Advanced Visual Metaphors and Techniques for Software Maps

Daniel Limberger
Hasso Plattner Institute, Faculty of Digital Engineering,
University of Potsdam, Germany
daniel.limberger@hpi.de

Willy Scheibel
Hasso Plattner Institute, Faculty of Digital Engineering,
University of Potsdam, Germany
willy.scheibel@hpi.de

Jürgen Döllner
Hasso Plattner Institute, Faculty of Digital Engineering,
University of Potsdam, Germany
juergen.doellner@hpi.de

Matthias Trapp
Hasso Plattner Institute, Faculty of Digital Engineering,
University of Potsdam, Germany
matthias.trapp@hpi.de

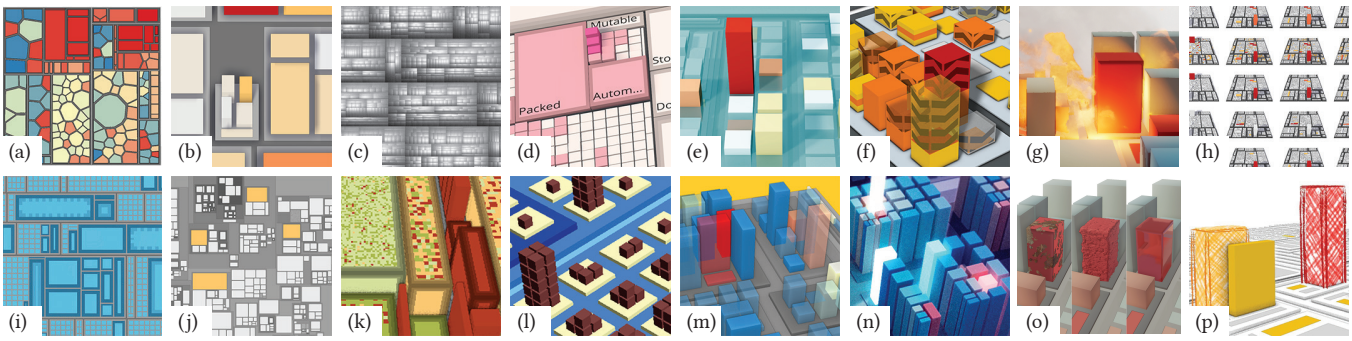

Figure 1: A compilation of advanced software maps: (a) mixed use of rectangular and voronoi layout, (b) mixed-projection to combine 2D and 2.5D depictions, (c) structure-enhancing cushion shading, (d) visual aggregation for a higher-level view on the data, (e) a reference surface for improved height comparison and visual filtering, (f) in-situ comparison for different points in time, (g) using the fire metaphor to augment interpretability, (h) small multiples to compare multiple points in time and map themes, (i) different contour types to convey information on aggregated data, (j) using growing and shrinking to derive subsequent layouts, (k) color weaving to encode variation in underlying data, (l) subdivision on leaf nodes, (m) transparency as visual variable, (n) highlighting of nodes using glow, (o) rustiness, shininess, and (p) sketchiness as visual variable.

ABSTRACT

Software maps provide a general-purpose interactive user interface and information display for software analytics tools. This paper systematically introduces and classifies software maps as a treemap-based technique for software cartography. It provides an overview of advanced visual metaphors and techniques, each suitable for interactive visual analytics tasks, that can be used to enhance the expressiveness of software maps. Thereto, the metaphors and techniques are briefly described, located within a visualization pipeline model, and considered within the software map design space. Consequent applications and use cases w.r.t. different types of software system data and software engineering data are discussed, arguing for a versatile use of software maps in visual software analytics.

CCS CONCEPTS

• **Human-centered computing** → **Treemaps**; *Information visualization*; *Visualization design and evaluation methods.*

KEYWORDS

treemap, design space, information visualization

ACM Reference Format:
Daniel Limberger, Willy Scheibel, Jürgen Döllner, and Matthias Trapp. 2019. Advanced Visual Metaphors and Techniques for Software Maps. In *The 12th International Symposium on Visual Information Communication and Interaction (VINCI'2019), September 20–22, 2019, Shanghai, China.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3356422.3356444

1 INTRODUCTION

Treemaps are a well established information visualization technique that implements the information seeking mantra [54] and is used in different domains [55]. They allow for a depiction of non-spatial, multi-variate data and provide the convenience and expressiveness of thematic maps, i.e., given a spatialization strategy (layouting), they support a combination of two or more visual variables for information display, such as color, size, or texture [10]. The 2D depiction is often extended into the third dimension, resulting in so called 2.5D treemaps, allowing for additional information display [62].

VINCI'2019, September 20–22, 2019, Shanghai, China
2019. ACM ISBN 978-1-4503-7626-6/19/09…$15.00
https://doi.org/10.1145/3356422.3356444

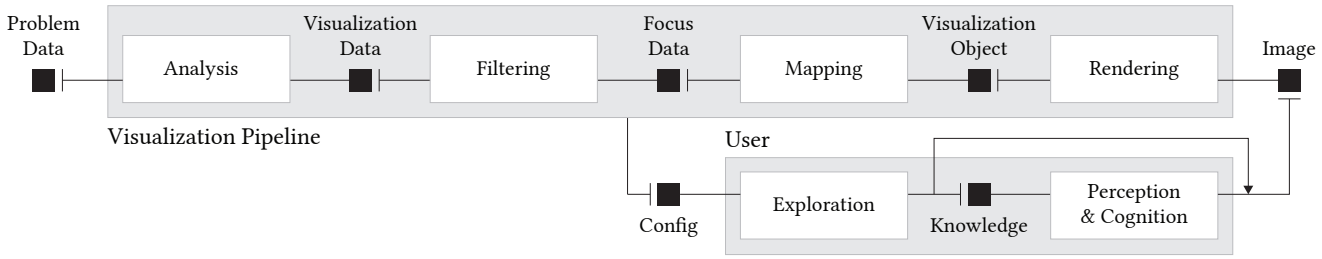

Figure 2: Model of the visualization process [63] consisting of a visualization pipeline [17] and a feedback loop for knowledge gathering and interactive user control. This model is applicable to visual software analytics using software maps.

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" [30].

In the domain of software analytics, treemaps are used to depict software system data and software engineering data [8, 66], resulting in so called *software maps*. Software maps provide a unique and general-purpose interactive user interface and information display for software analytics tools. Typically, a catalogue of *software map themes* compiles commonly used attribute selections related to specific tasks in software analytics. A software map theme, hereinafter referred to as *map theme*, defines a selection of software information dimensions that are mapped onto visual variables of a software map. It portrays selected aspects of the software information gathered and analyzed by software analytics processes. A map theme presents, in a sense, topic-specific software map templates that support different stakeholders in software engineering in data-driven decision making as well as in finding actionable insights.

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" [30], we derive the following challenges for software maps:

- What are best practices and variations to software maps?
- How does the type of data impacts its visualization?
- Can techniques be used individually or in combination?
- In what ways can data be depicted beyond the basic use of the visual variables position, area, height, and color?

To provide a basis for answering these questions, we discuss advanced visual metaphors and techniques for software maps by exploring the visualization pipeline of software maps. A superimposed visualization process integrates a visualization pipeline into a feedback loop with interactive user control of the various stages [63]. Such visualization pipeline is basically 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 only a few variations of models of visualization pipelines: 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 [9, 13, 14]. An extensive overview of specific transformations suitable for visualization purposes was recently provided by Liu *et al.* [43]. A similar naming schemes uses (1) data enrichment/enhancement, (2) visualization mapping, and (3) rendering to describe the same process [23]. This scheme was later refined to (1) data analysis, (2) filtering, (3) mapping, and (4) rendering [17]. For our purposes, we use the latter model and refer to the first two stages as *preprocessing and filtering*. *Preprocessing and filtering* covers, among others, analytics, selection, enrichment, transformation, and resampling of raw data. *Mapping* maps abstract data to visual representations. *Rendering* creates a visual representation using image synthesis (cf. Figure 2).

Based on this approach, we provide an overview of advanced visual metaphors that have already been or can be used in combination with software maps to approach above mentioned challenges (cf. Figure 1). For it, we briefly summarize and discuss each metaphor and technique within the context of a stage with the conceptual model of the visualization pipeline and describe common practices based on our experiences. We discuss the extent to which each metaphor or technique can be used to communicate different types of data and, furthermore, identify feasible combinations of isolated techniques and possible interferences, spanning as well as extending a design space of software maps.

2 DESIGN SPACE OF SOFTWARE MAPS

In software cartography, the term software maps is not uniquely defined. Each variation (such as software cities [66], code cities [67], cityscapes, or thematic software maps [33]) is focusing 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" [33]. In order to obtain a more precise, low-level definition of *software maps*, we propose a definition that precisely denotes them as a subset of treemaps using existing systematization approaches.

Schulz *et al.* [53] identified the following axis 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.* [18] further 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.

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 by means of nesting/inclusion. Adjacency in 2.5D maps can also be used, but we suggest to consider its use carefully; eventually we use flat representations of inner nodes only for all map themes in order to allow for special representation of aggregates. Techniques that are based on containment and overlap in 3D are only marginally useful for medium to large data sets due to added interaction complexity and hard-to-resolve occlusion. Fortunately, this restriction does not exclude any of the commonly used treemap layouting algorithms based on rectangular or Voronoi shapes [25].

For the representation of leaf nodes, 2D graphics primitives are common, e.g., Voronoi shapes or rectangles ($\mathcal{A}^2$). They further can be extruded, optionally tapered, and thereby exploiting the third spatial dimension ($\mathcal{A}^3$). "Increasing the visual vocabulary can provide for richer information resolution" [62] and allows for additional information display. For these so called 2.5D software maps, we generally prefer graphical primitives of rectangular footprints over more complex ones (cf. Figure 3) – 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, and performance. Since the reference space is still in $\mathcal{R}^2$, we support the term 2.5D [62] to avoid confusion with actual 3D treemaps that utilize a 3D layouting for positioning graphical elements in a 3D reference space ($\mathcal{R}^3$).

With the attribute and reference space 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 treemaps with the purposeful depictions of abstract software-system data and software engineering data, as well as software development data (cf. Figure 4). This has one outstanding benefit: their characteristics largely match these of ubiquitously available, interactive 2D and 2.5D (geo-spatial) maps. With this definition of software maps, both visualization engineers and software engineers can rely on and build upon well established (1) interaction metaphors, (2) provisioning strategies, and (3) visualization metaphors, thereby harnessing on the habits and experiences of most users.

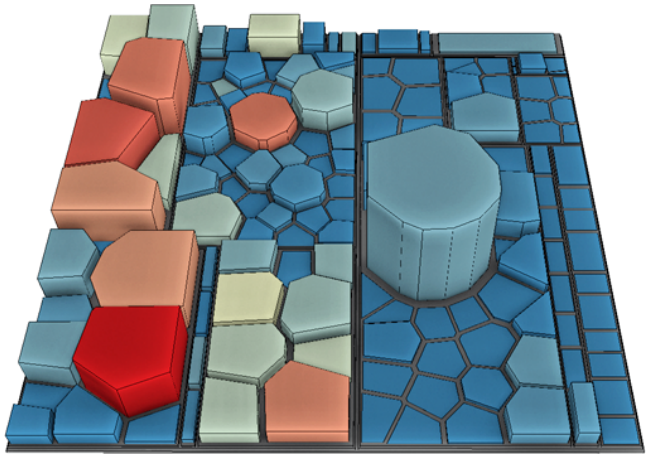

Figure 3: An example of a 2.5D software map with mixed use of rectangular and Voronoi treemap layouts.

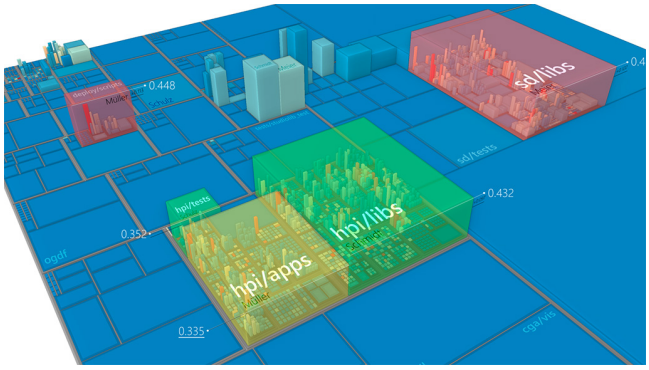

Figure 4: An example of a semi-transparent information overlay to superimpose additional information.

2.1 Data Characteristics

Traditionally, visualizations targeting software analytics are required to handle three major aspects of software system data: static, dynamic, and evolving aspects [16]. We use a broader scope for software maps by including software development data as well, which basically includes all stakeholders of the software development process. The data typically covers mined, preprocessed data gathered from various sources:

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

Even if measured exactly, the data, is (1) prone to uncertainty due to missing semantic normalization, (2) lacks standardized metrics and interpretations, (3) is highly language specific and, thus, highly heterogeneous in large software system. Further, anonymization may be required, the measurement resolution is unnecessarily high, and finally, the data is massive for medium to large sized engineering processes (metrics per file and per commit, plus all changes w.r.t. various issue tracking systems and continuous integration systems). All 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 a survey by Caserta and Zendra [11].

2.2 Preliminaries and Assumptions

This paper does not provide a comprehensive set of common design guidelines for creating rectangular treemaps. Guidelines concerned with perception [32] and use of color [47, 52, 69] are outside the scope of this paper, since these are of complex nature itself and still subject to active research. The selection of presented visualization techniques focuses on visualization metaphors and techniques that cope with the data characteristics within the domain of software engineering, i.e., volume and complexity of the data. Thus, we selected visualization techniques that were used for software data and where an implementation can handle datasets up to hundreds of thousands of data items. Contrarily, this paper does not cover city or city-like metaphors [3, 31] that map abstract data to features of virtual 3D city models such as building facades, car or pedestrian traffic, or street furniture. Regarding frameworks and implementations of software maps, we refer to state-of-the-art approaches based on hardware acceleration and highly dynamic mappings [48, 60]. Similarly, the deployment and provisioning of data and visualization is highly contextual and application-specific [38, 42]. Fortunately, a simplifying view on the application scenarios allows to derive common characteristics and shared approaches for visualization [49].

3 PREPROCESSING AND FILTERING

During preprocessing and filtering, the incoming data – also known as problem or task data – is transformed and prepared for visualization. This includes operations such as resampling, normalization, filtering of outlier, accumulating weighted leaf-node data to inner nodes, and more. From a framework design perspective, we tend to leave any input data unmodified and let any transformation result in additional tree-structures and data views.

Attribute Resolution. While creating software maps in practice, we encountered several challenges. For example, the data is usually available in much higher resolution than required for visualization. In those cases, we suggest to reduce the data to a resolution that is reasonable for the map theme, e.g., attributes mapped to area, color, or height do not require a 32 bit floating-point resolution. Often, reducing the attribute resolution to a few bits can increase readability by means of discriminability and emphasize results of preceeding 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 and the exact size of a module within a treemap should be of subordinate relevance. Thus, often a transformation to a categorical data type as *irrelevant, low, medium, and high* (for color, height, or change) or *lower-outlier, below average, average, above average, and upper-outlier* (for area) seem to be comparably or even more effective.

Streaming and Level-of-Detail. The major challenge, however, is that industry software projects tend to be massive (in terms of number of modules, code units, metrics, or activities). This is often ignored or only marginally covered in research but has significant impact on visualization design. Software maps might be streamed on demand (node by node, slice by slice, etc.). Loading millions of attribute values for nodes that should not even be depicted (e.g., as they are out of scope or of sub-pixel size) can decrease responsiveness and result in visual clutter or memory shortages.

Figure 5: Stepwise generalization of a software map by use of aggregation, ranging from no aggregation (left) to strong aggregation (right). Outlier nodes – also called landmarks – are preserved [37].

It is beneficial to have a highly customizable, interactive level-of-detail [19, 37] (filtering and rendering) that allows for dynamic detail (cf. Figure 5), e.g., detecting source code duplicates, generated code, automated activities, and filter irrelevant areas such as third party code. Furthermore, the software map should always support the concept of mental maps, i.e., the overall landscape/spatial layout should not change due to a dynamic level-of-detail. A particular technique to combine aggregated and detailed views in combination with reduced occlusion is the *lifted map* [12].

4 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 [65]. 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 [48, 60]).

Software maps usually use inner nodes to depict applications, modules, or even source code units, i.e., source code files [8]. The leave nodes depict either modules, source code units, functions, events, developers, or activity (by means of commits). The position of each node is computed by a layout technique specified by the map theme, and a metric is usually mapped to the size of the graphical primitive, e.g., lines-of-code, file size, or size of domain logic.

Use of Height. We utilize height of cuboids by means of an extrusion of the 2D shape [6, 15] as secondary visual variable. The idea is to provide the following order with declining importance for the task: (1) color, (2) height, (3) other visual variables. For 2.5D software maps, pyramid-like shapes can be used to further encode an attribute or just reduce occlusion [62]. When depicting evolving data with accustomed map themes, height seems to allow for an intuitive encoding of data changes by means of growing/increasing vs. shrinking/decreasing. Height should not be used, however, to depict negative or diverging scales directly, as this would result in

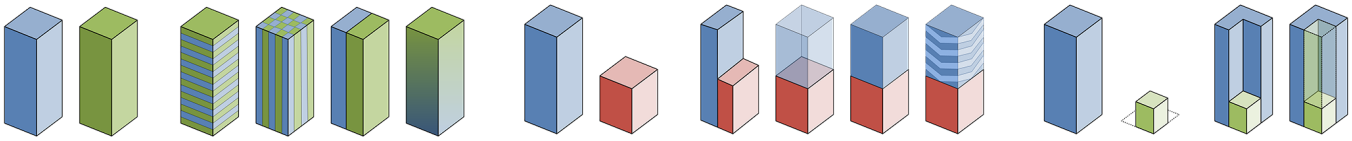

Figure 6: A set of in-situ templates to depict changes in underlying data for up to three attributes using up to three visual variables [41]. Left group contains templates for color-only changes. Middle group contains templates for color and height changes. Right group presents templates for changes in all area, height, and color.

downwards facing cuboids. Even though this was suggested several times, we prefer to refine the map theme to transform or map relevant value ranges of attributes to make a salient but expected use of visual variables (cuboids are high if data is of interest). In contrast thereto, if a negative value range is relevant, it is mapped inversely to height. If positive values are relevant as well, 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 [34] and the type – by means of poly cylinders [45] and three-dimensional glyphs [7] are further suitable as visual variables.

Evolutionary Data. One of the biggest challenges for treemaps – and thus, software maps – is the handling of topology changes, i.e., addition, move, and removal of nodes, in a predictable and comprehensible manner. If the time range of interest is in the past, a multi-revision hierarchy can be computed [61]. The quality of this hierarchy heavily depends on the computations capabilities to track nodes over multiple revisions (renames, copies, clones, splits, moves, deletions, etc.). If the software map is used to visualize ongoing processes and continuously evolving data, an initial layout can be incrementally evolved based on the topology changes, resulting in nodes growing, shrinking, appearing, and disappearing [51].

Juxtaposing and Complex Shapes. For visual display of multiple states or sub-elements, data vases, stacked cuboids, or fragmenting can be used. In remembrance of stacked bar charts, the extruded polytopes can be subdivided in height, allowing for depiction of subcategories and their share on the overall height [21, 29]. This process can further be utilized, e.g., to encode evolution by means of *evolution segments* [58] and *data vases* [59]. Small multiples can be used for "the comparison of multiple software map themes and revisions simultaneously on a single screen" [50]. We found this especially useful for generation of an overview on as well as exploration and identification of map theme variations.

Depicting Relations. For an emphasis of the topology w.r.t. to the nested structure of nodes, cushion shading [64], variations of margins or padding, as well extruded, stacked inner nodes [6] can be applied. If additional relations 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 [27, 57]. This approach, however, is visually constrained by the number of depicted relations and data set size. Superimposing relations by means of edges or tubes on top of 2.5D treemaps introduces additional clutter, occlusion, and visual complexity that is not inherent to the treemap metaphor.

5 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 usually 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 resulting image by use of visual variables, some of which are mapped by means of geometry, e.g., position or extent.

Depicting Change. Many software map themes are designed to facilitate understanding of the evolution of the underlying data sets. 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 [61] 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. It was conceptually extended by in-situ templates [41], which allow for visual encoding of former and latter states on a per-cuboid basis for rectangular software maps (cf. Figure 6). Specifically, basic and more complex templates were introduced in order to depict changes in more than one visual variable simultaneously, including area, color, and height. Although not discussed in the paper, the templates allow for comparison of additional per-node visual variables as well. The visual quality and expressiveness also depends on the available rendering techniques, e.g., procedural texturing and transparency are accounted for by several templates.

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

Utilizing Textures. Different texture patterns can be applied to encode categorical data [46]. 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 visual variable for scales with a natural zero [28]). With most modern rendering engines supporting some form of physically-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

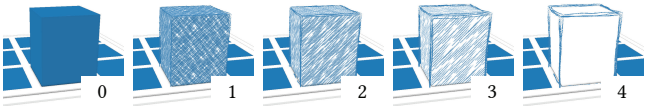

Figure 7: A five-step scale to encode "uncertainty, imprecision or vagueness" [35] using sketchiness as visual variable. The scale ranges from no uncertainty (left) to a high degree of uncertainty (right) with varying degrees of a sketch-rendering effect.

procedural textures to depict quality measures (e.g., degree of documentation, error-prone-ness), complexity measures (incoming or outgoing dependencies, mixed use of third-party dependencies), and development process indicators (e.g., number of authors).

Utilizing Transparency. Regarding geometric processing of inner nodes, there are approaches that extrude the layout as well, creating platforms [1] or pyramid frustums [2], placing child nodes *on top* of the platforms. Instead of stacking, there are approaches for nested depiction as well, e.g., full spheres [5] and hemispheres [4] in combination with transparency. Other uses of transparency [44] are (1) to reduce occlusion and (2) as a visual variable to encode different node states. Transparency can further be used to depict removed or planned components, goals, irrelevant nodes as of the current map theme, or enhance the expressiveness and quality of (procedural) texturing. However, the most relevant obstacle for using transparency is the complexity of its implementation: 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 in a rendering framework based on multi-frame sampling [36].

Depicting Uncertainty. 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 [22] and used to indicate vagueness. Pencil-like outlines and hatching of surfaces can be combined into a single visual variable, i.e., *sketchiness* (cf. Figure 7). Although technically an application of texture as a visual variable, mapping data to sketchiness is especially useful to encode "uncertainty, imprecision, or vagueness" [35]. Sketchiness is designed to encode varying degrees of uncertainty and, furthermore, can be used in addition to color. For software maps, it can display measurement inaccuracies (incomplete or estimated data) or target data that is yet to get discussed or implemented (i.e., targets for complexity, coverage, performance, or other quality measure).

Depicting Aggregates. As previously discussed for the preprocessing and filtering stages, aggregation is an essential technique for software maps (cf. Figure 8). In the context of rendering, it is important to adhere to aggregation guidelines [19]. 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 [56]. Likewise, noise or color weaving [24] as well as

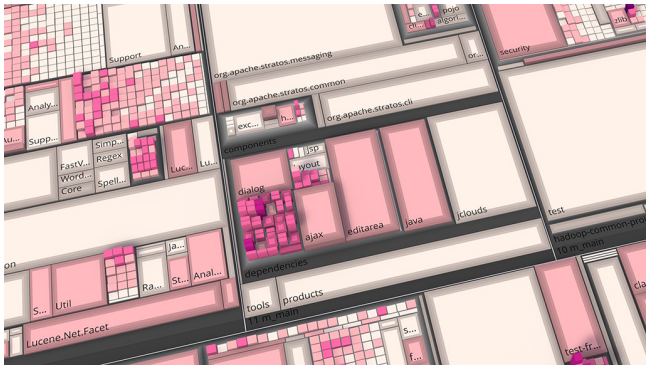

Figure 8: An example of software map aggregation and internal labeling [37]. The resulting space from aggregating nodes is used to embed labels into the software map.

nesting-level contours (multiple contours that hint the depth of the aggregated sub-tree, i.e., the number of an aggregate's subjacent hierarchy levels) can be used as well [37]. For software maps, the aggregation should summarize contextual data of low variation to provide orientation without distraction. Aggregation can further extend user control and interaction, e.g., by means of folding and unfolding, as well as for dynamically scaling the rendering load in order to comply to performance constraints.

Height-based Selection and Thresholding. For 2.5D software maps, a reference surface – a height reference – can be used to "facilitate accurate identification of highest nodes as well as similar nodes [...]. 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." [40]. Depending on the capabilities of the renderer and the already occupied visual variables, one of various approaches for the visual display of the height reference can be applied, e.g., intersection, stilts, explicit surface, closed surface, and implicit surface (cf. Figure 9). Using user-controlled 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.

Using Natural Metaphors. The use of natural metaphors is 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" [70]. To this end, natural metaphors, such as physically-based materials (e.g., rust, radiant emittance, roughness, and shininess; cf. Figure 10) or weather phenomena (e.g., rain, clouds, fire, dust, snow, etc.) were suggested to "emotionalize the visual communication by providing memorable visualizations" [68]. 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 [20, 26]. However, the use of weather phenomena requires 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.

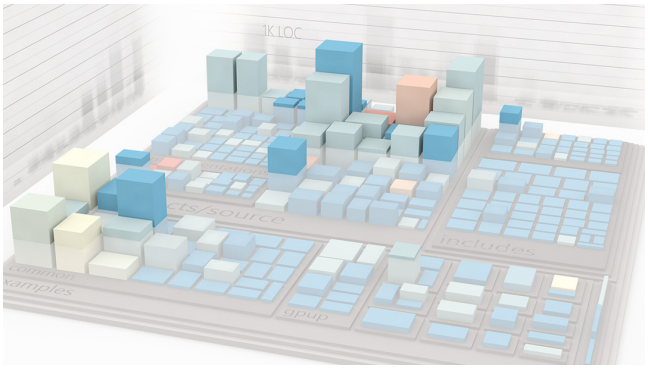

Figure 9: An example combining visual filtering using a reference surface and adjacent charts that display per-axis projected height information [40]. The nodes that are below the reference surface are visually filtered by means of a semi-transparent surface metaphor.

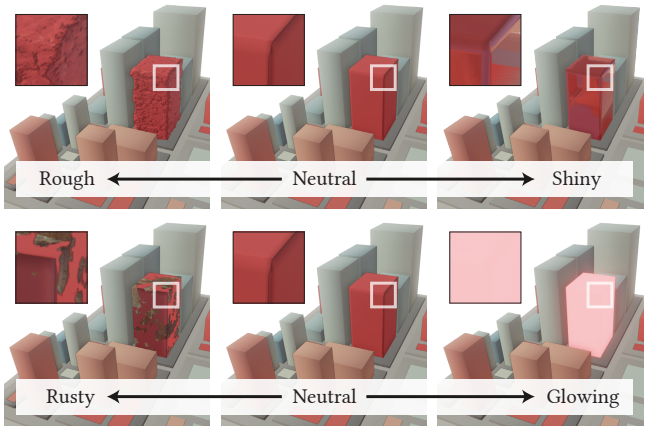

Figure 10: An example of (top) rough-to-shiny and (bottom) rusty-to-glowing visualization metaphors to be used as visual variables. These metaphors are capable to communicate deterioration or negative deviation (roughness, rust) and improvement or positive deviation (shine, glow) without need for further introduction of the mapping semantics [68].

Mixed Projections. Recently, an approach for focus-sensitive use 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" [39]. This allows for on-demand separation of high-detail focus areas and context areas. It further allows to reduce occlusion issues – which might occur for a certain data set or map theme – and complexity of navigation (3D to 2D). For software analytics, mixed projection can facilitate communication of the source code modules that are target to change for the current sprint. In this example, the context are the other source code files present in the project. Similarly, mixed projection can be of use 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.

6 CONCLUSIONS AND FUTURE WORK

This paper reviewed advanced visual metaphors and visualization techniques for software maps. A brief discussion for 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. Thereby, the approaches can visually scale from small to large and even massive data sets. Summarizing, we argue that the software map is a highly versatile tool in visual software analytics.

For future work, we evaluate the applicability of the advanced metaphors and techniques to other 2D and 2.5D information visualization techniques. In order to broadly provide the software map and its extensions, we plan to provide an open software ecosystem.

ACKNOWLEDGMENTS

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the KMUi project "BIMAP" and the German Federal Ministry for Economic Affairs and Energy (BMWi) within the ZIM project "Twin4Bim – TASAM".

REFERENCES

[1] Keith Andrews. 1995. Case study. Visualising cyberspace: information visualisation in the Harmony Internet browser. In *Proc. IEEE Vis.* 97–104. https://doi.org/10.1109/INFVIS.1995.528692
[2] Keith Andrews. 2002. Visual Exploration of Large Hierarchies with Information Pyramids. In *Proc. IEEE iV.* 793. https://doi.org/10.1109/IV.2002.1028871
[3] G. Balogh, A. Szabolics, and Á. Beszédes. 2015. CodeMetropolis: Eclipse over the city of source code. In *Proc. IEEE SCAM.* 271–276. https://doi.org/10.1109/SCAM.2015.7335425
[4] Michael Balzer and Oliver Deussen. 2004. Hierarchy Based 3D Visualization of Large Software Structures. In *Proc. IEEE VIS (VIS '04).* IEEE Computer Society, Washington, DC, USA. https://doi.org/10.1109/VISUAL.2004.39
[5] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. 2004. Software Landscapes: Visualizing the Structure of Large Software Systems. In *Proc. EG/IEEE VisSym*, Oliver Deussen, Charles Hansen, Daniel Keim, and Dietmar Saupe (Eds.). The Eurographics Association. https://doi.org/10.2312/VisSym/VisSym04/261-266
[6] Thomas Bladh, David A Carr, and Jeremiah Scholl. 2004. Extending tree-maps to three dimensions: A comparative study. In *Proc. APCHI.* Springer, 50–59.
[7] S. Boccuzzo and H. Gall. 2007. CocoViz: Towards Cognitive Software Visualizations. In *Proc. IEEE VISSOFT.* 72–79. https://doi.org/10.1109/VISSOF.2007.4290703
[8] Johannes Bohnet and Jürgen Döllner. 2011. Monitoring Code Quality and Development Activity by Software Maps. In *Proc. ACM MTD (MTD '11).* ACM, 9–16. https://doi.org/10.1145/1985362.1985365
[9] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman (Eds.). 1999. *Readings in Information Visualization: Using Vision to Think.* Morgan Kaufmann Publishers.
[10] M. Sheelagh T. Carpendale. 2003. *Considering visual variables as a basis for information visualisation.* Technical Report. University of Calgary.
[11] P. Caserta and O. Zendra. 2011. Visualization of the Static Aspects of Software: A Survey. *IEEE TVCG* 17, 7 (2011), 913–933. https://doi.org/10.1109/TVCG.2010.110
[12] Abon Chaudhuri and Han-Wei Shen. 2009. A self-adaptive treemap-based technique for visualizing hierarchical data in 3D. In *Proc. IEEE PacificVis.* IEEE Computer Society, 105–112. https://doi.org/10.1109/PACIFICVIS.2009.4906844
[13] Ed H. Chi. 2000. A Taxonomy of Visualization Techniques Using the Data State Reference Model. In *Proc. IEEE InfoVis (INFOVIS '00).* IEEE Computer Society.
[14] Ed Huai-hsin Chi and John Riedl. 1998. An Operator Interaction Framework for Visualization Systems. In *Proc. IEEE InfoVis (INFOVIS '98).* IEEE Computer Society, 63–70.
[15] Junghong Choi, Oh-hyun Kwon, and Kyungwon Lee. 2011. Strata Treemaps. In *Proc. ACM SIGGRAPH Posters (SIGGRAPH '11).* ACM, 87. https://doi.org/10.1145/2037715.2037813
[16] Stephan Diehl. 2007. *Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software.* Springer Science & Business Media.
[17] Selan R. dos Santos and Ken Brodlie. 2004. Gaining understanding of multivariate and multidimensional data through visualization. *Elsevier Computers & Graphics* 28 (2004), 311–325.

[18] S. Dübel, M. Röhlig, Heidrun Schumann, and Matthias Trapp. 2014. 2D and 3D presentation of spatial data: A systematic review. In *Proc. IEEE 3DVis*. 11–18. https://doi.org/10.1109/3DVis.2014.7160094

[19] Niklas Elmqvist and Jean-Daniel Fekete. 2010. Hierarchical Aggregation for Information Visualization: Overview, Techniques, and Design Guidelines. *IEEE TVCG* 16, 3 (2010), 439–454. https://doi.org/10.1109/TVCG.2009.84

[20] Stephen Few. 2004. Tapping the power of visual perception. *Visual Business Intelligence Newsletter* 39 (2004), 41–42.

[21] M. Giereth, H. Bosch, and T. Ertl. 2008. A 3D treemap approach for analyzing the classificatory distribution in patent portfolios. In *Proc. IEEE VAST*. 189–190. https://doi.org/10.1109/VAST.2008.4677380

[22] J. Görtler, C. Schulz, Daniel Weiskopf, and Oliver Deussen. 2017. Bubble Treemaps for Uncertainty Visualization. *IEEE TVCG* (2017). https://doi.org/10.1109/TVCG.2017.2743959

[23] Robert B Haber and David A McNabb. 1990. Visualization idioms: A conceptual model for scientific visualization systems. *Visualization in Scientific Computing* 74 (1990), 93.

[24] Haleh Hagh-Shenas, Victoria Interrante, Christopher Healey, and Sunghee Kim. 2006. Weaving Versus Blending: A Quantitative Assessment of the Information Carrying Capacities of Two Alternative Methods for Conveying Multivariate Data with Color. In *Proc. ACM APGV*. 164–164. https://doi.org/10.1145/1140491.1140541

[25] Sebastian Hahn and Jürgen Döllner. 2017. Hybrid-Treemap Layouting. In *Proc. EG EuroVis – Short Papers*, Barbora Kozlikova, Tobias Schreck, and Thomas Wischgoll (Eds.). The Eurographics Association. https://doi.org/10.2312/eurovisshort.20171137

[26] Philipp N Hesse, Constanze Schmitt, Steffen Klingenhoefer, and Frank Bremmer. 2017. Preattentive processing of numerical visual information. *Frontiers in human neuroscience* 11 (2017), 70.

[27] Danny Holten. 2006. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *IEEE TVCG* 12, 5 (2006), 741–748. https://doi.org/10.1109/TVCG.2006.147

[28] Danny Holten, Roel Vliegen, and Jarke J. van Wijk. 2005. Visual realism for the visualization of software metrics. In *Proc. IEEE VISSOFT*. IEEE, 1–6.

[29] Takayuki Itoh, H. Takakura, A. Sawada, and K. Koyamada. 2006. Hierarchical visualization of network intrusion detection data. *IEEE CG&A* 26, 2 (2006), 40–47. https://doi.org/10.1109/MCG.2006.34

[30] Daniel A. Keim, Gennady Andrienko, Jean-Daniel Fekete, Carsten Gorg, Jorn Kohlhammer, and Guy Melançon. 2008. Visual analytics: Definition, process, and challenges. *Springer LNCS* 4950 (2008), 154–176.

[31] C. Knight and M. Munro. 1999. Comprehension with[in] virtual environment visualisations. In *Proc. WPC*. 4–11. https://doi.org/10.1109/WPC.1999.777733

[32] N. Kong, Jeffrey Heer, and M. Agrawala. 2010. Perceptual Guidelines for Creating Rectangular Treemaps. *IEEE TVCG* 16, 6 (2010), 990–998. https://doi.org/10.1109/TVCG.2010.186

[33] A. Kuhn, P. Loretan, and O. Nierstrasz. 2008. Consistent Layout for Thematic Software Maps. In *Proc. WCRE*. 209–218. https://doi.org/10.1109/WCRE.2008.45

[34] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. 2005. Visualization-based Analysis of Quality for Large-scale Software Systems. In *Proc. ASE (ASE '05)*. ACM, 214–223. https://doi.org/10.1145/1101908.1101941

[35] Daniel Limberger, Carolin Fiedler, Sebastian Hahn, Matthias Trapp, and Jürgen Döllner. 2016. Evaluation of Sketchiness as a Visual Variable for 2.5 D Treemaps. In *Proc. IEEE iV*. IEEE, 183–189.

[36] Daniel Limberger, Marcel Pursche, Jan Klimke, and Jürgen Döllner. 2017. Progressive High-quality Rendering for Interactive Information Cartography Using WebGL. In *Proc. ACM Web3D*. ACM, 8:1–8:4. https://doi.org/10.1145/3055624.3075951

[37] Daniel Limberger, Willy Scheibel, Sebastian Hahn, and Jürgen Döllner. 2017. Reducing Visual Complexity in Software Maps using Importance-based Aggregation of Nodes. In *Proc. IVAPP*. INSTICC, SciTePress, 176–185. https://doi.org/10.5220/0006267501760185

[38] Daniel Limberger, Willy Scheibel, Stefan Lemme, and Jürgen Döllner. 2016. Dynamic 2.5D Treemaps Using Declarative 3D on the Web. In *Proc. ACM Web3D (Web3D '16)*. ACM, 33–36. https://doi.org/10.1145/2945292.2945313

[39] Daniel Limberger, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. 2017. Mixed-Projection Treemaps: A Novel Approach Mixing 2D and 2.5D Treemaps. In *Proc. IEEE iV*. 164–169. https://doi.org/10.1109/iV.2017.67

[40] Daniel Limberger, Matthias Trapp, and Jürgen Döllner. 2018. Interactive, Height-Based Filtering in 2.5D Treemaps. In *Proc. ACM VINCI (VINCI '18)*. ACM, 49–55. https://doi.org/10.1145/3231622.3231638

[41] Daniel Limberger, Matthias Trapp, and Jürgen Döllner. 2019. In-Situ Comparison for 2.5D Treemaps. In *Proc. IVAPP*. INSTICC, SciTePress.

[42] Daniel Limberger, Benjamin Wasty, Jonas Trümper, and Jürgen Döllner. 2013. Interactive Software Maps for Web-based Source Code Analysis. In *Proc. ACM Web3D (Web3D '13)*. ACM, 91–98. https://doi.org/10.1145/2466533.2466550

[43] S. Liu, D. Maljovec, B. Wang, P. Bremer, and V. Pascucci. 2017. Visualizing High-Dimensional Data: Advances in the Past Decade. *IEEE TVCG* 23, 3 (2017), 1249–1268. https://doi.org/10.1109/TVCG.2016.2640960

[44] Martin Luboschik and Heidrun Schumann. 2008. Discovering the covered: Ghost-views in information visualization. In *Proc. WSCG*.

[45] Andrian Marcus, Louis Feng, and Jonathan I. Maletic. 2003. 3D Representations for Software Visualization. In *Proc. ACM SoftVis (SoftVis '03)*. ACM, 27–ff. https://doi.org/10.1145/774833.774837

[46] P. Molli, H. Skaf-Molli, and C. Bouthier. 2001. State Treemap: an awareness widget for multi-synchronous groupware. In *Proc. CRIWG*. 106–114. https://doi.org/10.1109/CRIWG.2001.951823

[47] P. Samuel Quinan, Lace Padilla, Sarah H. Creem-Regehr, and Miriah Meyer. 2019. Examining Implicit Discretization in Spectral Schemes. *EG CGF (from EuroVis '19), to appear* (2019).

[48] Willy Scheibel, Stefan Buschmann, Matthias Trapp, and Jürgen Döllner. 2017. *Attributed Vertex Clouds*. Bowker Identifier Services, Chapter Geometry Manipulation, 3–21.

[49] Willy Scheibel, Judith Hartmann, and Jürgen Döllner. 2019. HiViSer API – Provisioning of Hierarchy Visualizations in the Web. In *Proc. IVAPP*. INSTICC, SciTePress.

[50] Willy Scheibel, Matthias Trapp, and Jürgen Döllner. 2016. Interactive Revision Exploration using Small Multiples of Software Maps. In *Proc.IVAPP*. INSTICC, SciTePress, 131–138. https://doi.org/10.5220/0005694401310138

[51] Willy Scheibel, Christopher Weyand, and Jürgen Döllner. 2018. EvoCells - A Treemap Layout Algorithm for Evolving Tree Data. In *Proc. IVAPP*. INSTICC, SciTePress.

[52] K. B. Schloss, C. C. Gramazio, A. T. Silverman, M. L. Parker, and A. S. Wang. 2018. Mapping Color to Meaning in Colormap Data Visualizations. *IEEE TVCG* (2018). https://doi.org/10.1109/TVCG.2018.2865147

[53] Hans-Jörg Schulz, Steffen Hadlak, and Heidrun Schumann. 2011. The design space of implicit hierarchy visualization: A survey. *IEEE TVCG* 17, 4 (2011), 393–411.

[54] Ben Shneiderman. 1996. The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations. In *Proc. IEEE VL*. IEEE Computer Society, 336. https://doi.org/10.1109/VL.1996.545307

[55] Ben Shneiderman. 2009. *Treemaps for space-constrained visualization of hierarchies*. Technical Report. Human-Computer Interaction Lab. http://www.cs.umd.edu/hcil/treemap-history.

[56] A. G. Soares, D. H. dos Santos, C. L. Barbosa, A. S. Goncalves, C. G. dos Santos, B. S. Meiguins, and E. T. Miranda. 2018. Visualizing Multidimensional Data in Treemaps with Adaptive Glyphs. In *Proc. Proc. iV*. 58–63. https://doi.org/10.1109/iV.2018.00021

[57] Marcel Steinbeck, Rainer Koschke, and Marc O. Rüdel. 2019. Comparing the EvoStreets Visualization Technique in Two D and Three-dimensional Environments: A Controlled Experiment. In *Proc. IEEE ICPC (ICPC '19)*. IEEE Press, 231–242. https://doi.org/10.1109/icpc.2019.00042

[58] Frank Steinbrückner and Claus Lewerentz. 2013. Understanding software evolution with software cities. *Information Visualization* 12, 2 (2013), 200–216. https://doi.org/10.1177/1473871612438785

[59] Sidharth Thakur and Theresa-Marie Rhyne. 2009. Data Vases: 2D and 3D Plots for Visualizing Multiple Time Series. In *Proc. ISVC Part II*. Springer Berlin Heidelberg, 929–938. https://doi.org/10.1007/978-3-642-10520-3_89

[60] Matthias Trapp, Sebastian Schmechel, and Jürgen Döllner. 2013. Interactive rendering of complex 3d-treemaps. In *Proc. GRAPP*. 165–175.

[61] Y. Tu and H. W. Shen. 2007. Visualizing Changes of Hierarchical Data using Treemaps. *IEEE TVCG* 13, 6 (2007), 1286–1293. https://doi.org/10.1109/TVCG.2007.70529

[62] David Turo and Brian Scott Johnson. 1992. Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation. In *Proceedings IEEE Vis (VIS '92)*. IEEE Computer Society Press, 124–131.

[63] Jarke J. van Wijk. 2005. The value of visualization. In *Proc. IEEE Vis*. 79–86. https://doi.org/10.1109/VISUAL.2005.1532781

[64] Jarke J. van Wijk and Huub van de Wetering. 1999. Cushion treemaps: Visualization of hierarchical information. In *Proc. IEEE InfoVis*. IEEE, 73–78.

[65] Colin Ware. 2012. *Information visualization: perception for design*. Elsevier.

[66] Richard Wettel and Michele Lanza. 2007. Program Comprehension Through Software Habitability. In *Proc. IEEE ICPC (ICPC '07)*. IEEE Computer Society, 231–240. https://doi.org/10.1109/ICPC.2007.30

[67] Richard Wettel and Michele Lanza. 2008. CodeCity: 3D Visualization of Large-scale Software. In *Companion of the 30th International Conference on Software Engineering*. ACM, 921–922. https://doi.org/10.1145/1370175.1370188

[68] Hannes Würfel, Matthias Trapp, Daniel Limberger, and Jürgen Döllner. 2015. Natural Phenomena as Metaphors for Visualization of Trend Data in Interactive Software Maps. In *Proc. CGVC*. The Eurographics Association. https://doi.org/10.2312/cgvc.20151246

[69] Yingtao Xie, Tao Lin, Rui Chen, and Zhi Chen. 2017. Toward improved aesthetics and data discrimination for treemaps via color schemes. *Wiley CR&A* (2017), to be published. https://doi.org/10.1002/col.22196

[70] Jin Zhang. 2008. The Implication of Metaphors in Information Visualization. In *Visualization for Information Retrieval*. Vol. 23. Springer, 215–237.