010.

[51] P. Isenberg, D. Fisher, S. A. Paul, M. R. Morris, K. Inkpen, and M. Czerwinski, "Co-located collaborative visual analytics around a tabletop display," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, no. 5, pp. 689–702, 2012.

[52] C. Anslow, S. Marshall, J. Noble, and R. Biddle, "Sourcevis: Collaborative software visualization for co-located environments," in *IEEE Working Conference on Software Visualization*, Sep. 2013, pp. 1–10.

[53] C. Anslow, "Reflections on collaborative software visualization in co-located environments," in *IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 645–650.

[54] B. Kot, B. Wuensche, J. Grundy, and J. Hosking, "Information visualisation utilising 3d computer game engines case study: A source code comprehension tool," in *ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: Making CHI Natural*, 2005, pp. 53–60.

[55] M. D'Ambros and M. Lanza, "A flexible framework to support collaborative software evolution analysis," in *European Conference on Software Maintenance and Reengineering*, 2008, pp. 3–12.

[56] M. D'Ambros and M. Lanza, "Distributed and collaborative software evolution analysis with Churrasco," *Science of Computer Programming*, vol. 75, no. 4, pp. 276–287, 2010.

[57] T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, and R. Vuduc, "Communicating software architecture using a unified single-view visualization," in *IEEE International Conference on Engineering Complex Computer Systems*, 2007, pp. 217–228.

[58] M. Ferenc, I. Polasek, and J. Vincur, "Collaborative modeling and visualization of software systems using multidimensional UML," in *IEEE Working Conference on Software Visualization*, 2017, pp. 99–103.

[59] E. Stroulia, I. Matichuk, F. Rocha, and K. Bauer, "Interactive exploration of collaborative software-development data," in *IEEE International Conference on Software Maintenance*, 2013, pp. 504–507.

[60] 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.

[61] R. Oberhauser, "VR-UML: The unified modeling language in virtual reality – an immersive modeling experience," in *Business Modeling and Software Design*, B. Shishkov, Ed. Cham: Springer International Publishing, 2021, pp. 40–58.

[62] R. Holt, A. Winter, and A. Schürr, "GXL: toward a standard exchange format," in *IEEE Working Conference on Reverse Engineering*, 2000, pp. 162–171.

[63] W. MacNamara, "Evaluating the effectiveness of the gestalt principles of perceptual observation for virtual reality user interface design," Master's thesis, Technological University Dublin, 2017.

[64] S. G. Eick, J. L. Steffen, and E. E. Sumner Jr, "Seesoft—a tool for visualizing line oriented software statistics," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 957–968, 1992.

[65] K. Yoshimura and R. Mibe, "Visualizing code clone outbreak: An industrial case study," in *International Workshop on Software Clones*. IEEE, 2012, pp. 96–97.

[66] M. Rieger, S. Ducasse, and M. Lanza, "Insights into system-wide code duplication," in *IEEE Working Conference on Reverse Engineering*, 2004, pp. 100–109.

[67] 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*, 2019, pp. 231–242.

[68] ——, "Movement patterns and trajectories in three-dimensional software visualization," in *IEEE International Working Conference on Source Code Analysis and Manipulation*, Sep. 2019, pp. 163–174.

[69] H. Byelas and A. Telea, "Visualization of areas of interest in software architecture diagrams," in *ACM Symposium on Software Visualization*, 2006, pp. 105–114.