J Vis (2023) 26:249–274
https://doi.org/10.1007/s12650-022-00868-1

REGULAR PAPER

Daniel Limberger · Willy Scheibel · Jürgen Döllner · Matthias Trapp

Visual variables and configuration of software maps

Received: 2 February 2022 / Revised: 25 May 2022 / Accepted: 4 July 2022 / Published online: 16 September 2022

Abstract Software maps provide a general-purpose interactive user interface and information display in software analytics. This paper classifies software maps as a containment-based treemap embedded into a 3D attribute space and introduces respective terminology. It provides a comprehensive overview of advanced visual metaphors and techniques, each suitable for interactive visual analytics tasks. The metaphors and techniques are briefly described, located within a visualization pipeline model, and considered within a software map design space. The general expressiveness and applicability of visual variables are detailed and discussed. Consequent applications and use cases for different software system data and software engineering data are discussed, arguing for the versatile use of software maps in visual software analytics.

Keywords Software visualization · Treemaps · Visual variables · Design space

1 Introduction

Treemaps denote a group of well-known, widely used, versatile techniques for visualizing information. They enable the presentation of extensive, non-spatial, tree-structured data and reliably serve thematic maps' familiarity, convenience, and informative value. In its most common 2D form, data are mapped to color and area. They can also be embedded into the third dimension, e.g., using cuboids or pyramids (Turo and Johnson 1992), resulting in so-called 2.5D treemaps (Fig. 1) or 3D-embedded treemaps. These offer further possibilities for mapping and overlaying data or visual variables (Carpendale 2003). 2D and 2.5D treemaps alike can support visual analytics to "foster the constructive evaluation, correction, and rapid improvement of our processes and models and—ultimately—the improvement of our knowledge and our decisions" (Keim et al. 2008). Treemaps are a tool that is used in many variations and different domains (Shneiderman 2009). For example, they have been used to depict file systems (Johnson and Shneiderman 1991), stock markets (Wattenberg 1999), controller performance data (Shah et al. 2005), health data (Chazard et al. 2006), demographics (Jern et al. 2009), business intelligence (Vliegen et al. 2006; Roberts and Laramee 2018), and software development (Merino et al. 2018). This has ultimately led to treemaps being increasingly integrated into well-known charting tools, frameworks, and libraries.

D. Limberger (✉) · W. Scheibel (✉) · J. Döllner · M. Trapp
Digital Engineering Faculty, Hasso Plattner Institute, University of Potsdam, Potsdam, Germany
E-mail: daniel.limberger@hpi.uni-potsdam.de

W. Scheibel
E-mail: willy.scheibel@hpi.uni-potsdam.de

J. Döllner
E-mail: juergen.doellner@hpi.uni-potsdam.de

M. Trapp
E-mail: matthias.trapp@hpi.uni-potsdam.de

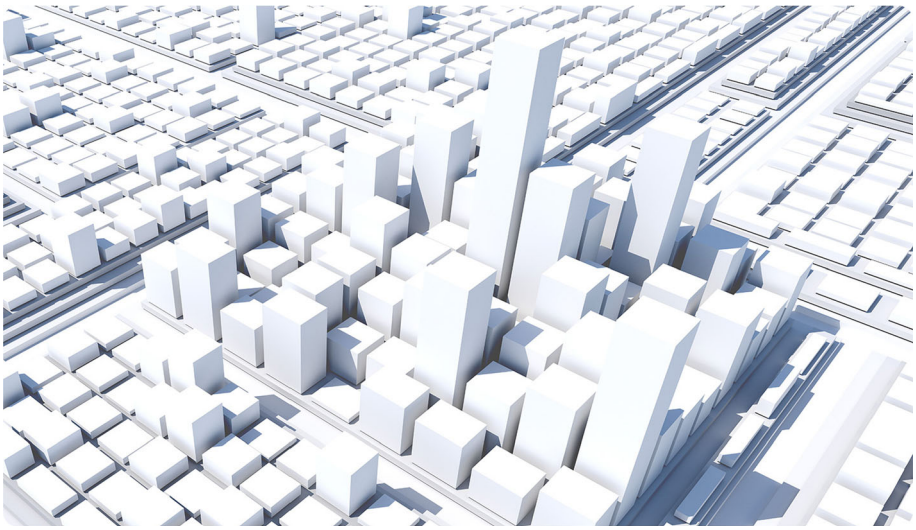

Fig. 1 An illustration of a 2.5D software map with a software metric mapped to the height of cuboids (leaf nodes) representing software modules, thus making it a *software map*. Neither color nor area is used for attribute mapping. Instead, it shows the expressiveness of additional visual cues, i.e., floating nodes and shadows, that can enhance the synthesized gestalt of a software system

In software analytics, treemaps are used to depict software system and software engineering data, resulting in so-called *software maps* (Limberger et al. 2019a). Software maps are a general-purpose interactive user interface for information display in software analytics. Typically, a catalog of *software map themes* compiles commonly used attribute selections and mappings to visual variables relevant to specific tasks. A *software map theme*, hereinafter referred to as *map theme*, defines a selection of software information dimensions mapped onto a software map's visual variables. It portrays selected aspects of the software information gathered and analyzed by software analytics processes. In a sense, a map theme presents a topic or use-case-specific software map template that supports different stakeholders in software engineering in data-driven decision making.

To advance towards the goals of visual software analytics, i.e., "(1) derive insight from massive, dynamic, ambiguous, and often conflicting data, (2) detect the expected and discover the unexpected, (3) provide timely, defensible, and understandable assessments, [and] (4) communicate assessment effectively for action" (Keim et al. 2008), we extend on the recent work of Limberger et al. (2019a) and derive the following research questions for software maps:

- How can data be depicted beyond the primary use of area, height, and color?
- To what extent can techniques be used individually and in combination?
- How does the data type impact the utilization of certain visual variables?

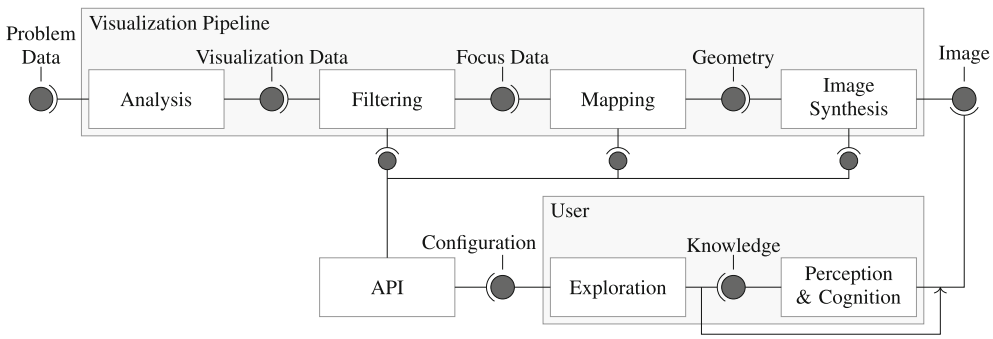

Fig. 2 Model of the visualization process (van Wijk 2005) consisting of a visualization pipeline (dos Santos and Brodlie 2004) and a feedback loop for knowledge gathering and interactive user control. This model applies to visual software analytics using software maps

– What are helpful configurations and variations of software maps?

To answer these questions, we discuss advanced visual metaphors and techniques for software maps by exploring the underlying visualization pipeline. A superimposed visualization process integrates a visualization pipeline into a feedback loop with interactive user control of the various stages (van Wijk 2005). Such a visualization pipeline is a sequence of three to four data processing stages with named inputs and outputs that conceptualize the transformation process of data into images. This visualization process is well understood, and there are a few variations of models of visualization pipelines: When using a data state reference model, the stages can be referred to as transformations, namely (1) data transformation, (2) visualization transformation, and (3) view or visual mapping transformation (Chi and Riedl 1998; Card et al. 1999; Chi 2000). An extensive overview of specific transformations suitable for visualization purposes was provided by Liu et al. (2017). A similar naming schema uses (1) data enrichment/enhancement, (2) visualization mapping, and (3) rendering to describe the same process (Haber and McNabb 1990). This schema was later refined to (1) data analysis, (2) filtering, (3) mapping, and (4) rendering (dos Santos and Brodlie 2004). We use the latter model and refer to the first two stages as *preprocessing and filtering*. *Preprocessing and filtering* cover, among others, analytics, selection, enrichment, transformation, and resampling of raw data. The *mapping* stage maps abstract data to visual representations (visual variables and configuration). Finally, *rendering* creates a visual representation using image synthesis (cf. Fig. 2).

Using this model, we provide an overview of advanced visual metaphors that have already been or can be used in combination with software maps to approach the challenges mentioned above. We briefly summarize each metaphor and technique, discuss it in the context of a stage, and describe everyday practices based on our experience from research and industry projects. We discuss the extent to which each metaphor or technique can be used to communicate different types of data and identify feasible combinations of isolated techniques and possible interferences, spanning a design space of software maps.

The remainder of this paper is structured as follows: Sect. 2 outlines the concept of software maps. The following sections cover algorithms and techniques applied for software maps. Section 3 covers filtering and preprocessing techniques. Section 4 introduces layout algorithms associated with software maps. Design decisions for mapping software data to visual primitives are discussed in Sect. 5. The rendering-related techniques are discussed in Sect. 6. Configuration and assembly of software maps are discussed in Sect. 7, and in Sect. 8, this paper concludes.

2 Definition and classification of software maps

The term *software map* is not uniquely defined in software cartography. Each variation, such as code cities (Wettel and Lanza 2008), software cities (Steinbrückner and Lewerentz 2013), or thematic software maps (Kuhn et al. 2008), focuses on their specific intended use, i.e., visualizing abundant software system and process data and, thereby, providing a communication artifact for software engineering. "A single graphic can convey a great deal of information about various aspects of a complex software system, such as its structure, the degree of coupling and cohesion, growth patterns, defect rates, and so on" (Kuhn et al. 2008). In order to obtain a more precise, low-level definition of *software maps*, we propose a definition that denotes them as a subset of treemaps using existing systematization approaches.

Schulz et al. (2011) identified the following axes for the design space of implicit hierarchy visualizations: dimensionality (either 2D or 3D), node representation (graphics primitives), edge representation (inclusion, overlap, and adjacency), as well as hierarchical layout (subdivision and packing). Dübel et al. (2014) differentiate between reference space and attribute space for a more precise classification—assuming the validity of their classification is not affected by the non-spatial nature of treemaps. Recently, Scheibel et al. (2020c) proposed a systematization of tree visualization techniques and treemaps, including space-filling treemaps, containment treemaps, implicit edge-representation trees, and mapped trees.

Using these design spaces, the software map can be expressed as follows. For the spatialization of nodes of a software map, we prefer subdivision or packing within a 2D reference space ($\mathcal{R}^2$).

The representation of edges is assumed to be implicit through nesting or inclusion, rendering a software map's underlying layout a *containment treemap* layout (Figs. 3 and 4). Some exceptions include a layout that encodes the parent–child relationship using adjacency, resulting in *implicit edge-representation trees* as software maps. Geometric adjacency in 2.5D maps can also be used (cf. Fig. 1), but due to their implications with height mapping, we suggest considering its use carefully (Limberger et al. 2018b); eventually, we

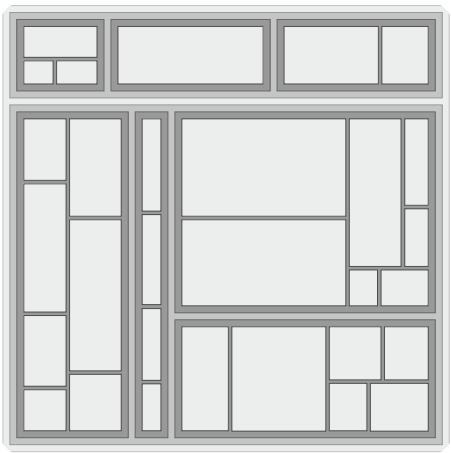

Fig. 3 Space-filling treemap $\mathcal{T}_S$

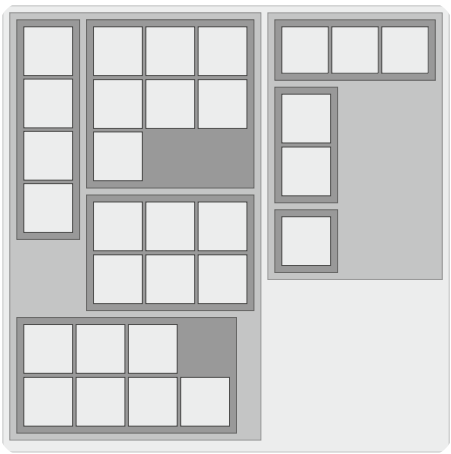

Fig. 4 Containment treemap $\mathcal{T}_C$

use flat representations of inner nodes only for all map themes in order to reduce occlusion and allow for particular representation of aggregates (level-of-detail). Techniques based on containment and overlap in 3D are marginally helpful for large data sets due to added interaction complexity and hard-to-resolve occlusion. Fortunately, this restriction does not exclude commonly used treemap-layouting algorithms based on rectangular or polygonal shapes (Hahn and Döllner 2017). For the representation of leaf nodes, 2D graphics primitives are common, e.g., Voronoi shapes or rectangles ($\mathcal{A}^2$). Embedded in 3D, they can be extruded, optionally tapered, and thus use the third spatial dimension ($\mathcal{A}^3$). "Increasing the visual vocabulary can provide for richer information resolution" (Turo and Johnson 1992) and allow for additional information display. For these 2.5D software maps, we generally prefer graphical primitives of rectangular footprints over more complex ones (cf. Fig. 25a)—depending on (1) the importance of stability for the respective map theme and (2) whether or not rendering of more complex geometry is justifiable in terms of visual complexity, implementation complexity, as well as interactivity and responsiveness of the visualization. Since the reference space is still in $\mathcal{R}^2$, we support the terms *2.5D* (Turo and Johnson 1992) and *3D-embedded* to avoid confusion with actual 3D treemaps that use 3D layouts for positioning graphical elements in a 3D reference space ($\mathcal{R}^3$).

With the attribute space and reference space being constrained, the typical characteristics of $\mathcal{A}^2 \otimes \mathcal{R}^2$ and $\mathcal{A}^3 \otimes \mathcal{R}^2$ visualizations adhere to software maps. Consequently, we define software maps as a subset of 2D and 2.5D containment treemaps—with some exceptions being implicit edge representation trees—for the targeted mapping of abstract software system data and software engineering data as well as software development data. This gives one outstanding benefit: their characteristics largely match these of

ubiquitously available, interactive 2D and 2.5D (geo-spatial) maps. Furthermore, with this definition of software maps, both visualization engineers and software engineers can rely on and build upon familiar (1) interaction metaphors, (2) provisioning strategies, and (3) visualization metaphors, drawing on their habits and experiences.

2.1 Data characteristics

Traditionally, visualizations targeting software analytics are required to handle three essential aspects of software system data: static, dynamic, and evolving aspects (Diehl 2007). We use a broader scope for software maps by including software development data and all stakeholders of the software development process. The data typically covers mined, preprocessed data gathered from various sources:

- Software implementations: typically modularized, available as distributed tree-structured components and source code units. In addition to topology information, metrics and other *key performance indicators* (KPIs) are measured and derived by applying static source code analysis to the programmed artifacts, namely source code, scripts, binaries, and documentation.
- Software executions (*traces*): e.g., analysis of run-time execution to capture and measure a software's behavior.
- Software development processes and evolution: e.g., analysis of the engineering work on the system done by software developers. This data is usually available in revisioning systems and issue, bug, and customer feedback tracking tools.

Even if measured precisely, the data (1) are prone to uncertainty, (2) are vague semantic normalization, (3) lack standardized metrics and interpretations, and (4) are highly language-specific and, thus, highly heterogeneous in large software systems. Furthermore, anonymization may be required, the resolution of measurements is unnecessarily high, and the data quickly becomes extremely extensive and complex even for medium-sized projects (metrics per file and per commit, every change with respect to issue tracking, continuous integration). These characteristics should be considered when specifying map themes or developing visualization techniques for visual information display using software maps. An overview of visualization techniques for static aspects and their evolution, including visualization techniques besides treemaps and software maps, is listed in surveys by Caserta and Zendra (2011) and Merino et al. (2016).

2.2 Preliminaries and assumptions

This paper does not provide a comprehensive set of design guidelines for creating treemaps regarding all aspects of visualization design. Guidelines concerned with perception (Kong et al. 2010) or use of color (Xie et al. 2018; Schloss et al. 2018; Quinan et al. 2019) are not specific to treemaps but relevant for every visualization and, thus, not the focus of this paper. Especially for color, we suggest referring to geographic, thematic maps and commonly used, sequential, diverging, and qualitative color schemes.

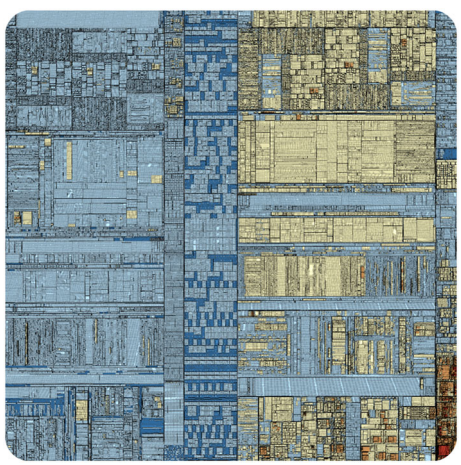

Fig. 5 Depiction of a software system with hundreds of thousands of software artifacts

Our selection of discussed visualization techniques focuses on metaphors and techniques that cope with the data characteristics within software engineering, i.e., volume and complexity of the data (Fig. 5). Therefore, we prefer techniques that have already been introduced for software data and can handle data sets with lots of data elements. Similarly, we focus on visualization techniques that entail abstract depictions, i.e., we follow the approach of mapping abstract software data—data without any natural *gestalt*—to abstract visual entities. This paper does not cover city visualizations or city-like metaphors (Knight and Munro 1999; Balogh et al. 2015) that map abstract data to features of virtual 3D cities such as building facades, car or pedestrian traffic, or street furniture.

2.3 Software maps and landscapes

As one example of a software project, the Qt 5.2.1 release from February 2014 concluded over 19 years of development from over 1400 developers that created more than 120,000 commits. The entire functionality is distributed over more than 145 000 source files with over 21,000,000 lines of code (Fig. 6). We see this project as a large-sized software project, typical in industry contexts and, for now, widely used open-source software. Although Qt contains various dependencies within its source code, the more recent approach would be to use software packages, which hide a large part of the source code relevant when working on projects. Much of the source code is present in today's software through dependency on software packages. This is important to keep in mind when exploring metrics using software maps. Software maps and, generally, most depictions based on a software's underlying hierarchy or structure tend to obfuscate a software's overall complexity since they only cover its top-most layer. The dependency tree of an NPM package, for example, has an average depth of 4.39 and a size of 86.55 (Vaidya et al. 2019). This quickly results in the tenth of thousands of additional source and configuration files (i.e., located in the node_-modules directory) that are usually not incorporated in the software analysis.

Whether it is more or less adequate to focus on a slice of software or cover all of it depends on the task and needs to be researched in more detail. Extending the idea of software maps to a *Google-Earth*-like service to explore millions of software projects as a landscape, planet, or galaxy is tempting. From an implementation perspective, available frameworks and tools need to scale to large and massive datasets. This includes preprocessing and layouting (Scheibel et al. 2021), hardware-accelerated rendering, and dynamic updates of the geometry (Trapp et al. 2013; Scheibel et al. 2017). Similarly, the deployment and provisioning of data and visualization are domain and application-specific (Limberger et al. 2013, 2016b). However, a generalized view of application scenarios allows to derive common characteristics and shared approaches for visualization, including visualization services (Scheibel et al. 2020a). Based on these

Fig. 6 A software map depicting the source code of the Qt 5.2.1 release in 2014. The map contains over 145 000 leaf nodes and covers most dependencies such as sqlite3 and assimp. The software data was extracted from the GitHub mirror github.com/qt/qt5 with resolved submodules

techniques, the foundations for a scalable and much more comprehensive exploration already exist. For the following techniques in the phases of preprocessing, layouting, mapping, and rendering, implementations should focus on scalability.

3 Preprocessing and filtering

The incoming data, also known as the problem or task data, is transformed and prepared for visualization during preprocessing and filtering. This includes resampling, normalization, filtering outlier, and accumulating weighted leaf-node data to inner nodes. The main goal is to prepare use-case-specific data augmentation, value resolution, and data residence. The latter two are essential for visualization systems that process large amounts of data.

3.1 Use-case-specific handling of data

Typical tasks for preparing the data for visualization result from the requirements of the visualization techniques used or the use case at hand. Prevalent examples are area mapping in treemap layouting algorithms where a value of an inner node is expected to be the sum of all children's weights (Johnson and Shneiderman 1991) and mapping of additional visual variables on inner nodes to display aggregated information (Limberger et al. 2017b). When dealing with data entailing uncertainty, the aggregation of values should employ an uncertainty model (Görtler et al. 2017). Especially for values that are used for area mapping, there are additional techniques to manipulate the values, e.g., to allow for (1) user-controlled sizes (Turo 1994), (2) scaling of otherwise invisible nodes (Csallner et al. 2003), (3) equalization of size through "atrophication" (Bladh et al. 2004), (4) employing a degree-of-interest (Schlechtweg et al. 2004), and (5) pre-allocating weights for the display of inner nodes (Yang et al. 2015).

Another example is the preprocessing of tree-structured topology. A common approach is to process the topology through the filtering of nodes. This reduces the display of information, selecting relevant subtrees and omitting irrelevant nodes (Veras and Collins 2017; Limberger et al. 2017b), e.g., detection and removal of source code duplicates, generated code, automated activities, and filtering of irrelevant areas such as third-party code. For the particular case of inner nodes having an inner node as a single child, tree pruning can be applied to simplify the topology (Bladh et al. 2004).

3.2 Value resolution

While assembling software maps in practice, we encountered several challenges. These include handling large amounts of data and ensuring reasonable memory usage in both main and graphics memory. Fortunately, the required data resolution for the visualization is usually much lower than the resolution of the input data. In these cases, we suggest reducing the data to a resolution appropriate for the map theme, e.g., attributes mapped to area, color, or height do not require a 32-bit floating-point resolution. Often, a reduction of attribute resolution to a few bits can increase readability by means of discriminability and emphasize the results of the preceding data analysis. This can also be supported from a perceptual point of view: height with perspective foreshortening is hardly comparable on a per-pixel basis. The exact size of a module within a software map should be of subordinate importance. Thus, most often, a transformation to a categorical data type such as *irrelevant*, *low*, *medium*, and *high* (for color, height, or change) or *lower-outlier*, *below average*, *average*, *above average*, and *upper-outlier* (for area) are easily comparable and more effective (Wettel and Lanza 2007).

3.3 Streaming and level-of-detail

The major challenge is that industry software projects tend to be massive in terms of the number of modules, code units, metrics, or activities. This is often ignored or only marginally covered in research but significantly impacts visualization design and software map assembly. Software maps are likely to be streamed on-demand (e.g., node by node, slice by slice). Loading millions of attribute values for nodes that should not even be depicted, e.g., as they are out of scope or sub-pixel size, can decrease responsiveness and result in visual clutter or memory shortages. It is advantageous to have a customizable, interactive level-of-detail (Elmqvist and Fekete 2010; Limberger et al. 2017b) in both filtering and rendering that allows for

dynamic detail (cf. Fig. 7). Furthermore, the software map should always support the concept of mental maps (Liu and Stasko 2010), i.e., the overall layout should not change due to a dynamic level of detail.

4 Layout

The base layout for a software map can, for example, be derived from the hierarchical structure of the software project that is captured in a tree-structured dataset. Software maps usually use inner nodes to depict applications, modules, or source code units, i.e., source code files. The leaf nodes depict modules, source code units, functions, events, developers, activity through commits, or associated data (Ardigò et al. 2021). The position of each node is computed by a layout technique specified by the map theme, and a metric is mapped to the size of the graphical primitive, e.g., lines-of-code or file size. This hierarchical structure is processed by treemap layout algorithms, resulting in treemap layouts.

Treemap Layout Algorithms for Software Maps. There are many algorithms available and under current research (Scheibel et al. 2020b). From them, mainly two-dimensional layouts are used for both 2D and 2.5D software maps. As a prominent classification, "one can discern two major layout methodologies: *subdivision* and *packing*" (Schulz et al. 2011). *Packaging layouts* use a spatial arrangement of child nodes based on proximity and the definition of an enclosing space as a separate layout. *Splitting layouts*, in contrast, define a space for the root node and dissect its layout for all of its child nodes. Another approach to differentiation is the class of shapes and metaphor for hierarchical nesting that an algorithm selects for the nodes. We see the coarse categories of rectangular splitting layouts, rectangular packing layouts, polygonal layouts, and adjacency layouts for software maps.

Choice of Software Map Layout Algorithms. The choice of a layout algorithm for a software map may depend on several factors and goals (Fig. 8). For example, the resulting whitespace can be the target for reduction, create more compact software maps, create and maintain landmarks, or as space for flexibility during layouting, e.g., allow for later insertions of nodes. The treemap layouts are a target for general optimization for treemap layout metrics (Vernier et al. 2020). However, the one main parameter is the choice of the leaf node type, as it dramatically influences the metaphor and general look of the software map. This results in software maps using either strict structures (space-filling rectangular layouts) or uniform leaf node shapes (packing layouts). The choice of algorithms and, thus, an underlying layout parameterization allow for different metaphors for the resulting visualization.

4.1 Rectangular splitting layouts

Treemap layout algorithms were coined in the early 1990s with the later called *Slice'n'Dice* algorithm (Johnson and Shneiderman 1991). A couple of directions were researched to improve particular properties of these rectangular splitting approaches, i.e., optimize aspect ratios, preserve the order of items, and achieve stability over time (Bethge et al. 2017). However, this is still a trade-off for the current state of

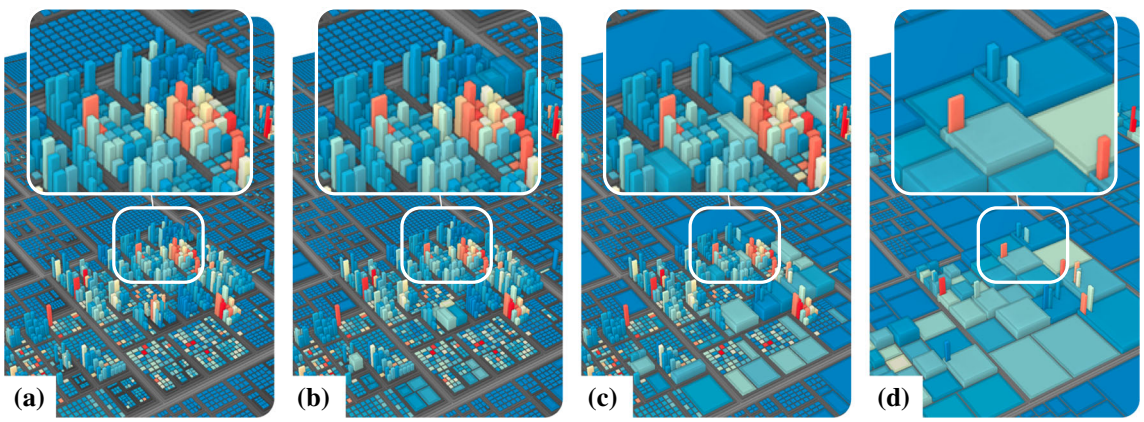

Fig. 7 Stepwise simplification of a software map using aggregation, ranging from no aggregation (left) to strong aggregation (right). The level of aggregation can be controlled using nodes of interest or, as shown here, based on the tree level. Note that outlier nodes, so-called landmarks, are preserved (Limberger et al. 2017b)

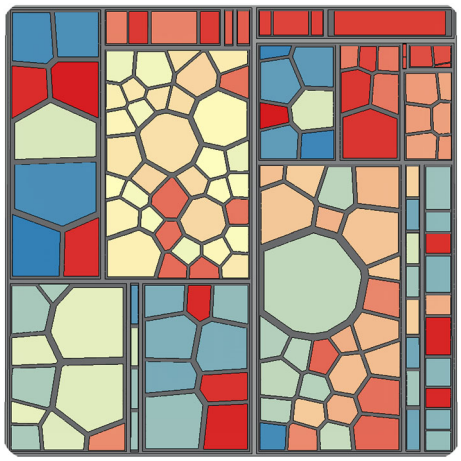

Fig. 8 Mixed splitting layout for software maps including rectangular and polygonal shapes

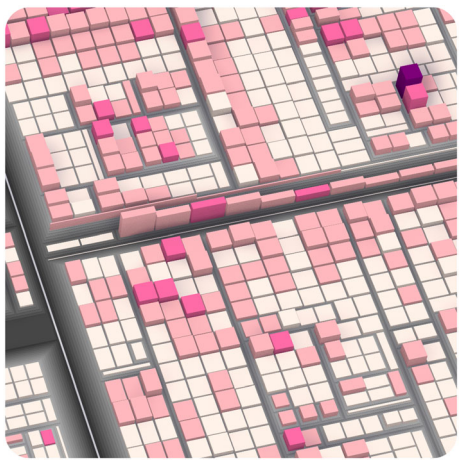

Fig. 9 Splitting layout for software maps (also usable in 2.5D)

the art, and thus, no optimal algorithm is found. For example, optimizing aspect ratios is best achieved with a reordering of nodes (Bruls et al. 2000) as this gives flexibility in layouting the nodes (Fig. 9).

Other layout algorithms are based on space partitioning approaches, such as the ordered layouts by Shneiderman and Wattenberg (2001) and Feng et al. (2019). Recent extensions are based on mathematical proofs of minimal aspect ratios for a given set of weights (Nagamochi and Abe 2007) solved by linear programming (Fügenschuh et al. 2014). Focusing more on the order of nodes, there are several algorithms based on space-filling curves. Starting with the Strip treemap building, a Z-like space-filling curve (Shneiderman and Wattenberg 2001), S-shapes, and spiral curves were introduced (Tu and Shen 2007). More sophisticated space-filling curves were applied to treemap layouting as well. As such, the Hilbert and Moore curves allow for good stability over time (Tak and Cockburn 2013) and are now on par with the majority of layout algorithms regarding the computation times (Scheibel et al. 2021).

4.2 Rectangular packing layouts

As an alternative approach to splitting, recursive packing can be used to derive treemap layouts. The most basic algorithm for this is Bin Packing (Shneiderman 2009). An optimization to this is the Data Jewelry Box algorithm (Yamaguchi and Itoh 2003), as it uses computational geometry to improve layout stability. By applying the evolution of data to a rectangular treemap layout, the EvoCells algorithm adjusts the parent rectangles for each change on leaf nodes by displacement and packing (Scheibel et al. 2018) (Fig. 10). For

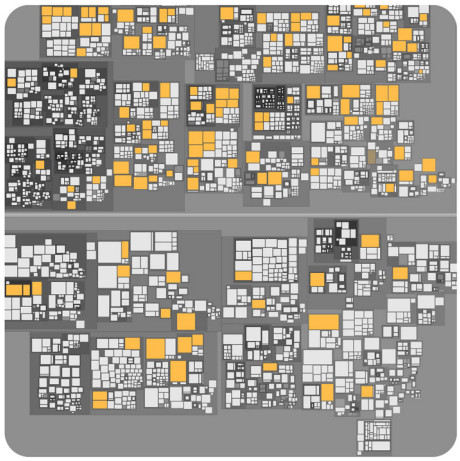

Fig. 10 EvoCells: a packing layout for software maps

packing layouts, reducing the whitespace between nodes has to be considered. If this whitespace should be reduced, layout postprocessing can be used (Domrös et al. 2021).

4.3 Polygonal layouts

Next to rectangular layouts, software maps may be constructed using general polygonal layouts. Mainly splitting algorithms are used to derive polygonal layouts, although they are not limited to this. For example, software maps with convex shapes can be constructed using Voronoi tessellation, creating Voronoi treemaps (Balzer and Deussen 2005). Another approach is space partitioning, which is used with polygonal partitions de Berg et al. (2013) or a divide-and-conquer approach Liang et al. (2015). The Voronoi variants can be used to derive orthogonal Voronoi treemaps Wang et al. (2022)—using non-convex orthogonal shapes—or build upon the layout to derive map-like visualizations, e.g., the CodeSurveyor (Hawes et al. 2015). When following the approach of space-filling curves for non-rectangular layouts, the Gosper curve can be used to create GosperMaps, which are especially stable over time (Auber et al. 2013).

A prominent extension of software maps is adjacency layouts (Fig. 11). Most of them are derived from one-dimensional treemap layouts. Using an extension to one dimension creates Icicle Plots and Flame Graphs (Gregg 2016), and a projection using polar coordinates results in Sunburst Views (Stasko et al. 2000) and Bundle Views (Holten 2006). Although a basic packing algorithm, the EvoStreets layout results in a software map using adjacency (Steinbrückner and Lewerentz 2013). These layouts are convenient at the coarser levels of a software entity hierarchy, e.g., to display whole landscapes of software projects. The

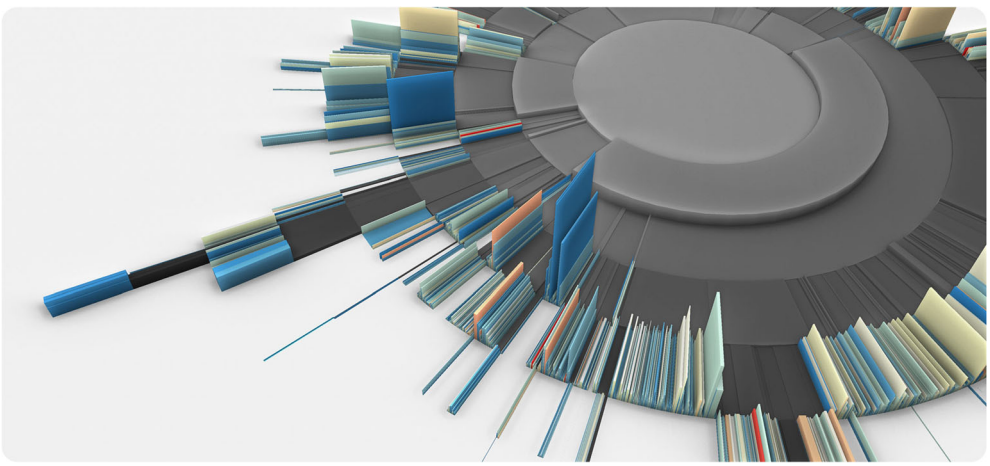

Fig. 11 An example for a adjacency layout for treemaps: a Sunburst view depicting a software system

nature of their layout allows for the explicit and complete depiction of inner nodes. The additional whitespace can be used for additional mapping of inter-node relations (Holten 2006).

4.4 Layout stability

One of the biggest challenges for software maps is stability over time, i.e., node positions remain stable over time. This is advantageous for the visual-data-correspondence (Kindlmann and Scheidegger 2014) and for maintaining a mental map (Archambault et al. 2011). Stability is achieved by maintaining topology changes and weight changes of the underlying hierarchy. Suppose the time range of interest is in the past. In that case, a multi-revision hierarchy can be computed (Tu and Shen 2007), allowing for stable positions of nodes as they can be predetermined using a layout that incorporates all nodes with their maximum weight Pfahler et al. (2020). Suppose the software map is used to visualize ongoing processes and continuously evolving data. In that case, an initial layout can be incrementally evolved based on the topology changes, resulting in nodes growing, shrinking, appearing, and disappearing (Scheibel et al. 2018). Moreover, Kokash et al. (2014) proposed an approach that ensures uniform treemap layouts over time by applying an explicit splitting scheme. A similar approach is *the layout of the treemap using only local modifications* (Sondag et al. 2018) by applying local swaps and approximating the layout of the previous instance. As an improvement to this, the Greedy Insertion approach operates on the derived layout tree and inserts new nodes next to the element with the worst aspect ratio (Vernier et al. 2018).

5 Mapping

The mapping stage transforms pre-processed and filtered data, e.g., attribute values, into depictable and reversibly encoded graphical primitives and scenes. This is a fundamental step for efficient encoding of input data and should address human capacities and abilities to decode a depiction (Ware 2012). There is not necessarily an explicit representation of the resulting visualization object in memory. The result may only be volatile during visualization, especially from an implementation point-of-view: the distinction between mapping and rendering sustains more on a conceptual level (Trapp et al. 2013; Scheibel et al. 2017). In this section, visual variables and additional techniques available for attribute mapping are briefly described and discussed. An overview of visual variables (not techniques in general) is provided and summarized in Table 1. In it, each visual variable is classified with respect to the stage typically responsible as well as its

Table 1 Legend: M—Mapping | R – Rendering | ● – Supported | ○ – Partially supported

Visual variable		Node		Dim.		Data type			
	Stage	Inner node	Leaf node	2D	2.5D	Nominal	Ordinal	Interval	Ratio
Area (foot print, size)	M	●	●	●	●		●	○	●
Color	R	●	●	●	●	●	●	●	●
Height	M	○	●		●		○	○	●
Transparency	R	○	●		●	○	○	○	●
Light emission (glow)	R	○	●	○	●		○	○	●
Stacking	M		●	●	●	●		○	●
Stacking (global layer)	M	○	●		●		○	○	●
Segments	M	○	●	●	●	●	○	○	●
Shape type	M		●	●	●	●	○		
Shape parameter	M	○	●	●	●		○	○	●
In situ (change, diff)	M		●	●	●		●		
Contour width	R	●	●	●	●		●	○	○
Contour color	R	●	●	●	●	●	●	●	●
(Contour) stippling	R	●	●	●	●	●	●	○	○
Sketchy contour	R	●	●	●	●	○	●	○	○
Surface pattern (texture)	R	○	●	●	●	●	●	○	○
Surface noise (texture)	R	○	●	●	●	○	●	○	●
Surface shading (texture)	R	●	●	○	●	○	●	●	
(Surface) hatching	R	●	●	●	●	○	●	○	○
Nesting-level margin	R	●		●	●		●	○	●
Color weaving	R	●	●	●	●	●	○		
Height threshold	R	●	●	●	●		○	○	●

applicability to (1) inner nodes and leaf nodes, (2) 2D and 2.5D software maps, and (3) the data type it most certainly can convey.

5.1 Use of height

We utilize the height of cuboids by extrusion of the 2D shape (Bladh et al. 2004; Choi et al. 2011) as secondary visual variable (Fig. 12). The idea is to convey information in the order of decreasing importance for the task: (1) color, (2) height, and (3) other visual variables. For 2.5D software maps, pyramid-like shapes can be used to further encode an attribute or reduce occlusion (Turo and Johnson 1992). When depicting evolving data with accustomed map themes, height seems to allow for an intuitive encoding of data changes by means of growing or increasing versus shrinking or decreasing, respectively. Height, however, should not be used to depict negative or diverging scales directly, as this would result in downward-facing cuboids. Although suggested several times, we prefer to refine the map theme to transform or map relevant ranges of values of attributes to make a salient but expected use of visual variables: cuboids are high when the underlying data is interesting. In contrast to that, if a negative value range is relevant, it is mapped inversely to height. If positive values are relevant, the absolute value could be mapped to height and the sign to color, shape, or texture. Finally, the orientation of the leaf node's geometry (Langelier et al. 2005) and the type, employing poly cylinders (Marcus et al. 2003) and three-dimensional glyphs (Boccuzzo and Gall 2007) are further suitable as visual variables.

5.2 Juxtaposing and complex shapes

For a visual display of multiple states or sub-elements, data vases, stacked cuboids, or segmenting/fragmenting can be used. In remembrance of stacked bar charts, the extruded polytopes can be subdivided in height, allowing for the depiction of subcategories and their share of the overall height (Itoh et al. 2006; Giereth et al. 2008). This process can further be utilized, e.g., to encode evolution using *evolution segments* (Steinbrückner and Lewerentz 2013) and *data vases* (Thakur and Rhyne 2009). Small multiples can be used for "the comparison of multiple software map themes and revisions simultaneously on a single screen" (Scheibel et al. 2016). We found this especially useful for the generation of an overview as well as exploration and identification of map theme variations for software map assembly.

5.3 Mixed projections

Recently, an approach for focus-sensitive use of 2D and 2.5D treemaps was presented. "The technique operates by tilting the graphical elements representing inner nodes using affine transformations and animated state transitions" (Limberger et al. 2017c). This allows for on-demand separation of high-detail focus areas and context areas (Fig. 13). It further reduces occlusion issues—which might occur for a particular data set or map theme—and the complexity of navigation. Whenever occlusion becomes an issue, this

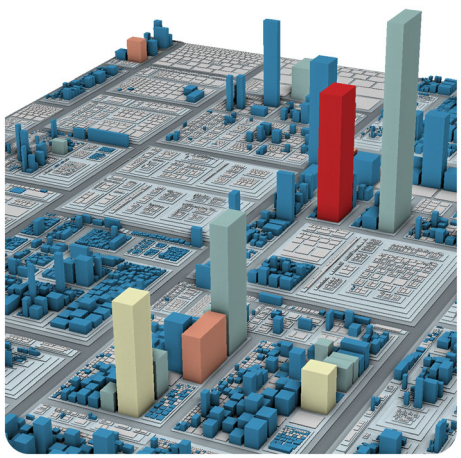

Fig. 12 Using height as a visual variable by extruding the 2D shapes up (2.5D)

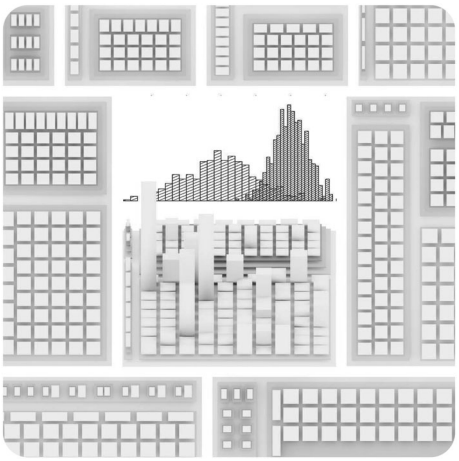

Fig. 13 Inner node tilted into 2.5D (mixed projection) combined with a local histogram

usually hints at an inappropriate scaling, camera angle, or suboptimal map theme specification. In almost all cases, occlusion issues can be resolved by applying simple mitigation strategies.

For software analytics, mixed projection can facilitate communication of the source code modules that target change for the current sprint. In this example, the context is the other source code files present in the project. Similarly, mixed projection can be used if the height mapping is irrelevant for the context but highly relevant for focus, e.g., using the estimated time to invest for a source code module to be mapped on height. Another technique to combine aggregated and detailed views with reduced occlusion is the *lifted map* (Chaudhuri and Shen 2009).

5.4 Depicting change

Many software map themes are designed to facilitate understanding of the evolution of the underlying data sets (Fig. 14). This eventually requires for visual comparison of (1) multiple visual variables for (2) multiple points in time over (3) large time ranges. Therefore, Tu and Shen (2007) introduced contrast treemaps. Their approach could be applied within visual software analytics by encoding two different states of the attribute mapped to area and color. This idea was extended by in situ templates (Limberger et al. 2019b), which allow for the visual encoding of former and latter states on a per-cuboid basis for rectangular 2.5D software maps (cf. Fig. 15). Specifically, basic and more complex templates were introduced to simultaneously depict changes in more than one visual variable, including area, color, and height. Although not discussed in the paper, the templates allow for the comparison of additional per-node visual variables as well. The visual

Fig. 14 In situ template depicting changes in height

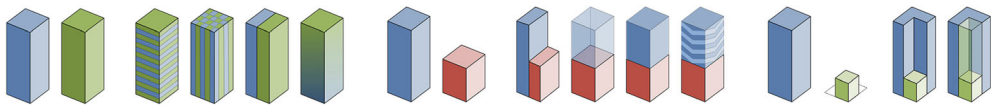

Fig. 15 A set of in situ templates to depict changes of up to three attributes using three or more visual variables (Limberger et al. 2019b). Left: templates for color-only changes. Middle: templates for color and height changes. Right: templates for changes in all area, height, and color

quality and expressiveness thereby depend on the available rendering techniques, e.g., procedural texturing and transparency are accounted for by several templates (Limberger et al. 2021).

In practice, most comparison templates would result in a complex mapping and should be used for complex tasks or expert systems only. For example, depicting area changes in combinations with others is challenging to perceive. However, they allow for a convenient encoding of changes in height that are usually mapped from secondary information such as the number of authors or the size of domain logic. Another use is to depict changes in color for preliminary information, i.e., the most prominent indicator for the primary purpose of the map theme, such as the maintainability or faultiness of a software module.

5.5 Depicting relations

For an emphasis on the topology with respect to the nested structure of nodes, cushion shading (van Wijk and van de Wetering 1999) or hierarchical stippling (Sondag et al. 2020), variations of margins or padding, as well extruded, stacked inner nodes (Bladh et al. 2004) can be applied. If other relations (e.g., Fig. 16) of nodes in addition to their tree-structured topology are of interest (e.g., functional dependencies or often-coinciding changes during the development process), edges or edge bundles can be superimposed to software maps (Holten 2006; Steinbeck et al. 2019). This approach, however, is visually constrained by the number of depicted relations and data set size. Superimposing relations using tubes on top of 2.5D treemaps introduces additional clutter and visual complexity not inherent to the treemap metaphor.

6 Rendering

In this stage, the visualization object is transformed into images using (real-time) image synthesis. Although the mapping stage is usually aware of the available visual variables and metaphors, it only provides appropriate attribute values, descriptions of graphic primitives, and additional data for rendering. As a result of the rendering stage, the mapped attributes should be visually encoded in the output image.

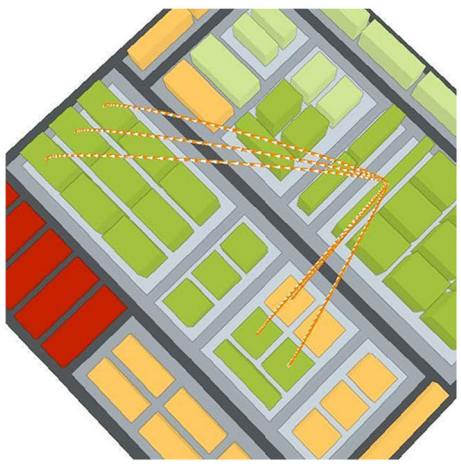

Fig. 16 Modules included by another module are connected using directed tubes (textured with procedurally generated arrows) in 3D

6.1 Utilizing texture patterns

Different texture patterns can be applied to encode categorical data (Molli et al. 2001) (Fig. 17). Exemplary attributes for texture pattern mapping are (the abstract) file type (e.g., source code, documentation, deployment), associated teams, or budgets. Alternatively, texture intensity was suggested as a visual variable for scales with a natural zero (Holten et al. 2005). With most modern rendering engines supporting physical-based materials, textures can be used to enable visual variables based on their metalness or roughness (see natural metaphors). In the past, we predominantly used procedural textures to depict quality measures (e.g., degree of documentation, error-proneness), complexity measures (incoming or outgoing dependencies, mixed-use of third-party dependencies), and development process indicators (e.g., number of authors). Procedural texturing can further be used to superimpose rulers or stripes, making the height effectively countable and increasing the accuracy of comparability. Another approach is to use procedural textures to encode underlying data distributions (Limberger et al. 2020).

6.2 Utilizing emissive light

Emissive lighting can be an intuitive metaphor for activity: if something lights up, something is going on (Fig. 18). We used this to display development activity, e.g., through commits per day or the number of issues referencing a module. This visual variable can also be used to emphasize (highlight) nodes due to user interactions such as filtering and selection or to highlight system activity (Dashuber and Philippsen 2021). However, today's real-time rendering systems are often incapable of rendering convincing light emissions, making this visual variable challenging to use.

6.3 Using sketchy rendering

For the visual display of uncertainty, fuzzy drawing styles can be applied. For example, node contour width, as well as multiplexed frequency and amplitude, could be parameterized (Görtler et al. 2017) and used to indicate vagueness. In addition, pencil-like outlines and hatching of surfaces can be combined into a single visual variable, i.e., *sketchiness*. Mapping data to sketchiness is instrumental in encoding "uncertainty, imprecision, or vagueness" (Limberger et al. 2016a). It allows to encode varying degrees of uncertainty and can be used in addition to color (Fig. 19). For example, it can display measurement inaccuracies (incomplete or estimated data) or target data that is yet to be discussed or implemented (i.e., targets for complexity, coverage, performance, or other quality measures).

6.4 Utilizing transparency

Regarding geometric processing of inner nodes, some approaches extrude the layout as well, creating platforms (Andrews 1995) or pyramid frustums (Andrews 2002), and placing child nodes *on top* on the

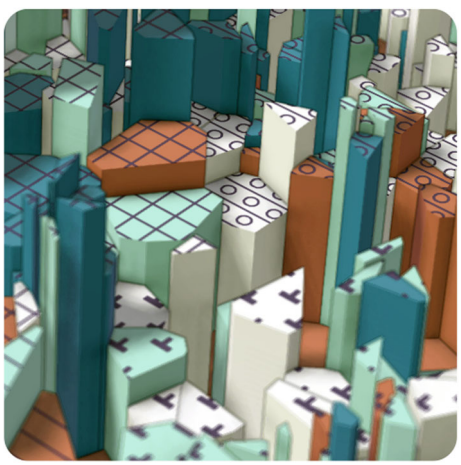

Fig. 17 Top faces textured with patterns (nominal mapping)

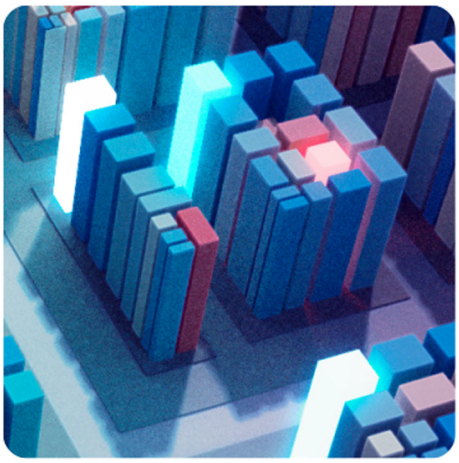

Fig. 18 Emissive light (glow) as a visual variable in 2.5D

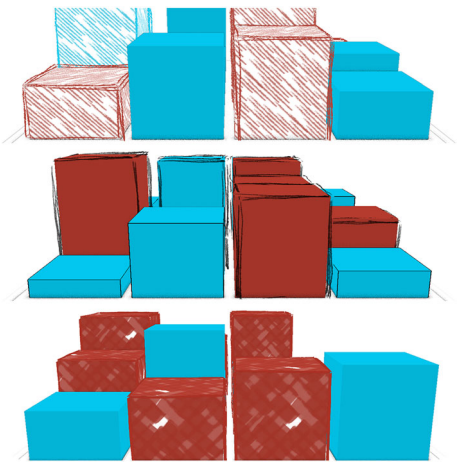

Fig. 19 Sketchy outlines and hatching to depict uncertainty

platforms. Instead of stacking, there are approaches for nested depiction as well, e.g., full spheres (Balzer et al. 2004) and hemispheres (Balzer and Deussen 2004) in combination with transparency. Other uses of transparency are (1) to reduce occlusion and (2) to encode different node states (Luboschik and Schumann 2008). Transparency can further be used to depict removed or planned components, goals, and irrelevant nodes or enhance the expressiveness and quality of texturing (Fig. 20). The most relevant obstacle to using transparency is its implementation complexity: especially for web-based rendering clients, modern strategies such as order-independent transparency cannot easily be implemented due to limited graphics APIs and device capabilities. One solution is to use stochastic dithering combined with multi-frame sampling (Limberger et al. 2017a).

6.5 Depicting aggregates and structure

As previously discussed for the preprocessing and filtering stages, aggregation is an essential technique for software maps (cf. Fig. 21). In the context of rendering, we suggest adhering to aggregation guidelines (Elmqvist and Fekete 2010). For example, the depiction should discern leaf nodes from aggregated ones. In addition, small charts, diagrams, or glyphs encoding information of the underlying data can be expressed on top or within aggregates (Soares et al. 2020). Likewise, noise or color weaving (Hagh-Shenas et al. 2007) as well as nesting-level contours—multiple contours that hint at the depth of the aggregated sub-tree, i.e., the number of an aggregate's subjacent hierarchy levels—can be used as well (Limberger et al. 2017b).

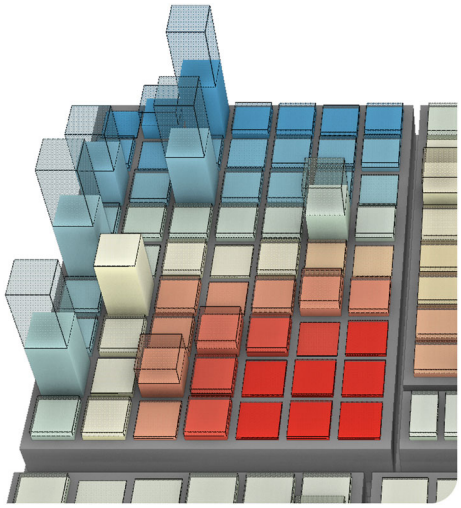

Fig. 20 Transparency (stochastic) as a visual variable in 2.5D

The aggregation should summarize contextual data of low variation for software maps to provide orientation without distraction. Aggregation can further extend user control and interaction, e.g., folding and unfolding, as well as dynamically scaling the rendering load to comply with performance constraints.

6.6 Height-based filtering

For 2.5D software maps, a reference surface, a height reference (Fig. 22) can be used to "facilitate accurate identification of highest nodes as well as similar nodes [...]. Furthermore, it allows filtering and selection of nodes based on their height and depicts filtered and unselected notes in a clean way without introducing additional visual clutter" (Limberger et al. 2018b). Depending on the renderer's capabilities and the already occupied visual variables, one of the various approaches for the visual display of the height reference can be applied, e.g., intersection, stilts, explicit surface, closed surface, and implicit surface. Using user-induced aggregation (by means of folding and unfolding) in combination with an interactive height reference can provide a sufficient foundation for exploration-heavy tasks and map themes for single software maps and whole software map landscapes.

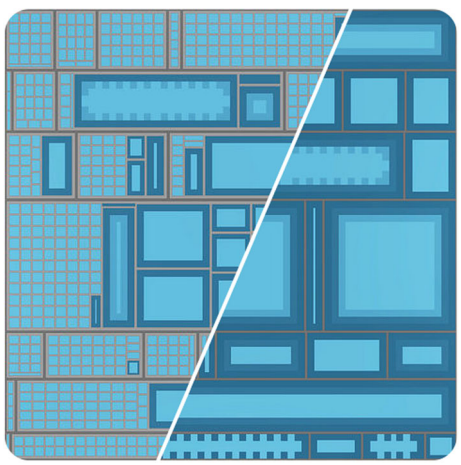

Fig. 21 Aggregation and nesting-level contours in a 2D treemap

Fig. 22 Reference surface for height-based filtering in 2.5D

6.7 Using natural metaphors

Natural metaphors are based on the idea that "when faced with unfamiliar concepts, our cognitive system searches for the best mapping between the unknown concept and existing knowledge of other domains" (Zhang 2008). To this end, natural metaphors, such as physical-based materials (e.g., rust, radiant emittance, roughness, and shininess; Fig. 23) or weather phenomena (e.g., rain, clouds, fire, dust, snow) were suggested to "emotionalize the visual communication by providing memorable visualizations" (Würfel et al. 2015). These metaphors can be used to depict change predictions or deviations from expected values on a secondary visual variable while considering knowledge or best practices on preattentive processing of, e.g., numerical information (Few 2004; Hesse et al. 2017). However, weather phenomena require sophisticated rendering techniques, renderers, and an interactive context. Further, their use might be inappropriate in certain professional contexts and even distracting for the map theme.

7 Configuration of software maps

When assembling use-case-specific map themes, implementation and provisioning have to be considered. For example, most visual variables have to be tweaked for the actual screen size, pixel density, and number or average size of nodes. In addition, all visual variables used have to be (re)adjusted within the context of their specific combination and their intended task. A few general remarks are provided in the following,

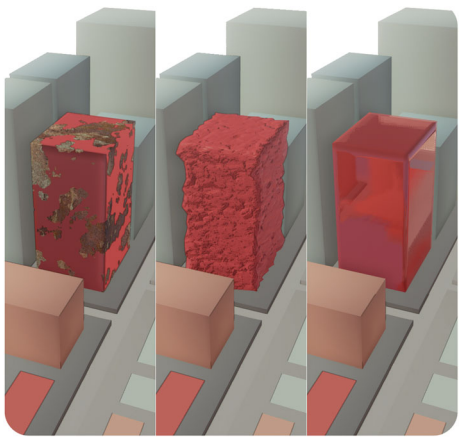

Fig. 23 Surface condition (rusty, shiny) as visual variable

which can be considered regardless of the specific visual variables used to assemble more effective map themes.

7.1 Labeling and text

More than twenty years ago, Fekete and Plaisant stated that "a major limiting factor to the widespread use of information visualization is the difficulty of labeling information abundant displays" (Fekete and Plaisant 1999). At the time of writing, this is still a "major limiting factor" today in most visualizations. Different techniques that augment treemaps with text have been proposed and evaluated for treemaps. The prominent approach is the usage of the nodes' surface to integrate labels in terms of *internal annotations* (Limberger et al. 2017b) (Fig. 24). This approach usually affects the layout computation if applied to inner nodes as well (Liu et al. 2008). The complementary approach, *external labeling*, is not treemap-specific but induces common problems such as "overlaps and data occlusion" (Fekete and Plaisant 1999). Typically, external labels are positioned as hovered text in proximity to its node (Slingsby et al. 2008) or a connected line to indicate association (Bladh et al. 2004). Additionally, the size of the label may be used to encode importance (Jadeja and Muthu 2017).

One challenge remains: Integrating techniques that allow for dynamic, interactive, adaptive, and high-quality text display in 3D scenes. To this end, the Open Label Library (OpenLL) project is an essential step toward a low-level, implementation-aware specification for dynamic, hardware-accelerated rendering and smart, adaptive placement of text (Limberger et al. 2018a). Users should be allowed to not only read a text but select, copy and paste, and change within their 3D visualization.

7.2 Visual cues

Visual cues can increase the effectiveness of visualization. They can be associated with certain visual variables to convey information either more effectively or at all. Apart from that, visual cues can be added to the visual variables used. Shadows, cushion shading, paddings and margins, or shading, for example, can significantly impact the readability of software maps. For example, a 2.5D treemap that does not employ padding or some form of lighting or shading will be much harder to read. Likewise, missing shadows or outlines can make it hard to perceive the topology or differentiate individual nodes from one another. At the same time, it is not necessarily helpful for the rendering to strive for as much realism as possible since this often can cause visual clutter (e.g., the use of reflections can be somewhat distracting). In any case, the visual cues should be adjusted carefully to the map theme. The combined use of information visualization and visual cues with a claim to reality is an open field in research.

Fig. 24 Enhancing treemaps using dynamic text placement

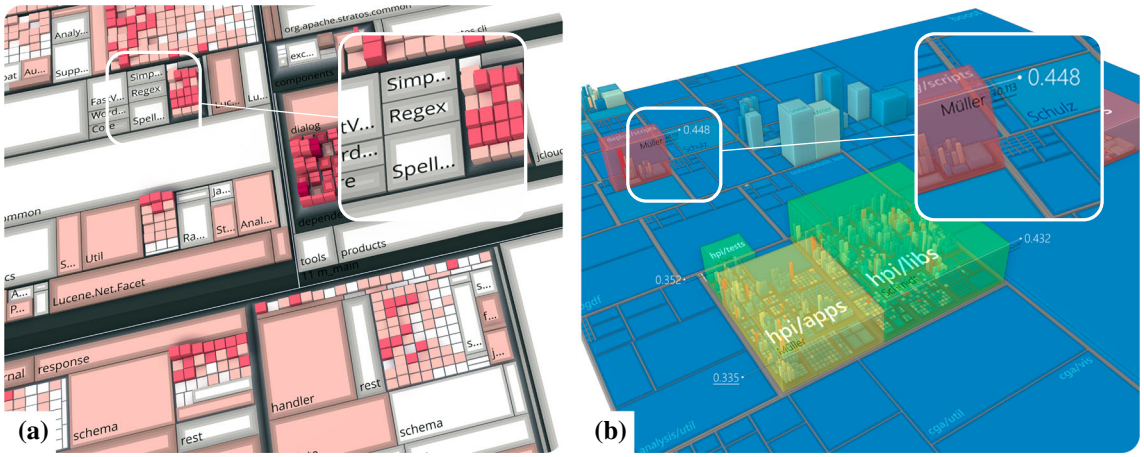

Fig. 25 Two examples for complex software map assemblies: **a** aggregation and internal labeling (Limberger et al. 2017b). The resulting space from aggregating nodes is used to embed labels into the software map and **b** a semi-transparent information overlay to superimpose and emphasize additional information

7.3 Assembling treemaps

When assembling a map theme, we usually gather all input characteristics (data types, data resolution, etc.) and specify the use case(s) and related tasks as clearly as possible. The resulting software maps differ in their used metaphors and visual variables (Fig. 25). For example, does the task consist of stepwise exploration or subsequent queries that could be addressed within the visualization directly (overview+detail, primary and secondary concerns, or root cause analysis)? Is the visualization part of a multiple-linked-view setup, preceded by other investigations, or part of a broader analytics process? What are the time and frequency the user needs to complete the task or use the visualization, respectively?

In most cases, our software maps first and foremost assist the user in identifying regions or nodes of interest. Then, additional visual variables are employed for subsequent exploration of the relevant data for those nodes. Tables 1 and 2 are intended to facilitate the identification of promising visual variables and combinations most appropriate for the task. Whether or not the visual variables can be used depends most certainly on the limitations of the targeted devices and used frameworks. As stated before, transparency, emissive lighting, and similar, technically demanding visual variables are highly convenient but still hard to implement for interactive use.

8 Conclusions

This paper reviewed advanced visual metaphors and visualization techniques for software maps. A brief discussion of each metaphor and technique is extended by common practices and own experiences by the authors. Notably, the presented techniques are modular and can be used in combination in an on-demand manner without conflicting with other techniques. This way, the software map can be used for a wide range of use cases. The approaches can visually scale from small to large and even massive data sets and allow for mapping multiple attributes for data-driven decision-making. To summarize, we argue that the software map is a highly versatile tool in visual software analytics.

For future work, we foremost acknowledge the current state of evaluation for treemaps and software maps (Fiedler et al. 2020). This motivates further evaluation of the proposed metaphors and techniques by means of user studies and studies in industry contexts. Further, we want to evaluate the applicability of the advanced metaphors and techniques to other 2D and 2.5D information visualization techniques. This includes visualization techniques from the classes of implicit edge representation trees and mapped trees for hierarchically structured software data (Scheibel et al. 2020c) and topic maps for unstructured or otherwise-structured software data Atzberger et al. (2021).

Table 2 This is-combinable-with matrix is symmetric. The data is based on the authors' assessment. Simultaneous use of four or more visual variables, however, should always be carefully considered regardless of their technical compatibility. Legend: ● – well combinable | ○ – difficult to combine

	Area	Color	Height	Trans-parency	Light emission	Stack-ing	Stacking (global layer)	Segments	Shape type	Shape parameter	In situ	Contour width	Contour color	Stippl-ing	Sketchy contour	Pattern	Noise	Shad-ing	Hatch-ing	Nesting-level margin	Color weaving	Height threshold
Area (foot print, size)	–																					
Color	●	–																				
Height	●	●	–																			
Transparency	●	○	●	–																		
Light emission (glow)	●	○	●	○	–																	
Stacking	●	●	●	●	●	–																
Stacking (global layer)	●	●	●	●	●		–															
Segments	●	●	●	●	●	●	●	–														
Shape type	●	●	●	●	○	○	○	○	–													
Shape parameter	●	●	●	●	●	○	○	○		–												
In situ (change, diff)	●	●	●	●	●			○	○	○	–											
Contour width	●	●	●	○	○	●	●	○	○	○	○	–										
Contour color	●	○	●	○	○	●	●	●	○	○	○	●	–									
(Contour) stippling	●	○	●	○	○	●	●	○	○	○	○	●	●	–								
Sketchy contour	●	●	●	○	○	●	●	○	○	○	○	●	○	○	–							
Surface pattern	●	●	●	○	○	●	●	●	●	●	○	●	●	○	○	–						
Surface noise	●	●	●	○	○	●	●	●	●	●	○	●	●	○	○		–					
Surface shading	●	●	●	○	○	●	●	○	●	●	○	●	●	●	○	○	○	–				
(Surface) hatching	●	●	●	○	○	●	●	●	●	●	○	●	●	○	●				–			
Nesting-level margin	●	○	●	○	○	●	●		○							●	●	●	●	–		
Color weaving	●	○	●	○	○	○	●	●	●	●	○	○	○	○	○	○				○	–	
Height threshold	●	●	●	○	●				○	○	○	●	●	●	○	●	●	●	●		●	–

Acknowledgements We want to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. This work is part of the "Software-DNA" project, which is funded by the European Regional Development Fund (ERDF or EFRE in German) and the State of Brandenburg (ILB). This work is also part of the KMUi project "KnowhowAnalyzer" (01IS20088B), which is funded by the German Federal Ministry of Education and Research (BMBF), and the ZIM project "TASAM", which is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi).

Funding Open Access funding enabled and organized by Projekt DEAL.

References

Andrews K (1995) Case study. visualising cyberspace: information visualisation in the harmony internet browser. In: Proceedings of the 1995 Conference on Visualization, IEEE, InfoVis '95, pp 97–104, https://doi.org/10.1109/INFVIS.1995.528692

Andrews K (2002) Visual exploration of large hierarchies with information pyramids. In: Proceedings of the 6th International Conference on Information Visualisation, IEEE, IV '02, pp 793–798, https://doi.org/10.1109/IV.2002.1028871

Archambault D, Purchase H, Pinaud B (2011) Animation, small multiples, and the effect of mental map preservation in dynamic graphs. IEEE Trans Vis Comput Graph 17(4):539–552. https://doi.org/10.1109/TVCG.2010.78

Ardigò S, Nagy C, Minelli R, Lanza M (2021) Visualizing data in software cities. In: Proceedings of the 2021 Working Conference on Software Visualization, IEEE, VISSOFT '21, pp 145–149, https://doi.org/10.1109/VISSOFT52517.2021.00028

Atzberger D, Cech T, de la Haye M, Söchting M, Scheibel W, Limberger D, Döllner J (2021) Software forest: A visualization of semantic similarities in source code using a tree metaphor. In: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Vol 3: IVAPP, INSTICC, SciTePress, IVAPP '21, pp 112–122, https://doi.org/10.5220/0010267601120122

Auber David, Huet Charles, Lambert Antoine, Renoust Benjamin, Sallaberry Arnaud, Saulnier Agnes (2013) GosperMap: using a gosper curve for laying out hierarchical data. IEEE Trans Vis Comput Graph 19(11):1820–1832. https://doi.org/10.1109/TVCG.2013.91

Balogh G, Szabolics A, Beszédes Á (2015) CodeMetropolis: Eclipse over the city of source code. In: Proceedings of the 15th International Working Conference on Source Code Analysis and Manipulation, IEEE, SCAM '15, pp 271–276, https://doi.org/10.1109/SCAM.2015.7335425

Balzer M, Deussen O (2004) Hierarchy based 3D visualization of large software structures. In: Proceedings of the 2004 Conference on Visualization, IEEE, VIS '04, https://doi.org/10.1109/VISUAL.2004.39

Balzer M, Deussen O (2005) Voronoi treemaps. In: Proceedings of the 2005 Symposium on Information Visualization, IEEE, INFOVIS '05, pp 49–56, https://doi.org/10.1109/INFVIS.2005.1532128

Balzer M, Noack A, Deussen O, Lewerentz C (2004) Software landscapes: Visualizing the structure of large software systems. In: Proceedings of the 2004 Joint Eurographics / IEEE VGTC Symposium on Visualization, EG, VisSym '04, https://doi.org/10.2312/VisSym/VisSym04/261-266

Bethge J, Hahn S, Döllner J (2017) Improving layout quality by mixing treemap-layouts based on data-change characteristics. In: Proceedings of the 2017 Conference on Vision, Modeling and Visualization, EG, VMV '17, pp 69–76, https://doi.org/10.2312/vmv.20171261

Bladh T, Carr DA, Scholl J (2004) Extending tree-maps to three dimensions: A comparative study. In: Proceedings of the 2004 Asia-Pacific Conference on Computer Human Interaction, Springer, APCHI '04, pp 50–59, https://doi.org/10.1007/978-3-540-27795-8_6

Boccuzzo S, Gall H (2007) CocoViz: Towards cognitive software visualizations. In: Proceedings of the 4th International Workshop on Visualizing Software for Understanding and Analysis, IEEE, VISSOFT '07, pp 72–79, https://doi.org/10.1109/VISSOF.2007.4290703

Bruls M, Huizing K, van Wijk JJ (2000) Squarified treemaps. In: Data Visualization 2000. Proceedings of the Joint EUROGRAPHICS / IEEE TCVG Symposium on Visualization, Springer, pp 33–42, https://doi.org/10.1007/978-3-7091-6783-0_4

Card SK, Mackinlay JD, Shneiderman B (1999) Readings in information visualization: using vision to think. Morgan Kaufmann Publishers

Carpendale MST (2003) Considering visual variables as a basis for information visualisation. Tech Rep, University of Calgary, https://doi.org/10.11575/PRISM/30495

Caserta P, Zendra O (2011) Visualization of the static aspects of software: a survey. Trans Vis Comput Graph 17(7):913–933. https://doi.org/10.1109/TVCG.2010.110

Chaudhuri A, Shen HW (2009) A self-adaptive treemap-based technique for visualizing hierarchical data in 3D. In: Proceedings of the 2009 Pacific Visualization Symposium, IEEE, PACIFICVIS '09, pp 105–112, https://doi.org/10.1109/PACIFICVIS.2009.4906844
Chazard E, Puech P, Gregoire M, Beuscart R (2006) Using treemaps to represent medical data. Stud Health Technol Inf 124:522–527
Chi EHh (2000) A taxonomy of visualization techniques using the data state reference model. In: Proceedings of the 2000 Symposium on Information Visualization, IEEE, INFOVIS '00, pp 69–75, https://doi.org/10.1109/INFVIS.2000.885092
Chi EHh, Riedl J (1998) An operator interaction framework for visualization systems. In: Proceedings of the 1998 Symposium on Information Visualization (Cat. No.98TB100258), IEEE, INFOVIS '98, pp 63–70, https://doi.org/10.1109/INFVIS.1998.729560
Choi J, Kwon Oh, Lee K (2011) Strata treemaps. In: Proceedings of the 2011 SIGGRAPH Conference – Posters, ACM, SIGGRAPH '11, p 87:1, https://doi.org/10.1145/2037715.2037813
Csallner C, Handte M, Lehmann O, Stasko J (2003) FundExplorer: supporting the diversification of mutual fund portfolios using context treemaps. In: Proceedings of the 2003 Symposium on Information Visualization (Cat. No.03TH8714), IEEE, INFOVIS '03, pp 203–208, https://doi.org/10.1109/INFVIS.2003.1249027
Dashuber V, Philippsen M (2021) Trace visualization within the software city metaphor: a controlled experiment on program comprehension. In: Proceedings of the 2021 Working Conference on Software Visualization, IEEE, VISSOFT '21, pp 55–64, https://doi.org/10.1109/VISSOFT52517.2021.00015
de Berg M, Onak K, Sidiropoulos A (2013) Fat polygonal partitions with applications to visualization and embeddings. J Comput Geom 4(1):212–239
Diehl S (2007) Software Visualization: visualizing the structure, behaviour, and evolution of software. Springer Science & Business Media
Domrös S, Lucas D, von Hanxleden R, Jansen K (2021) On order-preserving, gap-avoiding rectangle packing. In: Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 1: IVAPP,, INSTICC, SciTePress, IVAPP '21, pp 38–49, https://doi.org/10.5220/0010186400380049
Dübel S, Röhlig M, Schumann H, Trapp M (2014) 2D and 3D presentation of spatial data: a systematic review. In: Proceedings of the 2014 VIS International Workshop on 3DVis, IEEE, 3DVis '14, pp 11–18, https://doi.org/10.1109/3DVis.2014.7160094
Elmqvist N, Fekete JD (2010) Hierarchical aggregation for information visualization: overview, techniques, and design guidelines. Trans Vis Comput Graph 16(3):439–454. https://doi.org/10.1109/TVCG.2009.84
Fekete JD, Plaisant C (1999) Excentric labeling: dynamic neighborhood labeling for data visualization. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, CHI '99, pp 512–519, https://doi.org/10.1145/302979.303148
Feng C, Gong M, Deussen O, Huang H (2019) Treemapping via balanced partitioning. In: Proceedings of the Computational Visual Media Conference, CVM '19
Few S (2004) Tapping the power of visual perception. Perceptual Edge pp 1–8
Fiedler C, Scheibel W, Limberger D, Trapp M, Döllner J (2020) Survey on user studies on the effectiveness of treemaps. In: Proceedings of the 13th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '20, pp 2:1–10, https://doi.org/10.1145/3430036.3430054
Fügenschuh A, Junosza-Szaniawski K, Lonc Z (2014) Exact and approximation algorithms for a soft rectangle packing problem. Optimization 63(11):1637–1663. https://doi.org/10.1080/02331934.2012.728217
Giereth M, Bosch H, Ertl T (2008) A 3D treemap approach for analyzing the classificatory distribution in patent portfolios. In: Proceedings of the 2008 Symposium on Visual Analytics Science and Technology, IEEE, VAST '08, pp 189–190, https://doi.org/10.1109/VAST.2008.4677380
Görtler J, Schulz C, Weiskopf D, Deussen O (2017) Bubble treemaps for uncertainty visualization. Trans Vis Comput Graph 24(1):719–728. https://doi.org/10.1109/TVCG.2017.2743959
Gregg B (2016) The flame graph. Communications 59(6):48–57. https://doi.org/10.1145/2909476
Haber RB, McNabb DA (1990) Visualization idioms: a conceptual model for scientific visualization systems. Vis Sci Comput 74:93
Hagh-Shenas H, Kim S, Interrante V, Healey C (2007) Weaving versus blending: a quantitative assessment of the information carrying capacities of two alternative methods for conveying multivariate data with color. Trans Vis Comput Graph 13(6):1270–1277. https://doi.org/10.1109/TVCG.2007.70623
Hahn S, Döllner J (2017) Hybrid-treemap layouting. In: Proceedings of the Eurographics / IEEE VGTC European Conference on Visualization – Short Papers, EG, EuroVis '17, pp 79–83, https://doi.org/10.2312/eurovisshort.20171137
Hawes N, Marshall S, Anslow C (2015) CodeSurveyor: Mapping large-scale software to aid in code comprehension. In: Proceedings of the 3rd Working Conference on Software Visualization, IEEE, VISSOFT '15, pp 96–105, https://doi.org/10.1109/VISSOFT.2015.7332419
Hesse PN, Schmitt C, Klingenhoefer S, Bremmer F (2017) Preattentive processing of numerical visual information. Front Hum Neurosci 11:70. https://doi.org/10.3389/fnhum.2017.00070
Holten D (2006) Hierarchical edge bundles: visualization of adjacency relations in hierarchical data. Trans Vis Comput Graph 12(5):741–748. https://doi.org/10.1109/TVCG.2006.147
Holten D, Vliegen R, van Wijk JJ (2005) Visual realism for the visualization of software metrics. In: Proceedings of the 3rd International Workshop on Visualizing Software for Understanding and Analysis, IEEE, VISSOFT '05, pp 1–6, https://doi.org/10.1109/VISSOF.2005.1684299
Itoh T, Takakura H, Sawada A, Koyamada K (2006) Hierarchical visualization of network intrusion detection data. Comput Graph Appl 26(2):40–47. https://doi.org/10.1109/MCG.2006.34
Jadeja M, Muthu R (2017) Labeled object treemap: a new graph-labeling based technique for visualizing multiple hierarchies. Annals of Pure and Applied Mathematics 13:49–62, https://doi.org/10.22457/apam.v13n1a6

Jern M, Rogstadius J, Åström T (2009) Treemaps and choropleth maps applied to regional hierarchical statistical data. In: Proceedings of the 13th International Conference on Information Visualisation, IEEE, IV '09, pp 403–410, https://doi.org/10.1109/IV.2009.97

Johnson BS, Shneiderman B (1991) Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In: Proceedings of the 1991 Conference on Visualization, IEEE, VIS '91, pp 284–291, https://doi.org/10.1109/VISUAL.1991.175815

Keim DA, Andrienko G, Fekete JD, Gorg C, Kohlhammer J, Melançon G (2008) Visual analytics: definition, process, and challenges. Inf Vis Lect Notes Comput Sci 4950:154–176. https://doi.org/10.1007/978-3-540-70956-5_7

Kindlmann G, Scheidegger C (2014) An algebraic process for visualization design. Trans Vis Comput Graph 20(12):2181–2190. https://doi.org/10.1109/TVCG.2014.2346325

Knight C, Munro M (1999) Comprehension with[in] virtual environment visualisations. In: Proceedings oh the 7th International Workshop on Program Comprehension, IEEE, WPC '99, pp 4–11, https://doi.org/10.1109/WPC.1999.777733

Kokash N, de Bono B, Kok J (2014) Template-based treemaps to preserve spatial constraints. In: Proceedings of the 5th International Conference on Information Visualization Theory and Applications – Vol 1: IVAPP, SciTePress, IVAPP '14, pp 39–49, https://doi.org/10.5220/0004684900390049

Kong N, Heer J, Agrawala M (2010) Perceptual guidelines for creating rectangular treemaps. Trans Vis Comput Graph 16(6):990–998. https://doi.org/10.1109/TVCG.2010.186

Kuhn A, Loretan P, Nierstrasz O (2008) Consistent layout for thematic software maps. In: Proceedings of the 15th Working Conference on Reverse Engineering, IEEE, WCRE '08, pp 209–218, https://doi.org/10.1109/WCRE.2008.45

Langelier G, Sahraoui H, Poulin P (2005) Visualization-based analysis of quality for large-scale software systems. In: Proceedings of the 20th IEEE / ACM International Conference on Automated Software Engineering, ACM, ASE '05, pp 214–223, https://doi.org/10.1145/1101908.1101941

Liang J, Nguyen QV, Simoff S, Huang ML (2015) Divide and conquer treemaps: visualizing large trees with various shapes. J Vis Lang Comput 31:104–127. https://doi.org/10.1016/j.jvlc.2015.10.009

Limberger D, Wasty B, Trümper J, Döllner J (2013) Interactive software maps for web-based source code analysis. In: Proceedings of the 18th International Conference on 3D Web Technology, ACM, Web3D '13, pp 91–98, https://doi.org/10.1145/2466533.2466550

Limberger D, Fiedler C, Hahn S, Trapp M, Döllner J (2016a) Evaluation of sketchiness as a visual variable for 2.5D treemaps. In: Proceedings of the 20th International Conference on Information Visualisation, IEEE, IV '16, pp 183–189, https://doi.org/10.1109/IV.2016.61

Limberger D, Scheibel W, Lemme S, Döllner J (2016b) Dynamic 2.5D treemaps using declarative 3D on the web. In: Proceedings of the 21st International Conference on Web3D Technology, ACM, Web3D '16, pp 33–36, https://doi.org/10.1145/2945292.2945313

Limberger D, Pursche M, Klimke J, Döllner J (2017a) Progressive high-quality rendering for interactive information cartography using WebGL. In: Proceedings of the 22nd International Conference on 3D Web Technology, ACM, Web3D '17, pp 8:1–4, https://doi.org/10.1145/3055624.3075951

Limberger D, Scheibel W, Hahn S, Döllner J (2017b) Reducing visual complexity in software maps using importance-based aggregation of nodes. In: Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Vol 3: IVAPP, INSTICC, SciTePress, IVAPP '17, pp 176–185, https://doi.org/10.5220/0006267501760185

Limberger D, Scheibel W, Trapp M, Döllner J (2017c) Mixed-projection treemaps: A novel approach mixing 2D and 2.5D treemaps. In: Proceedings of the 21st International Conference on Information Visualisation, IEEE, IV '17, pp 164–169, https://doi.org/10.1109/iV.2017.67

Limberger D, Gropler A, Buschmann S, Döllner J, Wasty B (2018a) OpenLL: an API for dynamic 2D and 3D labeling. In: Proceedings of the 22nd International Conference on Information Visualisation, IEEE, IV '18, https://doi.org/10.1109/iV.2018.00039

Limberger D, Trapp M, Döllner J (2018b) Interactive, height-based filtering in 2.5D treemaps. In: Proceedings of the 11th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '18, pp 49–55, https://doi.org/10.1145/3231622.3231638

Limberger D, Scheibel W, Döllner J, Trapp M (2019a) Advanced visual metaphors and techniques for software maps. In: Proceedings of the 12th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '19, pp 1–8, https://doi.org/10.1145/3356422.3356444

Limberger D, Trapp M, Döllner J (2019b) In-situ comparison for 2.5D treemaps. In: Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP, SciTePress, IVAPP '19, https://doi.org/10.5220/0007576203140321

Limberger D, Trapp M, Döllner J (2020) Depicting uncertainty in 2.5D treemaps. In: Proceedings of the 13th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '20, pp 28:1–2, https://doi.org/10.1145/3430036.3432753

Limberger D, Scheibel W, van Dieken J, Döllner J (2021) Visualization of data changes in 2.5D treemaps using procedural textures and animated transitions. In: Proceedings of the 14th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '21, pp 6:1–5, https://doi.org/10.1145/3481549.3481570

Liu S, Cao N, Lv H (2008) Interactive visual analysis of the NSF funding information. In: Proceedings of the 2008 Pacific Visualization Symposium, IEEE, PACIFICVIS '08, pp 183–190, https://doi.org/10.1109/PACIFICVIS.2008.4475475

Liu S, Maljovec D, Wang B, Bremer P, Pascucci V (2017) Visualizing high-dimensional data: advances in the past decade. Trans Vis Comput Graph 23(3):1249–1268. https://doi.org/10.1109/TVCG.2016.2640960

Liu Z, Stasko J (2010) Mental models, visual reasoning and interaction in information visualization: a top-down perspective. Trans Vis Comput Graph 16(6):999–1008. https://doi.org/10.1109/TVCG.2010.177

Luboschik M, Schumann H (2008) Discovering the covered: ghost-views in information visualization. In: Proceedings of the 16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision, WSCG '08, pp 113–118, 11025/10927
Marcus A, Feng L, Maletic JI (2003) 3D representations for software visualization. In: Proceedings of the 2003 Symposium on Software Visualization, ACM, SoftVis '03, pp 27–37, https://doi.org/10.1145/774833.774837
Merino L, Ghafari M, Nierstrasz O (2016) Towards actionable visualisation in software development. In: Proceedings of the 2016 Working Conference on Software Visualization, IEEE, VISSOFT '16, pp 61–70, https://doi.org/10.1109/VISSOFT.2016.10
Merino L, Ghafari M, Anslow C, Nierstrasz O (2018) A systematic literature review of software visualization evaluation. J Syst Softw 144:165–180. https://doi.org/10.1016/j.jss.2018.06.027
Molli P, Skaf-Molli H, Bouthier C (2001) State treemap: an awareness widget for multi-synchronous groupware. In: Proceedings of the 7th International Workshop on Groupware, IEEE, CRIWG '01, pp 106–114, https://doi.org/10.1109/CRIWG.2001.951823
Nagamochi H, Abe Y (2007) An approximation algorithm for dissecting a rectangle into rectangles with specified areas. Discr Appl Math 155(4):523–537. https://doi.org/10.1016/j.dam.2006.08.005
Pfahler F, Minelli R, Nagy C, Lanza M (2020) Visualizing evolving software cities. In: Proceedings of the 2020 Working Conference on Software Visualization, VISSOFT '20, pp 22–26, https://doi.org/10.1109/VISSOFT51673.2020.00007
Quinan PS, Padilla L, Creem-Regehr SH, Meyer M (2019) Examining implicit discretization in spectral schemes. Comput Graph Forum 38(3):363–374. https://doi.org/10.1111/cgf.13695
Roberts RC, Laramee RS (2018) Visualising business data: a survey. Information 9(11):1–54. https://doi.org/10.3390/info9110285
dos Santos SR, Brodlie K (2004) Gaining understanding of multivariate and multidimensional data through visualization. Comput Graph 28:311–325. https://doi.org/10.1016/j.cag.2004.03.013
Scheibel W, Trapp M, Döllner J (2016) Interactive revision exploration using small multiples of software maps. In: Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 2: IVAPP, SciTePress, IVAPP '16, pp 131–138, https://doi.org/10.5220/0005694401310138
Scheibel W, Buschmann S, Trapp M, Döllner J (2017) Attributed vertex clouds. In: GPU Zen: Advanced Rendering Techniques, Bowker Identifier Services, chap Geometry Manipulation, pp 3–21
Scheibel W, Weyand C, Döllner J (2018) Evocells – a treemap layout algorithm for evolving tree data. In: Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 2: IVAPP, SciTePress, IVAPP '18, pp 273–280, https://doi.org/10.5220/0006617102730280
Scheibel W, Hartmann J, Limberger D, Döllner J (2020a) Visualization of tree-structured data using web service composition. VISIGRAPP 2019: Computer Vision, Imaging and Computer Graphics Theory and Applications pp 227–252, https://doi.org/10.1007/978-3-030-41590-7_10
Scheibel W, Limberger D, Döllner J (2020b) Survey of treemap layout algorithms. In: Proceedings of the 13th International Symposium on Visual Information Communication and Interaction, ACM, VINCI '20, pp 1:1–9, https://doi.org/10.1145/3430036.3430041
Scheibel W, Trapp M, Limberger D, Döllner J (2020c) A taxonomy of treemap visualization techniques. In: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications – Volume 3: IVAPP, INSTICC, SciTePress, IVAPP '20, pp 273–280, https://doi.org/10.5220/0009153902730280
Scheibel W, Weyand C, Bethge J, Döllner J (2021) Algorithmic improvements on Hilbert and Moore treemaps for visualization of large tree-structured datasets. In: Proceedings of the 23rd Eurographics / IEEE VGTC European Conference on Visualization – Short Papers, EG, EuroVis '21, pp 115–119, https://doi.org/10.2312/evs.20211065
Schlechtweg S, Schulze-Wollgast P, Schumann H (2004) Interactive treemaps with detail on demand to support information search in documents. In: Proceedings of the 6th Joint Eurographics / IEEE VGTC Symposium on Visualization, EG, VISSYM'04, pp 121–128, https://doi.org/10.2312/VisSym/VisSym04/121-128
Schloss KB, Gramazio CC, Silverman AT, Parker ML, Wang AS (2018) Mapping color to meaning in colormap data visualizations. Trans Vis Comput Graph 25(1):810–819. https://doi.org/10.1109/TVCG.2018.2865147
Schulz HJ, Hadlak S, Schumann H (2011) The design space of implicit hierarchy visualization: a survey. Trans Vis Comput Graph 17(4):393–411. https://doi.org/10.1109/TVCG.2010.79
Shah S, Mitchell W, Shook D (2005) Challenges in the detection, diagnosis and visualization controller performance data. In: Proceedings of the 2005 Seminar on Control Loop Assessment and Diagnosis (Ref. No. 2005/11008), IEE, pp 7–21, https://doi.org/10.1049/ic:20050170
Shneiderman B (2009) Treemaps for space-constrained visualization of hierarchies. Tech Rep, Hum Comput Interact Lab http://www.cs.umd.edu/hcil/treemap-history
Shneiderman B, Wattenberg M (2001) Ordered treemap layouts. In: Proceedings of the 2001 Symposium on Information Visualization, IEEE, INFOVIS '01, pp 73–78, https://doi.org/10.1109/INFVIS.2001.963283
Slingsby A, Dykes J, Wood J (2008) Using treemaps for variable selection in spatio-temporal visualisation. Inf Vis 7(3–4):210–224. https://doi.org/10.1057/PALGRAVE.IVS.9500185
Soares AGM, Miranda ETC, Lima RSdAD, Resque dos Santos CG, Meiguins BS (2020) Depicting more information in enriched squarified treemaps with layered glyphs. Information 11(2):1–21. https://doi.org/10.3390/info11020123
Sondag M, Speckmann B, Verbeek K (2018) Stable treemaps via local moves. Trans Vis Comput Graph 24(1):729–738. https://doi.org/10.1109/TVCG.2017.2745140
Sondag M, Meulemans W, Schulz C, Verbeek K, Weiskopf D, Speckmann B (2020) Uncertainty treemaps. In: Proceedings of the 2020 Pacific Visualization Symposium, IEEE, PacificVis '20, pp 111–120, https://doi.org/10.1109/PacificVis48177.2020.7614
Stasko J, Catrambone R, Guzdial M, McDonald K (2000) An evaluation of space-filling information visualizations for depicting hierarchical structures. Int J Hum Comput Stud 53(5):663–694. https://doi.org/10.1006/ijhc.2000.0420

Steinbeck M, Koschke R, Rüdel MO (2019) Comparing the EvoStreets visualization technique in two- and three-dimensional environments: a controlled experiment. In: Proceedings of the 27th IEEE / ACM International Conference on Program Comprehension, IEEE, ICPC '19, pp 231–242, https://doi.org/10.1109/icpc.2019.00042
Steinbrückner F, Lewerentz C (2013) Understanding software evolution with software cities. Inf Vis 12(2):200–216. https://doi.org/10.1177/1473871612438785
Tak S, Cockburn A (2013) Enhanced spatial stability with Hilbert and Moore treemaps. Trans Vis Comput Graph 19(1):141–148. https://doi.org/10.1109/TVCG.2012.108
Thakur S, Rhyne TM (2009) Data vases: 2D and 3D plots for visualizing multiple time series. In: Advances in Visual Computing, Springer, ISVC '09, pp 929–938, https://doi.org/10.1007/978-3-642-10520-3_89
Trapp M, Schmechel S, Döllner J (2013) Interactive rendering of complex 3D-treemaps with a comparative performance evaluations. In: Proceedings of the International Conference on Computer Graphics Theory and Applications and International Conference on Information Visualization Theory and Applications – Volume 1: GRAPP, SciTePress, GRAPP '13, pp 165–175, https://doi.org/10.5220/0004290101650175
Tu Y, Shen HW (2007) Visualizing changes of hierarchical data using treemaps. Trans Vis Comput Graph 13(6):1286–1293. https://doi.org/10.1109/TVCG.2007.70529
Turo D (1994) Hierarchical visualization with treemaps: making sense of pro basketball data. In: Conference Companion on Human Factors in Computing Systems, ACM, CHI '94, pp 441–442, https://doi.org/10.1145/259963.260441
Turo D, Johnson BS (1992) Improving the visualization of hierarchies with treemaps: design issues and experimentation. In: Proceedings of the 1992 Conference on Visualization, IEEE, VIS '92, pp 124–131, https://doi.org/10.1109/VISUAL.1992.235217
Vaidya RK, De Carli L, Davidson D, Rastogi V (2019) Security issues in language-based software ecosystems. https://doi.org/10.48550/ARXIV.1903.02613, https://arxiv.org/abs/1903.02613
van Wijk JJ (2005) The value of visualization. In: Proceedings of the 2005 Conference on Visualization, IEEE, VIS '05, pp 79–86, https://doi.org/10.1109/VISUAL.2005.1532781
van Wijk JJ, van de Wetering H (1999) Cushion treemaps: visualization of hierarchical information. In: Proceedings of the 1999 Symposium on Information Visualization, IEEE, InfoVis '99, pp 73–78, https://doi.org/10.1109/INFVIS.1999.801860
Veras R, Collins C (2017) Optimizing hierarchical visualizations with the minimum description length principle. Trans Vis Comput Graph 23(1):631–640. https://doi.org/10.1109/TVCG.2016.2598591
Vernier EF, Comba J, Telea AC (2018) A stable greedy insertion treemap algorithm for software evolution visualization. In: Proceedings of the 31st Conference on Graphics, Patterns and Images, IEEE, SIBGRAPI '18, pp 158–165, https://doi.org/10.1109/SIBGRAPI.2018.00027
Vernier EF, Sondag M, Comba J, Speckmann B, Telea A, Verbeek K (2020) Quantitative comparison of time-dependent treemaps. Comput Graph Forum 39(3):393–404. https://doi.org/10.1111/cgf.13989
Vliegen R, van Wijk JJ, van der Linden EJ (2006) Visualizing business data with generalized treemaps. Trans Vis Comput Graph 12(5):789–796. https://doi.org/10.1109/TVCG.2006.200
Wang YC, Xing Y, Lin F, Seah HS, Zhang J (2022) OST: a heuristic-based orthogonal partitioning algorithm for dynamic hierarchical data visualization. J Vis. https://doi.org/10.1007/s12650-022-00830-1
Ware C (2012) Information visualization: perception for design. Elsevier
Wattenberg M (1999) Visualizing the stock market. In: Proceedings of the 1999 SIGCHI Conference on Human Factors in Computing Systems – Extended Abstracts, ACM, CHI '99, pp 188–189, https://doi.org/10.1145/632716.632834
Wettel R, Lanza M (2007) Program comprehension through software habitability. In: Proceedings of the 15th International Conference on Program Comprehension, IEEE, ICPC '07, pp 231–240, https://doi.org/10.1109/ICPC.2007.30
Wettel R, Lanza M (2008) Visual exploration of large-scale system evolution. In: Proceedings of the 15th Working Conference on Reverse Engineering, IEEE, WCRE '08, pp 219–228, https://doi.org/10.1109/WCRE.2008.55
Würfel H, Trapp M, Limberger D, Döllner J (2015) Natural phenomena as metaphors for visualization of trend data in interactive software maps. In: Proceedings of the 2015 Conference on Computer Graphics and Visual Computing, EG, CGVC '15, https://doi.org/10.2312/cgvc.20151246
Xie Y, Lin T, Chen R, Chen Z (2018) Toward improved aesthetics and data discrimination for treemaps via color schemes. Color Res Appl 43(3):328–340. https://doi.org/10.1002/col.22196
Yamaguchi Y, Itoh T (2003) Visualization of distributed processes using "Data Jewelry Box" algorithm. In: Proceedings Computer Graphics International 2003, IEEE, CGI '03, pp 162–169, https://doi.org/10.1109/CGI.2003.1214461
Yang Y, Zhang K, Wang J, Nguyen QV (2015) Cabinet tree: an orthogonal enclosure approach to visualizing and exploring big data. J Big Data 2(15):1–18. https://doi.org/10.1186/s40537-015-0022-3
Zhang J (2008) The implication of metaphors in information visualization. Vis Inf Retr 23:215–237. https://doi.org/10.1007/978-3-540-75148-9_10