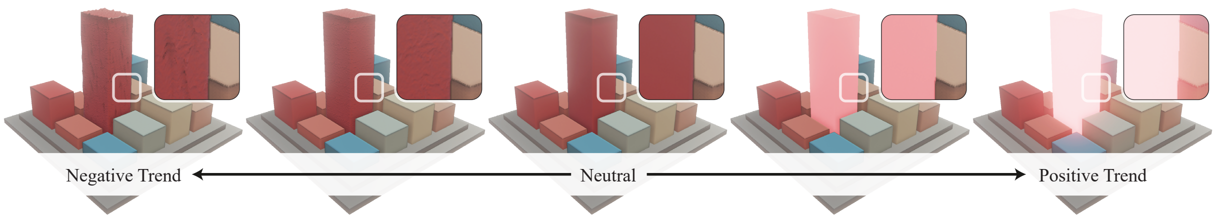
EG UK Computer Graphics & Visual Computing (2015)
Rita Borgo, Cagatay Turkay (Editors)

Natural Phenomena as Metaphors for Visualization of Trend Data in Interactive Software Maps

H. Würfel, M. Trapp, D. Limberger, and J. Döllner

Computer Graphics Systems Group, Hasso Plattner Institute, University of Potsdam, Germany



Abstract

Software maps are a commonly used tool for code quality monitoring in software-development projects and decision making processes. While providing an important visualization technique for the hierarchical system structure of a single software revision, they lack capabilities with respect to the visualization of changes over multiple revisions. This paper presents a novel technique for visualizing the evolution of the software system structure based on software metric trends. These trend maps extend software maps by using real-time rendering techniques for natural phenomena yielding additional visual variables that can be effectively used for the communication of changes. Therefore, trend data is automatically computed by hierarchically aggregating software metrics. We demonstrate and discuss the presented technique using two real world data sets of complex software systems.

Categories and Subject Descriptors (according to ACM CCS): D.2.2 [Software Engineering]: Design Tools and Techniques—Computer-aided software engineering (CASE) D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—Documentation I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism —Color, shading, shadowing, and texture

1. Introduction

Almost every software system is subject to constant change due to evolving technology and user requirements. Its continuous development is largely driven by corrective, adaptive, perfective as well as preventive maintenance [LS80]. The responsible development teams undergo changes as well and developers not involved in the previous development process are required to spent a considerable amount of their time understanding the system under maintenance. Furthermore, creating a mental model of a software is difficult, as it is virtual and intangible [KM00], thus, making it hard to communicate the structure and behavior of the system to decision makers. Visualization tools can provide communication artifacts such as software maps that assist software engineering and maintenance, e.g., by allowing for actionable insights.

In recent years software maps were established to communicate static, dynamic, and evolutionary information of software systems as well as their software development processes by means of information visualization. The application of 2D or 3D map-oriented techniques constitute a fundamental approach in software visualization, software analytics, and software diagnosis. Its primary applications include risk analysis for and monitoring of code quality, team activity, or software development progress [BD11].

Evolution of Software Systems The evolution of a software system can be tracked by, e.g., revision control systems, making it possible to analyze changes over revision ranges by sampling revisions over time. This information can be used for example to retrace, whether a design decision (e.g., an architectural refactoring) resulted in an expected change that manifests itself in an improvement of respective *software metrics*. Existing approaches either use transition animations between multiple revisions to depict the evolution process or they convey a progress within a single static image: Transition animations have the advantage of keeping the established metric mappings but do not scale for large software system or revision ranges with much change information to be displayed at once. Conveying the characteristics of an evolution process within a static image does not suffer from this problem but existing approaches sacrifice at least one attribute mapping to encode an evolution measure.

Metaphors for Trend Visualization To overcome those shortcomings, a software map which uses natural phenomena as metaphors to visualize software metric trends within a static representation (*trend map*) is used. In this paper, a static representation of the trend map denotes a visualization artifact that exhibits no transition animation with a revision range, but it can be explored using 3D interaction metaphors [JH12]. It combines the visualization of the static system structure with evolutionary information by augmenting interactive software maps with additional visual variables represented by natural phenomena (e.g., fire, rain) or material properties (e.g., rust, shininess, glow). Thereby, the visual variables that are used to depict the properties of the static system structure by mapping software metrics to color, area, and height can be maintained. Furthermore, the automatic computation of trends in metrics (*metrics trends*) enables forecasting their future course, e.g., increase, decrease, and stagnation. With the support of specific software metrics trends, decision makers can initiate preventive maintenance actions.

When faced with unfamiliar concepts, our cognitive system searches for the best mapping between the unknown concept and existing knowledge of other domains [Zha08]. Software maps use a virtual city metaphor yielding a familiar 2.5D reference geometry. In this context, the application of natural phenomena as metaphors seems promising, since it can be assumed that they closely match to virtual 3D environments: At best, their semantics are intuitively decoded making an additional legend non-obligatory. Examples of natural phenomena are objects aging in terms of their associated materials or degeneration within their environment (overgrown by grass, silted, gathering dust), changing in radiant emittance, as well as being exposed to natural forces such as fire, rain, wind, or sunshine (Figure 1).

Challenges & Contributions A main challenge is the identification of natural phenomena that are suitable to effectively communicate trend data. For it, metric values have to be

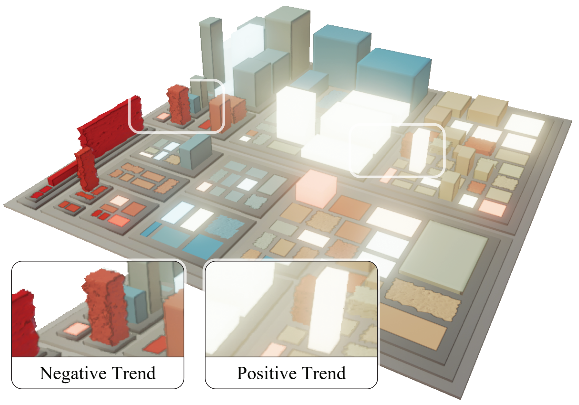


Figure 1: *This example depicts the trend of a system's test-coverage: Area comprises lines-of-code, height relates to source code complexity, and color indicates test-coverage (red for lower and blue for higher coverage). Glowing items depict a positive trend, while a rough surface symbolizes a decrease in test-coverage during a specific time interval.*

mapped to parameters of respective rendering techniques in a comprehensive way. Furthermore, to extend their application to hierarchical structuring elements (e.g., software modules) instead of individual leaf items only, hierarchical metric value aggregation is required. All in all, the challenges for an interactive visualization of trend data using natural phenomena as metaphors within software maps comprise (1) a concept for mapping natural phenomena to software map items, (2) the aggregation of software metric values within revision ranges, (3) the metrics based parametrization of rendering techniques, and (4) rendering within real-time constraints.

With respect to these challenges, this paper contributes:

1. A novel concept for the visualization of trend data within interactive software maps using natural phenomena.
2. A technique for hierarchical aggregation of software system information and metrics automatically extracted from, e.g., , revision control systems as well as a respective mapping to parametrize the real-time rendering techniques used for synthesis of natural phenomena.
3. A prototypical implementation using the Unreal Engine 4 [Epi15] capable of handling actual software system data.

The remainder of this paper is structured as follows. Related work with respect to interactive software visualization is reviewed in Section 2. Section 3 presents a concept for trend visualization using item-based and context-based metaphors and Section 4 describes our hierarchical metric aggregation. A prototype is briefly described in Section 5 and demonstrated using actual software system data presented and discussed in Section 6. Finally, Section 7 concludes this work.

2. Related Work

Related work comprises the fields of 2.5D software maps and metaphor-based visualization of software system evolution.

Interactive Software Maps Treemaps are commonly used to visualize hierarchical datasets. The original space-filling algorithm introduced by [Shn92] slices parent nodes recursively by their child nodes properties, alternating in horizontal and vertical direction. Due to undesirable aspect ratios produced by this slice and dice algorithm, Bruel et al. as well as Bederson and Shneiderman proposed squarified treemaps [BHvW99] and strip treemaps [BSW02]. Furthermore, while preserving the visualization of the hierarchy, Fekete and Plaisant added visual attributes, such as color and size, to treemap items [FP02]. The first 2.5D treemap was described in the *Step Tree* approach presented by Bladh et al. [BCS04]. The geometric representation of sub-directories is stacked on top of their parent directories. This extension to the third dimension enables metaphor-based visualizations as shown in *Information Pyramids* [AWP97] and *Code City* [WL08], a city metaphor for 3D treemaps in which the height of treemap's cuboids is used to map an additional software system metric. Similar to actual cities, these code cities are partitioned into several districts, including downtown areas and suburbs. Referenced as *software maps*, the city metaphor was adopted and extended by several other publications [LHM*09, BD11]. Bohnet and Döllner describe the usage of software maps as a decision-making tool. Recently, a comparative performance evaluation of rendering techniques for 2.5D treemaps was presented [TSD13]. With it, a deferred shape-generation algorithm outperforming existing approaches is presented. The basic idea is to generate and render an attributed point-cloud and to expand the points to cuboids on the GPU.

Visualization of Software Evolution The visualization of software evolution implies an additional time dimension. Most techniques visualize the change process by animating the transition phase between two software revisions. The *VERSO visualization tool* [LSP08] uses linear interpolation of color, height, and twist between two revisions. Another approach uses an Evolution Storyboard consisting of an ordered sequence of animated panels showing the structural changes during a time period [DB06]. In contrast to animation, the Evolution Matrix approach displays the entire evolution of the software system in a single image [Lan01]. Rectangles are used to map the number of methods to width and the number of attributes to height. Each revision is visualized next to its consecutive revision in a separate column.

Metaphors in Software Visualization Besides the city metaphor, other metaphors for software system visualization were proposed. Balzer et al. presented a software landscape metaphor [BNDL04] using landscape-like distributions of three dimensional objects on a 2D reference plane. Depending relationships between subsystems are visualized using a hierarchical network. Further, Graham et al. proposed a solar-system metaphor [GYB04] for object-oriented software metrics. Planets represent classes, orbits depict the inheritance level within a package: planets in each orbit represent an inheritance hierarchy, while the lines-of-code of a class are mapped to the planet size. A thoroughly written survey of visualization techniques for static and evolutionary information of software systems is presented by Caserta and Zendra [CZ11]. Holten et al. presented a 2D treemap visualization of software metrics using the surface structure of treemap elements by applying texture- and bump mapping to them [HVvW05]. Their approach is based on the fact that the human visual system can rapidly process visual cues such as shading and textures [Enn90, HE98, HE99]. Panas et al. use a realistic city metaphor to map code quality metrics to building textures [CZ11]. Furthermore, they proposed visual effects such as fire and bolts to communicate hot spot information in code execution and frequent component modifications [PBG03]. However, neither an implementation nor an evaluation of such visual mappings are provided. An empirical study by Borgo et al. suggests that visual embellishments aid memorization performance in both accuracy and response time, though it might have a negative impact on speed of visual search [BARM*12]. It further proves that visual embellishments aid concept grasping.

3. Natural Phenomena as Metaphors for Software Maps

The proposed concept of trend maps is a metaphorical extension to existing interactive software maps. However, there are various definitions of metaphors in multiple science disciplines. Hereinafter, a metaphor consists of a *target domain*, *target item*, *source domain*, *source item*, and *matching part* [Zha08]. The source item (e.g., weathering effect) from a well-known source domain (natural phenomena) is mapped to target items (cuboids representing software entities) of the target domain (software maps). The target domain is relatively unknown and the match will never be perfect. However, the larger the matching part the better the metaphor works.

3.1. Preliminaries and Assumptions

For the usage of natural phenomena we assume an appropriate match. This assumption is made due to our prior experience in assessing the semantics of natural phenomena. Furthermore, it is based on human usage of linguistic metaphors in idioms concerning natural phenomena such as “to go up in flames” and “grow grass upon”. With respect to the application and design of natural phenomena, we further assume that their image-synthesis can be performed in real-time to facilitate an interactive system. Therefore, this paper focuses on hardware-accelerated rasterization for real-time rendering [AMHH08].

3.2. Conceptual Overview

Our approach is structured as pipeline comprising data mining, metrics aggregation, and trend map rendering (Figure 2).

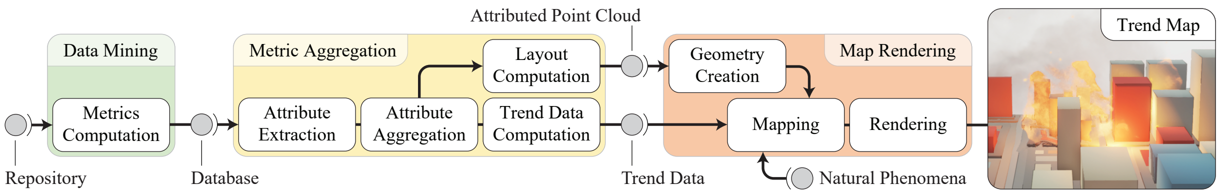


Figure 2: *Schematic overview (data and control flow) of our trend map pipeline comprising three conceptual stages.*

Data Mining At first, software metrics are computed on a per-item basis. Therefore, each revision of a software system is analyzed independently. The resulting artifact is a *database* encoding a revision tree per revision. This step is performed once for a given repository and allows for successive updates if new revisions have to be considered.

Metric Aggregation In this stage, a revision range $R = [n,\ m]$ with $n, m \in \mathbb{N}, n < m$ is specified by the user and used to compute the trend data later. The user also selects the metrics of which he wants to observe trends. The trend data of every treemap item is defined as a time series of metric data points over the revision range: For example, items that already existed at the start revision n and outlast the end revision m, consist of a time series with $m - n + 1$ data points. Trend data is then aggregated with respect to particular parent items of the hierarchy. Subsequently, a treemap layout is computed and encoded as *attributed point-cloud* for efficient rendering in the next stage. Simultaneously, the *trend data* is computed by analyzing the time series for all treemap items.

Map Rendering The rectangular treemap geometry is created based on the attributed point cloud. Trends are mapped to the parametrization of natural phenomena (Section 4). Depending on the degree of change, five discrete levels are depicted in one trend map to facilitate visual distinction: strong increase, moderate increase, stagnation, moderate decrease, and strong decrease. Up to two distinct effects can be specified by the user, one for negative and one for a positive trend. The effects are parametrized according to the computed trend and combined with the static system structure encoded in the attributed point-cloud. The latter is used for final image synthesis (Section 5).

4. Design and Parametrization of Natural Phenomena

Before focusing on suitable metaphors and their automatic parametrization, this section describes the design space of natural phenomena and how these can be mapped to software maps for effective visual communication. Considering 2.5D treemaps as reference geometry [DRST14], a differentiation between *item-based* and *context-based* metaphors, can be made with respect to the *design elements*:

Item-based metaphors Modifying the visualization of a single treemap item by changing its surface appearance using additional or altered materials,

Context-based metaphors Modifying a group of treemap items or modifying their surrounding context (3D space) of the visualization elements affected.

4.1. Item-based Metaphors

The surface properties of a treemap cuboid define a material that determines how light interacts with them and hence how it is perceived. To find suitable metaphors to communicate the change of an object state over time we pair contrary phenomena (Figure 3). We identified the roughness of a surface to be a suitable metaphor to visualize "how neat" a component implementation is: A negative trend is depicted by a high surface roughness, optionally amplified by texture based (displacment mapping) as well as geometry based (tessellation) displacement (*rough*). In contrast thereto, a positive trend is depicted using a low surface roughness in combination with a high specular reflection property, resulting in a shiny and clean cuboid (*shiny*). Concerning the evolution of test coverage of individual items, corrosion is a suitable metaphor. Thereby, high degree of corrosion relates to decreasing test coverage (*rusty*). Due to the fact that no corrosion would result in the neutral state (which results from the applied color mapping), a high radiant emittance (glow) is used to effectively communicate a positive trend in test coverage (*glowing*).

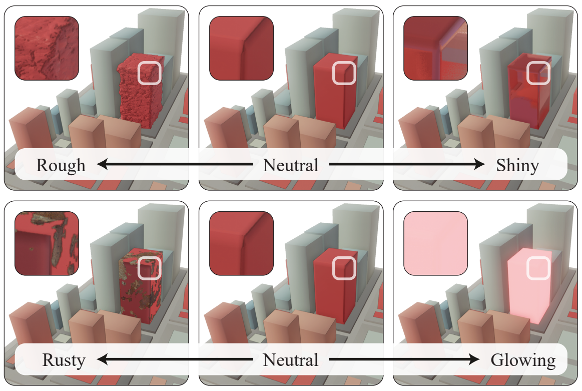


Figure 3: *Two examples for item-based metaphors covering the transition between negative and positive object states.*

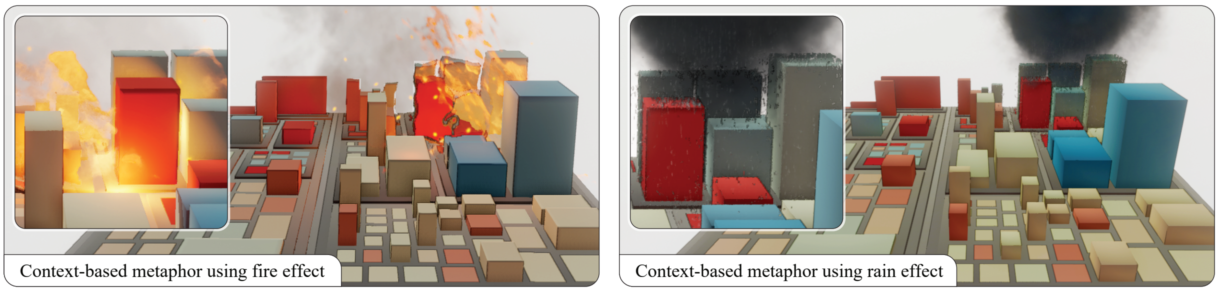

Context-based metaphor using fire effect
Context-based metaphor using rain effect

Figure 4: *Context-based metaphors, automatically assigned to development hot spots (e.g., strong decrease or stagnation).*

4.2. Context-based Metaphors

Context-based metaphors are either natural phenomena or effects that affect the area surrounding treemap items or a group of adjacent items. In order to parametrize such effects properly, context-based metaphors require information about the hierarchical aggregation of software metrics to determine trends for parent items. Particle systems are mainly used to visually communicate hot spots (or regions-of-interest) in source code, e.g., so-called "code smells". Therefore, we use a fire effect or rain (Figure 4) to indicate negative trends.

4.3. Trend-based Parametrization

To fully automate the trend-based parametrization, each effect contains a self description comprising all of its parameters. A single *parameter description* consists of the *parameter type* (i.e., vector, scalar, and color), the value range $[i_{\text{min}}, i_{\text{max}}]$ of the parameter v, a *default value* and a boolean value that determines whether to *inverse* the mapping or not (e.g., a positive trend is mapped to a roughness of zero). The parametrization is performed by mapping the trend v from an input trend value range $[i_{\text{min}}, i_{\text{max}}]$ to an output parameter value range $[j_{\text{min}}, j_{\text{max}}]$ by $v_{\text{mapped}} = (v - i_{\text{min}}) \frac{j_{\text{max}} - j_{\text{min}}}{i_{\text{max}} - i_{\text{min}}} + j_{\text{min}}$. Every effect defines parameter sub-ranges that discretize the mapped parameter value v_{mapped} to the five categories: *strong increase*, *increase*, *stagnation*, *decrease*, or *strong decrease*.

4.4. Hierarchical Aggregation of Software Metrics

Similar to software maps, the hierarchical structure of trend map mirrors the directory tree of the current revision chosen from a code repository. To determine software metric trends for individual (i.e., files) as well as groups of items (e.g., packages), trend data is aggregated bottom-up within the hierarchy in a specified revision range. Post-order and reversed level-order traversals are suitable for this task. Therefore, a node in a revision range tree holds a bucket k for each revision within the revision range. The respective metric value x_k contained in the bucket is computed using *weighted average*. Empty buckets represent items that are not present in a specific revision. The weights w_k are determined by counting the leaf nodes in the sub-tree where the node for which we want to aggregate the metric values is the root node.

5. Implementation Overview

In contrast to previous work in the field of software visualization, our approach is based on utilizing the rendering capabilities of modern GPUs for image synthesis of 2.5D treemaps that are augmented with complex surface materials and particle systems in real-time (Figure 5).

5.1. Rendering Engine Integration

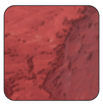
We have chosen to evaluate the Unreal Engine 4 (*UE4*) for our prototypical implementation since it offers us to rapidly prototype materials and particle systems for our purposes. At the date of writing, Unreal Engine 4.8 is the latest iteration of the game engine created by Epic Games Inc. It offers libraries and tools as well as the engines source code to develop graphics applications. Game development in C++ with UE4 works by implementing a shared library (game module) which is loaded by the engine at runtime. To change or extend the engine functionality there are two options: source code modifications and plugins. We have integrated our own software visualization library using plugins and instead of our own renderer, we use the unreal renderer. The game module uses the plugin to visualize the trend map and implements a basic navigation and graphical user interface to interact with it.

5.2. Surface Materials for Item-based Metaphors

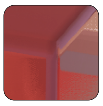
Shaders for rendering surfaces are denoted as *surface materials* in UE4. Usually, they are developed using a visual interface in a material editor based on creating a directed acyclic graph (*DAG*) of operations. Internally, these DAGs are compiled to shader code. UE4 implements a physically-based shading model with a small set of intuitive parameters (base color, metallic, specular, roughness, emissive color).

Rust is modeled using multi-texturing: The combination of color and texture (rust distribution) is controlled using Perlin noise with two octaves.

Radiant emittance is modeled by setting the base color as well as the emissive color multiplied by an emissive strength to the vertex color of the respective cuboid.



Roughness is implemented using a combination of geometry tessellation, normal mapping, and vertex displacement along a normal vector that represents the original surface orientation modified using a 3D noise function.



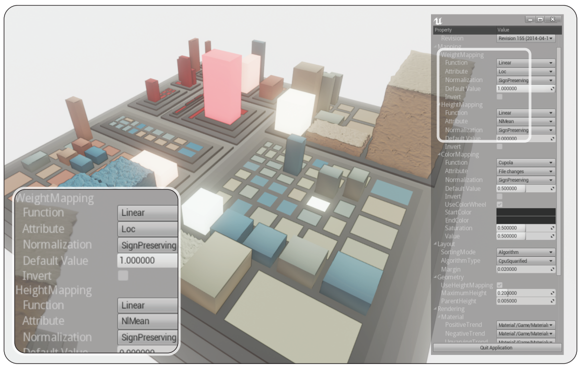
Shininess is created using the metallic parameter, with a high specular value and roughness close or equal to zero.

5.3. Particle Systems for Context-based Metaphors

Similar to surface materials, particle systems can be configured visually in an interactive editor. Multiple emitters can be specified and each emitter comprises various parameters such as spawning rate, collision behavior with scene geometry, and particle color and size. Ideally, a particle system is instantiated inside the bounding box of its affected item group. Since the engine (as of this writing) does not offer the option to specify a dynamic volume for the particle systems, we work around this by spawning several particle systems inside the bounding box. To ensure a minimum distance between multiple systems, Poisson disk sampling is used [Bri07].

5.4. Automatic Effect Parametrization

For item-based effects, the parametrization is based on the material system. The material parameters are mirrored in a material representation in main memory. We change material properties at runtime by querying the materials defined as negative or positive trend as well as their corresponding parameter descriptions, and finally applying a trend-dependent parametrization. The trend is computed over a revision range by analyzing the time series using linear regression. By assuming that the underlying trend function is linear, the slope of the trend line can be used to discretize the trend easily into the previously used five categories.



WeightMapping
Function: Linear
Attribute: Loc
Normalization: SignPreserving
Default Value: 1.000000
Invert
HeightMapping
Function: Linear
Attribute: NlMean
Normalization: SignPreserving
Default Value: 0.000000

Figure 5: *Unreal editor showing a trend map during setup.*

However, a linear regression model for time series is rarely the best choice, since (1) its deviation from the data is often larger at the end of the time series or (2) the underlying data is simply not linear. Our system is designed to be extensible to different trend computation models, since other prediction models might be better suited. For an overview of statistical forecasting methods and respective applications scenarios we refer to Robert F. Nau [Nau15].

A material is also an implementation of an element-based effect. Context-based metaphors using surface properties are implemented top-down by propagating aggregated metric values down the hierarchy. Particle systems are handled in a similar manner.

6. Results and Discussion

The results of our technique are discussed by means of application examples using real-world data sets and directions for future research are provided. With respect to to performance, the technique was tested using datasets of two software systems with different complexity, i.e., different static structure components and number of revisions (Figure 6):

Data Set	#Rev.	Rev. Range	max. # Items
libzeug	248	[130,155]	243
PocoProject	2038	[1000,1200]	2635

For each revision, the repository was downloaded, making it yet impractical for repositories containing binary files. Gathering the revision data, computing and aggregating metric values for 72 software metrics for every revision of the master branch resulted in expensive preprocessing. However, both projects can be rendered at interactive frame rates:

Data Set	Metric Comp.	Aggr.	Rendering
libzeug	appr. 04h	04sec	16ms
PocoProject	appr. 48h	17min	43ms

Our trend maps facilitate the depiction of metric changes over a revision range by extending software maps of a certain revision $x \in R$. Hence, a conceptual limitation of our approach, is that a trend map only shows trends for items which are still present at the respective revision x. A proper visualization of deleted or added system components within a software map is still subject to research.

Besides the static image approach for our trend maps, we have also tested several animated transitions for trend data of a revision range. We tested linear interpolation and cubic interpolations using uniform, chordal, and centripetal catmull-rom splines [YSK09]. By additionally rendering these trend curves the user gets a familiar view to explore the trend data of a particular item or a group of items. However, we found that using these trend curves to parametrize the natural phenomena during a transition animation overstrains the user's perception for complex real-world data sets, since there are too many change processes visualized simultaneously.

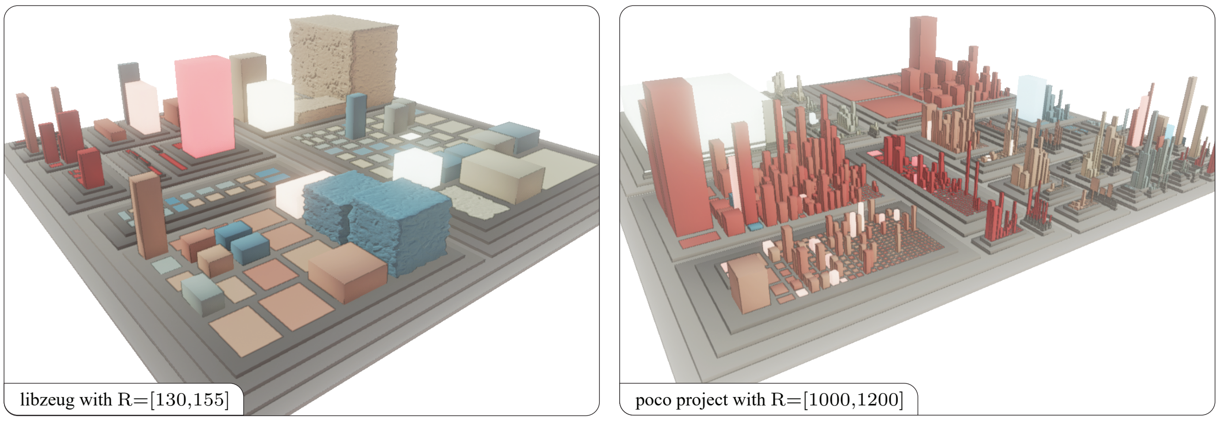


Figure 6: *Applications of the presented visualization technique using two real data sets using item-based natural phenomena as metaphors. The underlying software maps encode lines-of-code to item area, the mean nesting-level to height, and the number of file changes within the revision range to color. Left: the trends of the McCabe metric is mapped into the range between a rough surface and glow. Right: the inverted trend of the lines-of-code metric is mapped using the same metaphor.*

The presented approach can be used to facilitate overview of hot spots in software maps according to the information “seeking mantra: overview first, zoom and filter, then details-on-demand” [Shn96]. Especially effects such as glow seem to support pre-attentive processing and can be used for highlighting purposes, but in contrast to rust or shiny effects, it alters the item’s color mapping significantly and complicates the perception of shape. However, the perception of surface effects such as rust or corrosion depends on items’ screen sizes. Also, when depicting larger revision ranges the number of affected items and regions tend to cause cluttering and context-based metaphors (e.g., fire) comprising large regions-of-interest sometimes cause occlusion.

Future Work We plan to focus our research on evaluation and adjustment of the visual metaphors by conducting a comprehensive user study. With respect to this, research on how certain effects (e.g., fire or thunderstorms) are able to emotionalize. Furthermore, care has to be taken on how many natural phenomena implemented can be applied simultaneously and still be perceived and distinguished by users (Figure 7). Another topic for future work can be to evaluate different forecasting methods for software metrics and directly map them to metaphors commonly used in daily weather forecasting visualizations. To allow for natural phenomena to be used on larger data sets, level-of-detail (LoD) compliance of the presented techniques has to be evaluated, e.g., for surface appearances, since fine surface details are hard to perceive if the virtual camera is far away from a region-of-interest.

7. Conclusions

This paper presented *trend maps*, a novel technique using natural phenomena as metaphors for visualizing the evolution of software system structures based on software metric trends. It has been demonstrated that by analyzing the time series of software metrics values and visualizing their trends using our technique, it is possible to communicate the (1) number, (2) magnitude, and (3) gradient of software metric changes. Based on our experiences with the prototypical implementation, natural phenomena can also be used to emotionalize the visual communication by providing memorable visualizations of the software system.

This research was concerned with the application of natural phenomena within interactive software maps; however, the results can be applicable also to other fields in information visualization, though, caution for applying metaphors in visualization is remains advisable.

Acknowledgments

The authors would like to thank the anonymous reviewers as well as Martin Röhlig for their valuable comments. This work was funded by the Federal Ministry of Education and Research (BMBF), Germany, within the InnoProfile Transfer research group “4DnD-Vis” (`www.4dndvis.de`).

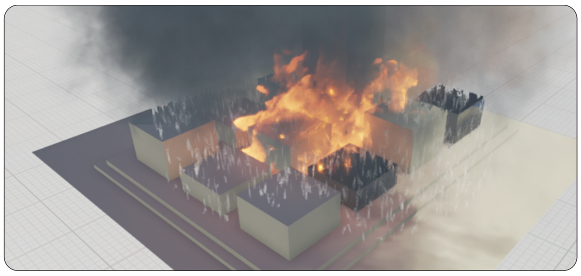


Figure 7: *Superposition of multiple natural phenomena.*

References

[AMHH08] AKENINE-MÖLLER T., HAINES E., HOFFMAN N.: *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., 2008. 3

[AWP97] ANDREWS K., WOLTE J., PICHLER M.: Information pyramids^{TM}: A new approach to visualising large hierarchies. In *Proc. of IEEE VIS* (1997), ACM Press, pp. 49–52. 3

[BARM*12] BORGO R., ABDUL-RAHMAN A., MOHAMED F., GRANT P. W., REPPA I., FLORIDI L., CHEN M.: An empirical study on using visual embellishments in visualization. *IEEE TVCG 18*, 12 (2012), 2759–2768. 3

[BCS04] BLADH T., CARR D. A., SCHOLL J.: Extending tree-maps to three dimensions: A comparative study. In *Proc. of APCHI* (2004), Springer Verlag, pp. 50–59. 3

[BD11] BOHNET J., DÖLLNER J.: Monitoring code quality and development activity by software maps. In *Proc. of ACM MTD* (2011), pp. 9–16. 1, 3

[BHvW99] BRULS M., HUIZING K., VAN WIJK J.: Squarified treemaps. In *Proc. of EG & IEEE VISSYM* (1999), pp. 33–42. 3

[BNDL04] BALZER M., NOACK A., DEUSSEN O., LEWERENTZ C.: Software landscapes: Visualizing the structure of large software systems. In *Proc. of VISSYM* (2004), Eurographics Association, pp. 261–266. 3

[Bri07] BRIDSON R.: Fast poisson disk sampling in arbitrary dimensions. In *ACM SIGGRAPH Sketches* (2007), ACM. 6

[BSW02] BEDERSON B. B., SHNEIDERMAN B., WATTENBERG M.: Ordered and quantum treemaps: Making effective use of 2d space to display hierarchies. *ACM Trans. Graph. 21*, 4 (2002), 833–854. 3

[CZ11] CASERTA P., ZENDRA O.: Visualization of the static aspects of software: A survey. *IEEE TVCG*, 7 (2011), 913–933. 3

[DB06] DIRK BEYER A. E. H.: Animated visualization of software history using evolution storyboards. In *Proc. of IEEE WCRE* (2006), IEEE Computer Society, pp. 199–210. 3

[DRST14] DÜBEL S., RÖHLIG M., SCHUMANN H., TRAPP M.: 2d and 3d presentation of spatial data: A systematic review. In *IEEE VIS International Workshop on 3DVis* (11 2014). 4

[Enn90] ENNS J. T.: *Three-Dimensional Features that Pop Out in Visual Search*. Taylor & Francis, 1990. 3

[Epi15] EPIC GAMES, INC.: Unreal Engine 4. `https://www.unrealengine.com/`, 2015. 2

[FP02] FEKETE J.-D., PLAISANT C.: Interactive information visualization of a million items. In *Proc. of IEEE InfoVis* (2002), IEEE, pp. 117–. 3

[GYB04] GRAHAM H., YANG H. Y., BERRIGAN R.: A solar system metaphor for 3d visualisation of object oriented software metrics. In *Proc. of APVis* (2004), Australian Computer Society, Inc., pp. 53–59. 3

[HE98] HEALEY C. G., ENNS J. T.: Building perceptual textures to visualize multidimensional datasets. In *Proc. of VIS* (1998), IEEE Computer Society Press, pp. 111–118. 3

[HE99] HEALEY C. G., ENNS J. T.: Large datasets at a glance: Combining textures and colors in scientific visualization. *IEEE TVCG 5*, 2 (1999), 145–167. 3

[HVvW05] HOLTEN D., VLIEGEN R., VAN WIJK J. J.: Visual realism for the visualization of software metrics. In *Proc. of VISSOFT* (2005), IEEE Computer Society, pp. 1–6. 3

[JH12] JANKOWSKI J., HACHET M.: A Survey of Interaction Techniques for Interactive 3D Environments. In *EG STAR* (2012), Sbert M., Szirmay-Kalos L., (Eds.). 2

[KM00] KNIGHT C., MUNRO M.: Virtual but visible software. In *Proc. of IEEE IV* (2000), pp. 198–205. 1

[Lan01] LANZA M.: The evolution matrix: Recovering software evolution using software visualization techniques. In *Proc. of IWPSE* (2001), ACM, pp. 37–42. 3

[LHM*09] LIGGESMEYER P., HEIDRICH J., MÜNCH J., KALCKLÖSCH R., BARTHEL H., ZECKZER D.: Visualization of software and systems as support mechanism for integrated software project control. In *Proc. of HCI* (2009), pp. 846–855. 3

[LS80] LIENTZ B. P., SWANSON E. B.: *Software Maintenance Management*. Addison-Wesley Longman Publishing Co., Inc., 1980. 1

[LSP08] LANGELIER G., SAHRAOUI H., POULIN P.: Exploring the evolution of software quality with animated visualization. In *Proc. of IEEE VLHCC* (2008), pp. 13–20. 3

[Nau15] NAU R. F.: Steps in choosing a forecasting model. `http://people.duke.edu/~rnau/411fcst.htm`, 2015. 6

[PBG03] PANAS T., BERRIGAN R., GRUNDY J.: A 3d metaphor for software production visualization. In *Proc. of IV* (2003), pp. 314–319. 3

[Shn92] SHNEIDERMAN B.: Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph. 11*, 1 (1992), 92–99. 3

[Shn96] SHNEIDERMAN B.: The eyes have it: A task by data type taxonomy for information visualizations. In *Proc. of IEEE VL* (1996), pp. 336–. 7

[TSD13] TRAPP M., SCHMECHEL S., DÖLLNER J.: Interactive rendering of complex 3d-treemaps with a comparative performance evaluations. In *Proc. of GRAPP/IVAPP* (2013), pp. 165–175. 3

[WL08] WETTEL R., LANZA M.: Codecity: 3d visualization of large-scale software. In *Proc. of ICSE* (2008), ACM, pp. 921–922. 3

[YSK09] YUKSEL C., SCHAEFER S., KEYSER J.: On the parameterization of catmull-rom curves. In *Joint Conference on Geometric and Physical Modeling* (2009), ACM, pp. 47–53. 6

[Zha08] ZHANG J.: The implication of metaphors in information visualization. In *Visualization for Information Retrieval*, Zhang J., (Ed.), vol. 23. Springer, 2008, pp. 215–237. 2, 3