Image-Schematic Metaphors in Software Visualization

David Heidrich[1], Jörn Hurtienne[2] and Andreas Schreiber[3]

[1]*German Aerospace Center (DLR), Institute for Software Technology, Weßling, Germany*
[2]*Julius-Maximilians-Universität, Chair of Psychological Ergonomics, Würzburg, Germany*
[3]*German Aerospace Center (DLR), Institute for Software Technology, Cologne, Germany*

Abstract

Software visualization (SoftVis) is widely used to facilitate the process of obtaining an in-depth understanding of complex software systems. SoftVis designers can draw on a wide pool of pre-existing conceptual metaphors that model the abstract target domain to tangible source domains. As regular user-centered design methods do not provide guidance on choosing metaphorical mappings, SoftVis designers choose conceptual metaphors primarily based on their subjective similarity to the underlying data structure. We want to include image-schematic metaphors in the SoftVis design process to provide designers with guidance on choosing visualization metaphors that are also in line with the users' mental model. This could allow SoftVis designers to make more data-driven design decisions and result in SoftVis that provides better insights.

Keywords

image schema, image-schematic metaphor, software visualization, intuitive use

1. Introduction

Maintaining a deep understanding of a software system, like its modular structure or dependencies between components, is crucial for programmer productivity to understand, change, and repair code [1]. As functionality is added to software systems, developers must spend more and more time on comprehension activities. Due to the abstract and complex nature of source code, this can quickly evolve into a mentally demanding and time consuming activity. In fact, professional developers invest $\sim 58\%$ of their working time on software comprehension instead of writing source code [2]. SoftVis tools are widely used to facilitate different aspects of this comprehension process [3], like familiarizing with an unknown software system [4, 5] or performing reverse engineering and debugging tasks [6, 7].

1.1. Software Visualization Metaphors

Conceptual metaphors that model the abstract target domain to tangible source domains are commonly used in SoftVis to make aspects of the target domain more understandable. By choosing source domains from the *real world*, SoftVis designers want to present the software in a familiar context [8] that relies on ”the human natural understanding of the physical world” [9].

The Seventh Image Schema Day (ISD7), September 2, 2023, Rhodes, Greece
david.heidrich@dlr.de (D. Heidrich); joern.hurtienne@uni-wuerzburg.de (J. Hurtienne); andreas.schreiber@dlr.de (A. Schreiber)
0000-0002-8501-5089 (D. Heidrich); 0000-0001-5295-9772 (J. Hurtienne); 0000-0001-5750-5649 (A. Schreiber)

SoftVis designers can draw on a wide pool of pre-existing conceptual metaphors (see Fig. 1), like the *City Metaphor* [10], the *Solar System Metaphor* [11], the *Island Metaphor* [12], or the *Forest Metaphor* [13]. While these metaphors use different source domains, most tend to follow the same hierarchical object-oriented structure (see Fig. 2). For that reason, these metaphors can be applied to a wide range of object-oriented languages, like Java, C#, or C++.

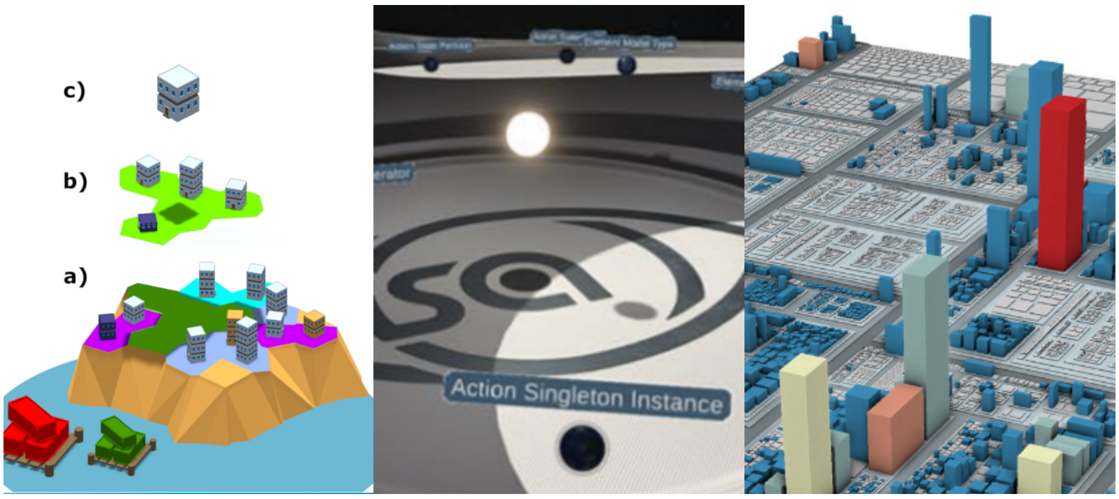
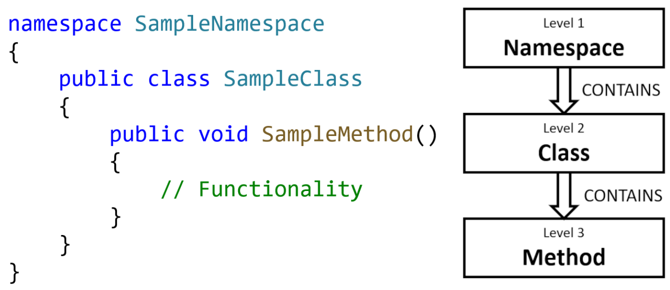


Figure 1: Examples of SoftVis tools using the island metaphor [14] (left), the solar system metaphor [15] (center), and the city metaphor [16] (right).



```
namespace SampleNamespace
{
    public class SampleClass
    {
        public void SampleMethod()
        {
            // Functionality
        }
    }
}
```

Figure 2: In the simplified structure of object-oriented source code, *Namespaces* contain *Classes* that contain *Methods*: The sample C# source code (left) with it's hierarchical structure (right).

User-centered design methods, like contextual design [17], can help SoftVis designers to identify relevant target domains. However, they do not provide guidance on choosing source domains for metaphorical mappings. As a result, SoftVis designers choose conceptual metaphors primarily based on their similarity to the underlying data structure. But when multiple metaphorical mappings fit the data structure, the final decision is generally based on subjective preferences of the SoftVis designer. Over the last decade, researchers have expressed the need for more data-driven design methods for specialized visualization metaphors [18, 19, 20].

2. Image-Schematic Metaphors

To facilitate data-driven design decisions, we can use design methods that align the SoftVis to the software developers' mental model of the target domain. One of these approaches is based on *image-schematic metaphors* (ISM) that are extracted from the users' subconscious mental model [21]. Image schemas are abstract representations of basic recurring experiences in the world that form very early in life [22, 23]. They act as pre-conceptual building blocks that we use to conceptualize objects and events on a high level of abstraction [23]. As image schemas derive from "the human natural understanding of the physical world" — which is something SoftVis metaphors commonly try to achieve [9] — they seem well suited as a foundation for designing and evaluating SoftVis metaphors. In addition, image schemas are in line with existing ideas of deriving visualization metaphors from "conceptual structures" [18] or "image-schemas" [19].

The recurrent co-activation of an image schema with a specific target domain results in an ISM [24]. These co-activations form through subjective experiences in the real world, where the target domain can be tangible (e.g., MORE is UP, LESS is DOWN) or abstract (e.g., HAPPY is UP, SAD is DOWN). Due to their ability to describe abstract concepts and events, image schemas are increasingly being utilized for abstract data comprehension. In computer science, e.g., image schemas are used to explain blockchain technology [25] or to describe complex events for artificial intelligence [26]. As image schemas are linked to literal linguistic expressions [22, 27], ISM can be extracted from the user's subconscious mental model by analyzing users' linguistic expressions [21, 28]. Previous work indicates that user interfaces that are in line with ISM — which were previously coded from the users' subconscious mental models — can facilitate the users' subconscious application of prior knowledge [21, 28]. This intuitive use then results "in an effective and satisfying interaction using a minimum of cognitive resources" [29]. As — to the best knowledge of the authors — no studies on ISM-based SoftVis exist, further research is needed to confirm similar positive effects for SoftVis. However, the importance of a SoftVis' compliance with the user's mental model for its ability to provide insights is suspected [20].

2.1. Method

Software developers (at different expertise levels) are the core target audience for SoftVis tools [30]. While other target groups exist, e.g., project managers [31, 32] or students [33], we focus on software developers that work with an object-oriented programming language. In this context, we code ISM based on a list of 47 image schemas from the categories *BASIC*, *SPACE*, *CONTAINMENT*, *MULTIPLICITY*, *PROCESS*, *FORCE*, and *ATTRIBUTE* described by Hurtienne et al. [21]. As designers generally do not strive to match technical properties but instead match the mental model of the user [34], our primary data source are software developer interviews. However, as software developers work closely with the technical implementation, we explore programming language documentation as an additional data source.

2.1.1. Software Developer Interviews

To determine the users' mental model of our target domain, we explore transcribed user interviews with software developers who perform software comprehension tasks. Following the coding process by Hurtienne et al. [21], contextual interviews are transcribed and ISM coded based on a list of image schemas. The identified ISM are then prioritized based on their frequency across multiple interviews and — for SoftVis designed for multiple object-oriented programming languages – their consistency across programming languages. Ambiguous ISM might further be validated by consolidated additional interviews, as described by Huber et al. [28].

To test if we can code ISM with the proposed method, we conducted contextual interviews with three english-speaking expert-level C# software developers. The interviews lasted ~ 10 minutes each. The developers performed a software comprehension task where they searched for a bug in the source code. The software developers were instructed to think-out-loud and the interviewer did not ask any questions. We coded ISM from the three interviews and removed all ISM that were not directly related to the software system or that only appeared in one interview. Hence, Table 1 shows the ISM that were present in at least two interviews.

Table 1
ISM in the context of software comprehension that were present in at least two interviews.

TARGET DOMAIN is SOURCE DOMAIN	Example Expression
C# is CONTAINER	*"... way of iterating collections in C#."*
EXECUTION SYSTEM is CONTAINER	*"... is using it in .NET."*
PROGRAM is CONTAINER	*"... problems in your application."*
PROGRAM CONTENT is on PATH	*"... passed down from the constructor."*
ACTIVE PROGRAM is MOMENTUM	*"Let's run this and see ... "*
SOURCE FILE is CONTAINER	*"In the file, all I do ... "*
CLASS is CONTAINER	*"In this class ... "*
EXECUTING METHOD is ENABLEMENT	*"... anyone can call this method."*
USED CODE is IN, UNUSED CODE is OUT	*"... comment out the benchmark."*

2.1.2. Language Documentation

The second data source is programming language documentation. As software developers work closely with the actual technical implementation, the documentation might give additional insights on the image-schematic structure of the target domain. As programming languages consists of specific concepts and naming schemes, we expect resulting image schemas to generally be in line with the ISM identified in the contextual interviews.

We analyzed the first chapter (eight pages) of the official Microsoft C# documentation *"A tour of the C# language"* [35]. Table 2 shows the identified ISM.

Table 2
ISM in the context of software comprehension that were present in the language documentation.

TARGET DOMAIN is SOURCE DOMAIN	Example Expression
C# is CONTAINER	*"Here it is in C#."*
EXECUTION SYSTEM is SURFACE	*"C# programs run on .NET ... "*
PROGRAM is CONTAINER	*"... at those points in the program."*
PROGRAM CONTENT is on PATH	*"The program starts with a ... "*
ACTIVE PROGRAM is MOMENTUM	*"C# programs run on .NET ... "*
SOURCE FILE is CONTAINER	*"... can be stored in several source files."*
NAMESPACE is CONTAINER	*" Namespaces contain types ... ".*
TYPE is CONTAINER	*" ... types, which contain members ... "*

3. Discussion

The described analysis seems to be capable of coding ISM from software developers' subconscious mental models. In relatively short interviews, we identified a number of ISM that were present across multiple interviews. However, we did not identify ISM for all target domains (e.g., Namespaces or Methods). This was probably due to the selected comprehension task and the short interview duration. We are confident that we will be able to collect more ISM with longer interviews and different comprehension tasks. As the first results suggest that most software components could be mapped to the *CONTAINER* image schema, we might adjust our methodology to focus more on ISM that describe relationships between the components. These ISM could play a fundamental role for designing SoftVis, as they seem to be one of the key distinctions between existing SoftVis metaphors. For example, the *CENTER-PERIPHERY* image schema might suggest something similar to the solar system metaphor while the *CONTACT* image schema might suggest something more similar to the city or island metaphor.

The proposed analysis also seemed capable of coding ISM from programming language documentation. First results indicate that ISM coded from documentation mostly match ISM coded from interviews. However, our small sample size already included one mismatching ISM. The virtual execution system .NET — which is C#-specific — has a *SURFACE* image schema in the documentation but a *CONTAINER* image schema in the interviews. While further ISM coding of the documentation is needed, this highlights potential benefits of SoftVis tools that are designed to match the mental model of the user [34]. ISM coded from documentation are probably more in line with the mental model of the language designer (i.e., *design model*), while the ISM coded from user interviews represent the *user model*, as described by Norman [36]. But ISM coded from language documentation might still facilitate design decisions, e.g., by resolving ambiguous ISM from user interviews that could not be resolved through additional interviews.

Once the list of ISM is more complete, it could help SoftVis designers to create visualization metaphors that are more in line with software developers' mental model. ISM could provide guidance on choosing intuitive visualization metaphors and could help to enhance existing SoftVis, e.g., by highlighting mismatching ISM. As SoftVis tools are generally used for mentally demanding and time consuming comprehension tasks, even small optimizations could have a significant impact on the user experience. However, further research is needed to measure benefits of a SoftVis' compliance with the user's mental model.

4. Future Work

Our future work includes further exploration of our two data sources with the proposed analysis. We will conduct more contextual interviews with software developers in different object-oriented programming languages and will continue to analyze programming language documentations. Additionally, we want to apply the identified ISM to SoftVis design. This includes creating novel ISM-based SoftVis metaphors and evaluating existing conceptual metaphors based on their compliance with the ISM of the target domain. Finally, we will evaluate ISM-based SoftVis tools in regards to intuitive use and their ability to provide insights compared to traditionally designed SoftVis.

References

[1] T. Ball, S. G. Eick, Software visualization in the large, Computer 29 (1996) 33–43.

[2] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, S. Li, Measuring program comprehension: A large-scale field study with professionals, IEEE Transactions on Software Engineering 44 (2017) 951–976.

[3] B. A. Price, R. M. Baecker, I. S. Small, A principled taxonomy of software visualization, Journal of Visual Languages & Computing 4 (1993) 211–266.

[4] L. von Kurnatowski, D. Heidrich, N. Güden, A. Schreiber, H. Polzin, C. Stangl, Analysing and visualizing large aerospace software systems, in: ASCEND 2021, 2021, p. 4082.

[5] M. Misiak, A. Schreiber, A. Fuhrmann, S. Zur, D. Seider, L. Nafeie, Islandviz: A tool for visualizing modular software systems in virtual reality, in: 2018 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2018, pp. 112–116.

[6] E. R. Gansner, S. C. North, An open graph visualization system and its applications to software engineering, Software: practice and experience 30 (2000) 1203–1233.

[7] D. Holten, Hierarchical edge bundles: Visualization of adjacency relations in hierarchical data, IEEE Transactions on visualization and computer graphics 12 (2006) 741–748.

[8] C. Knight, M. Munro, Virtual but visible software, in: 2000 IEEE Conference on Information Visualization. An International Conference on Computer Visualization and Graphics, IEEE, 2000, pp. 198–205.

[9] P. Caserta, O. Zendra, Visualization of the static aspects of software: A survey, IEEE transactions on visualization and computer graphics 17 (2010) 913–933.

[10] R. Wettel, M. Lanza, Visualizing software systems as cities, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE, 2007, pp. 92–99.

[11] H. Graham, H. Y. Yang, R. Berrigan, A solar system metaphor for 3d visualisation of object oriented software metrics, in: Proceedings of the 2004 Australasian symposium on Information Visualisation-Volume 35, 2004, pp. 53–59.

[12] A. Schreiber, M. Misiak, Visualizing software architectures in virtual reality with an island metaphor, in: International Conference on Virtual, Augmented and Mixed Reality, Springer, 2018, pp. 168–182.

[13] D. Atzberger, T. Cech, M. de La Haye, M. Söchting, W. Scheibel, D. Limberger, J. Döllner, Software forest: A visualization of semantic similarities in source code using a tree metaphor., in: VISIGRAPP (3: IVAPP), 2021, pp. 112–122.

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

[15] A. Hoff, L. Gerling, C. Seidl, Utilizing software architecture recovery to explore large-scale software systems in virtual reality, in: 2022 Working Conference on Software Visualization (VISSOFT), IEEE, 2022, pp. 119–130.

[16] D. Limberger, W. Scheibel, J. Döllner, M. Trapp, Visual variables and configuration of software maps, Journal of Visualization (2022) 1–26.

[17] K. Holtzblatt, J. B. Wendell, S. Wood, Rapid contextual design: a how-to guide to key techniques for user-centered design, Elsevier, 2004.

[18] A. P. Andreou, Conceptual metaphors as image schemas in information visualizations, in:

2CO Communicating complexity: 2013 Conference Proceedings, Edizioni Nuova Cultura, 2013, pp. 12–18.

[19] V. L. Averbukh, Approach to semiotic theory of computer visualization, Advances in Computer Science: An International Journal 4 (2015) 44–54.

[20] V. L. Averbukh, M. O. Bakhterev, D. V. Manakov, Evaluations of visualization metaphors and views in the context of execution traces and call graphs, Scientific Visualization 9 (2017) 1–18.

[21] J. Hurtienne, K. Klöckner, S. Diefenbach, C. Nass, A. Maier, Designing with image schemas: resolving the tension between innovation, inclusion and intuitive use, Interacting with Computers 27 (2015) 235–255.

[22] M. Johnson, The body in the mind: The bodily basis of meaning, imagination, and reason., University of Chicago Press, 1987.

[23] A. Blackler, J. Hurtienne, Towards a unified view of intuitive interaction: definitions, models and tools across the world, MMI-interaktiv 13 (2007) 36–54.

[24] J. Grady, Foundations of meaning: Primary metaphors and primary scenes (1997).

[25] I. E. Khairuddin, C. Sas, C. Speed, Blockit: A physical kit for materializing and designing for blockchain infrastructure, in: Proceedings of the 2019 on Designing Interactive Systems Conference, 2019, pp. 1449–1462.

[26] M. M. Hedblom, O. Kutz, R. Peñaloza, G. Guizzardi, Image schema combinations and complex events, KI-Künstliche Intelligenz 33 (2019) 279–291.

[27] J. M. Mandler, C. P. Cánovas, On defining image schemas, Language and cognition 6 (2014) 510–532.

[28] S. Huber, P. Schulz, E. Hauke, J. Hurtienne, Image schematic metaphors in air traffic controllers' language (2022).

[29] J. Hurtienne, Image schemas and design for intuitive use, Doctoral dissertation, Technische Universität Berlin (2011).

[30] L. Merino, M. Ghafari, C. Anslow, O. Nierstrasz, A systematic literature review of software visualization evaluation, Journal of systems and software 144 (2018) 165–180.

[31] A. Schreiber, M. Brüggemann, Interactive visualization of software components with virtual reality headsets, in: 2017 IEEE Working Conference on Software Visualization (VISSOFT), IEEE, 2017, pp. 119–123.

[32] M. Ogawa, K.-L. Ma, Stargate: A unified, interactive visualization of software projects, in: 2008 IEEE Pacific Visualization Symposium, IEEE, 2008, pp. 191–198.

[33] A. Al-Sakkaf, M. Omar, M. Ahmad, A systematic literature review of student engagement in software visualization: a theoretical perspective, Computer Science Education 29 (2019) 283–309.

[34] A. Cooper, R. Reimann, D. Cronin, C. Noessel, About face: the essentials of interaction design, John Wiley & Sons, 2014.

[35] Micosoft, A tour of the c# language, 2023. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

[36] D. A. Norman, Cognitive engineering, User centered system design 31 (1986) 2.