A Task Oriented View of Software Visualization

Jonathan I. Maletic, Andrian Marcus, Michael L. Collard
Department of Computer Science
Kent State University
Kent Ohio 44242
330 672 9039
jmaletic@cs.kent.edu, amarcus@cs.kent.edu, collard@cs.kent.edu

Abstract

A number of taxonomies to classify and categorize software visualization systems have been proposed in the past. Most notable are those presented by Price [1993] and Roman [1993]. While these taxonomies are an accurate representation of software visualization issues, they are somewhat skewed with respect to current research areas on software visualization. We revisit this important work and propose a number of realignments with respect to addressing the software engineering tasks of large-scale development and maintenance. We propose a framework to emphasize the general tasks of understanding and analysis during development and maintenance of large-scale software systems. Five dimensions relating to the what, where, how, who, and why of software visualization make up this framework. The focus of this work is not so much as to classify software visualization system, but to point out the need for matching the method with the task. Lastly, a number of software visualization systems are examined under our framework to highlight the particular problems each addresses.

1. Introduction

Software visualization represents many things to many people. Price presents the following general definition of software visualization:

> *"Software visualization is the use of the crafts of typography, graphic design, animation and cinematography with modern human-computer interaction and computer graphics technology to facilitate both the human understanding and effective use of computer software."* [18].

This definition subsumes such diverse topics as program visualization, algorithm animation, visual programming, programming by demonstration, data visualization, and source code browsers. This diversity is reflected in the taxonomic descriptions of the field by researchers such as Price [17, 18], Roman [22], Myers [15], and Stasko [24].

While each of these topics has interesting and important problems, the breadth induces many orthogonal features and issues. There is a need to focus the scope and highlight the current issues reflected in software engineering of today. Therefore, we develop our framework to emphasize the general tasks of understanding and analysis during development and maintenance of large-scale software systems. This said we exclude a discussion of topics such as algorithm animation and visual programming, as these are tangential to this perspective. As our framework is presented, the reasoning of this decision will become more apparent.

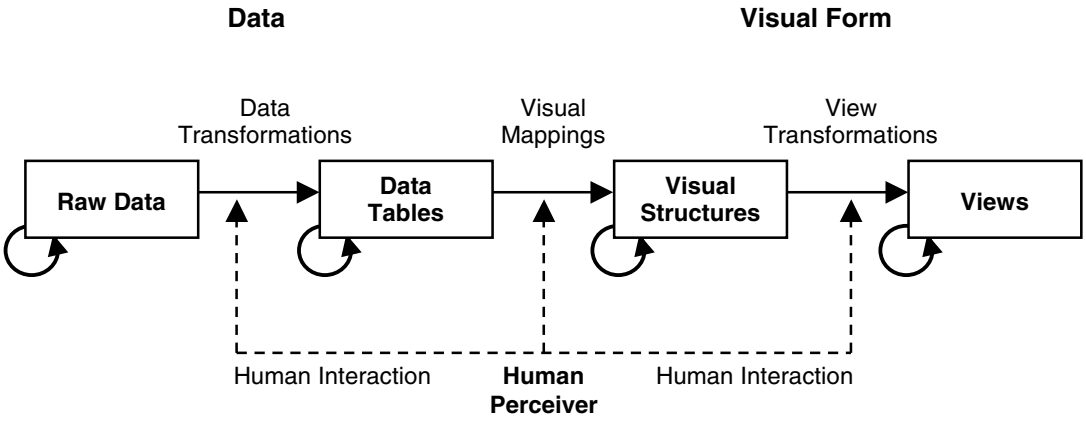
Additionally, we argue that no single software visualization tool can address all software engineering tasks simultaneously. While this may be obvious, taxonomies often highlight the lack of functionality in a tool rather then focusing on its strength in addressing a particular problem.

Large-scale software maintenance and development involve a variety of application tasks. These tasks range from coding and debugging, to design and re-engineering. The underlying theme is that most development and maintenance tasks require a level of understanding the associated software system and documentation. This is the promise of visualization tools – that they assist the user in (better) understanding some aspect of the software. This could range from uncovering bottlenecks in execution data or identifying poor architecture or design. These two problems are quite orthogonal with respect to the types of understanding necessary for problem solving.

These different software engineering tasks should be addressed by different visual representations. That is, we should use the most appropriate visualization mechanism for the given task.

Before we define our framework, let us step back and adopt a reference model for visualization. This reference model will lay the foundation of our framework and more formally tie software visualization research with the more general research on information visualization.

Proceedings of the First International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT'02)
0-7695-1662-9/02 $17.00 © 2002 IEEE



Raw Data: idiosyncratic formats
Data Tables: relations (cases by variables) + meta data
Visual Structures: spatial substrates + marks + graphical properties
Views: graphical parameters (position, scaling, clipping, etc.)

Figure 1. Reference Model for Visualization. Visualization can be described as a mapping of data to visual form that supports human interaction for making visual sense [3].

2. A Reference Model for Visualization

Card [3] proposes that visualization is a mapping from data to a visual form that the human perceives. Figure 1, adapted from [3], describes these mappings and serves as a simple reference model for visualization. In this figure, the flow of data goes through a series of transformations. The human may adjust these transformations, via user controls, to address the particular application task.

The first transformation converts raw data into more usable data tables. The raw data is typically in some domain specific format that is often hard, or impossible, to work with. This is very apparent when working with trace data generated from program executions. Data tables [3] are relational depictions of this data. Information about the relational characteristics of the data (meta data) can also be included in the data tables. Meta data is descriptive information about the data [27]. From here, visual mappings transform the data tables into visual structures (graphical elements). Finally, the view transformations create views of the visual structures by specifying parameters such as position, rotation, scaling, etc. User interaction controls the parameters of these transformations. The visualizations and their controls are all with respect to the application task.

The core of the reference model is the mapping of a data table to a visual structure. Data tables are based on mathematical relationships whereas visual structures are based on graphical properties processed by human vision. Although raw data can be viewed directly, data tables are a vital intermediate step when the data is abstract [4, 11, 19].

Software visualization maps to this reference model directly. The raw data is source code, execution data, design documents, etc. In the case of execution (trace) data, the readability is minimal. However, source code is readable, at least on a small scale, that is, one can hardly keep in mind more the a few dozen lines of source at one time. Data tables, an abstraction of the raw data, take the form of abstract syntax trees, program dependence graphs, or class/object relationships for example. A variety of software analysis tools can generate this type of data (table). Visual structures are then the software-specific visualizations we render. These visual structures are typically very specific to a particular software engineering task.

This model also points out the need to transform raw data into something more usable. This includes initial acquisition, quality, and granularity of the data. While these issues are not high profile for source code, they are a key component for dealing with the huge amounts of data that can be generated from execution traces, or from parse trees of large systems.

The software visualization process maps on top of this reference visualization model. Roman [22] and Price [17, 18], each define their own general model of the software visualization. Their views are more domain specific and omit aspects related to generation of views and data transformations. These models drive the definition of their respective taxonomies. We believe, the general information visualization reference model should

also be taken into direct consideration by a software visualization system designer.

We now describe the dimensions used to characterize software visualization.

3. Dimensions of Software Visualization

As mentioned previously, our focus is to describe software visualization systems in light of their applications toward supporting large-scale software development and maintenance. In order to accomplish this task we define five dimensions of software visualization. These dimensions reflect the why, who, what, where and, how of the software visualization. The dimensions are as follows:

- Tasks – *why* is the visualization needed?
- Audience – *who* will use the visualization?
- Target – *what* is the data source to represent?
- Representation – *how* to represent it?
- Medium – *where* to represent the visualization?

These dimensions define a framework capable of accommodating a large spectrum of software visualization systems, including topics outside the scope of this work (e.g., algorithm animation and visual programming tools).

Table 1 presents how these five dimension map to the attributes proposed by Roman and Price in their respective taxonomies. Both these taxonomies describe the attributes of software visualization; the difference here is what we choose to emphasis as the most important aspects.

We now describe these dimensions in detail after which we present examples of select visualization systems and map them along our dimensions. The majority of these systems predate the previously mentioned taxonomies.

3.1. Tasks

The *task* dimension defines why the visualization is needed. In other words, it specifies what particular software engineering task(s) are supported by the software visualization system. In general, every visualization system supports understanding of one or more aspects of a software system. This understanding process will in turn support a particular engineering task. Early visualization tools (e.g., algorithm animation, data flow, etc.) were aimed at supporting understanding for education purposes. Visual programming systems support domain specific programming. Many of today's software visualization tools support software engineering tasks for large software systems. These tasks include: development activities (e.g., programming [2], debugging [1], testing [10] [5], etc.), maintenance [7] (e.g., fault detection [8, 10], re-engineering, reverse engineering [16], etc.), software process management, and marketing. Based on these specific tasks, the user needs to obtain different levels, or types, of understanding of the software. Consequently, the developer will require different visualization tools, each with its own specific goals.

This aspect of software visualization is not covered by Roman's taxonomy and is marginally addressed within the taxonomy proposed by Price et al (as seen table 1). This is due in great part to the state of the art of the field nearly a decade ago.

In our view, this dimension is the driving force behind defining a classification of software visualization systems. If such a system does not support the engineering tasks on the user's agenda, the other features are of no importance. In addition, as discussed earlier, different software engineering task require visualizations with different characteristics. These characteristics are then later defined along the other dimensions, with respect to the supported engineering task.

With respect to the reference model described in section 2, the task dictates the type of views and visual

Table 1. Overview of the relations between the proposed dimensions and the criteria defined in the taxonomies of Roman [22] and Price et all [17,18] respectively.

Dimension	Roman [Roman'93]	Price et all. [Price'93, '98]
Task	***	F.1: Purpose
Audience	***	F.1: Purpose
Target	Scope Abstraction	A: Scope B: Content
Representation	Specification method Interface Presentation	C: Form D: Method E: Interaction F: Effectiveness
Medium	***	C: Form

structures. The same underlying data (or data tables) can be used to produce task specific visualizations with which the user interacts.

3.2. Audience

Based on the supported task, the software visualization tool may be geared towards different types of users. The *audience* dimension defines the attributes of the users of the visualization system. If the primary supported task is education, students and/or instructors form the audience. In the industrial setting (which is of keen interest here), the audience will be developers, maintainers, testing personnel, and/or software process or team managers. In addition, different tools can be tailored towards users with different skills (e.g., experienced versus beginners or developer versus manager). In general, an experienced developer will have drastically different information needs than a novice or new team member. Also, the amount of training necessary to use a visualization tool comes into play. There are two schools of thought here, one is that a tool should be simple to use, however this typically limits the functionality of the tool. The other alternative is to require users to be trained to use the system properly. We believe that if a developer takes the time to be trained in a programming language/environment than spending time to learn about the visualization tool is a reasonable request.

The audience aspect is omitted in Roman's taxonomy, while the taxonomy proposed by Price et al combines it into one common attribute (i.e., Purpose) with the supported engineering task (see table 1). This is a matter of priorities rather then a fault in the taxonomies.

With respect to the reference model, the users role is obvious. However, note the user interacts with the visualization system via sets of parameters. These parameters dictate the views and thus support the particular task for the type of user.

3.3. Target

The *target* of a software visualization system defines what (low level) aspects of the software are visualized. The target is a work product, artifact, or part of the environment of the software system. It is the source of the data (raw data in the reference model). Meta data and data tables represent the underlying meaning of the data along with translations of the data (e.g., the abstract syntax tree). Along this dimension, we are considering as targets of visualization the architecture, the design, the algorithm, the source code, the data, execution/trace information, measurements and metrics, documentation, and process information.

The simplest visualization systems aim to represent the source code in a more readable and easier to understand form for the user. Pretty printers, integrated development environments are examples of tools that perform these types of visualizations. Notable here is SeeSoft [8]; a visualization tool that is specialized in representing source code of large software systems. Flow-charts are probably the oldest form of visualizing algorithmic type information. Other systems are concerned with visualizing execution information BeeHive [20], Jinsight [6] that can be captured real time or as a whole and examined after the execution. These types of tools are usually geared to support testing, optimizations, etc.

This dimension corresponds to the Scope and Abstraction criteria defined by Roman, and the Scope and Content criteria defined by Price (as seen in table 1). Both taxonomies do an excellent job covering these aspects in detail, thus it is not necessary to further elaborate on this dimension. An important issue with those taxonomies is the omission of architecture and design level information from these categories. Once again, this is explained by the fact that at the time software visualization systems were, for the most part, targeted toward small programs, rather than large-scale software systems. Tools that target design and/or architecture level information usually support program understanding with reverse engineering or reengineering in mind: IMSOvison [13], SoftArch [9], SHriMP [25], etc.

Other types of target source data are measurements and metrics obtained from software, process information, and documentation. Visualization of this type of information can support the software process and team management activities.

At a more detailed level, this dimension includes attributes relating to issues such as data collection (i.e., time of collection, method of collection, invasiveness, etc.) and issues relating to the programming language and environments (e.g., paradigm, concurrency, parallel processing, etc.). These criteria are well covered in the taxonomy proposed by Price et al and map directly onto this dimension in our framework.

Finally, a very important aspect of the target is the scalability issue. Some systems can only visualize a small amount of information but, in general, engineering tasks require the ability to visualize (very) large amounts of data, considering that real-life software systems have millions of lines of code and can generate gigabytes of trace information. To represent such large amounts of data special mediums and representation techniques need to be utilized.

3.4. Representation

Depending on the goals and target of the software visualization system, the type of users and available medium, a form of *representation* needs to be defined to best convey the target information to the user. This dimension defines how the visualization is constructed based on the available information. The representation manifests itself as the visual structures in the reference model. In designing a software visualization system, this is one of the more important elements. We look to the research in information visualization and cognitive sciences [12, 21, 26, 28-30] to make the best choices in designing software visualization systems. This research centers on methods to best map raw data into a visual structure and view.

MacKinlay [12] defined two criteria to evaluate the mapping of data to a visual metaphor: expressiveness and effectiveness. These criteria were used in 2D mappings, but can also be applied for 3D mappings. Expressiveness refers to the capability of the metaphor to visually represent all the information we desire to visualize. For instance, if the number of visual parameters available in the metaphor for displaying information is fewer than the number of data values we wish to visualize, the metaphor will not be able to meet the expressiveness criterion.

The relationship between data values and visual parameters has to be a univocal relationship; otherwise, if more than one data value is mapped onto the same visual parameter then it will be impossible to distinguish one value's influence from the other. On the other hand, there can always be visual parameters that are not used to map information, as long as there is no need for them to be utilized.

The second criterion, effectiveness, relates to the efficacy of the metaphor as a means of representing the information. Along the effectiveness dimension we can further distinguish several criteria: effectiveness regarding the information passing as visually perceived, regarding aesthetic concerns, regarding optimization (e.g., number of polygons needed to render the world).

In the case of quantitative data (e.g., software metrics, LOC, trace data), not only the number of visual parameters has to be sufficient to map all the data, but also, they must be able to map the right data. There are visual parameters that are not able to map a specific category of data; for instance, shape is not useful for mapping quantitative data, while the size of a metaphor is.

Effectiveness implies the categorization of the visual parameters according to its capabilities of encoding the different types of information. Moreover, this also implies categorizing the information according to its importance so that information that is more important can be encoded more efficiently when options must be taken. This categorization of the importance of the information has two expressions: one is an assigned importance of the information in the context of a software system; the other is a preference of the user. Nonetheless, the user may choose to override this and define his own importance of the data, according to his priorities when visualizing a software system. For example, one could give preference in a visualization to the public members of an object-oriented class, versus the private ones.

In order to satisfy these criteria for the mapping, one must have a solid data characterization. Data characterization is usually the first step to understand a phenomenon or system. Developing a taxonomy helps to make sense of large amounts of information. Although these characteristics of data apply mostly to data visualization, they must be taken into consideration in software visualization as well. The metaphors should be designed such that they maximize the amount of data that can be represented with an accent on the user's information seeking goals.

The power of a visualization language is derived from its semantic richness, simplicity, and level of abstraction. The aim is to develop a language with few metaphors and constructs, but with the ability to represent a variety of elements with no ambiguity or loss of meaning. In addition, the visualization has to maximally use the potential of the used media. For example, a good VR representation will make use of all the navigation possibilities in a 3D landscape and the fact that the user is immersed in the environment, while maintaining a natural feeling of the representation, and avoiding the information overload.

As mentioned, an important aspect to be considered in defining a visual representation is the nature of its users. One may design a representation for use by software developers with solid knowledge of programming, program designs, and system architecture. The metaphors in the representation should be simple, having a familiar form and straightforward mapping to the target.

With all these considerations in mind, the representation can take several forms (e.g., source code, tables, diagrams, charts, visual metaphors – icons, figures, images, worlds, etc.) and have various attributes (e.g., interactive, static, dynamic, on-line or off-line views, multiple views, drill-down capabilities, multiple abstraction levels, etc.). Once again, these elements and attributes need to be defined and designed with several goals in mind, to support the needs of the user.

Shneiderman [23], presents seven high level user needs that an information visualization application should support. For evaluation purposes, we must refine these into lower-level tasks as done by Wiss, Carr, and Jonsson [32]. The needs are presented below and should act as a guideline for developing navigational needs of the user:

Overview: Gain an overview of the entire collection of data that is represented. This is often a difficult problem in the case of visualizing the structural information of large systems. Constructing good visualizations of large connected graphs is an open research area.

Zoom: Zoom in on items of interest. When zooming, it is important that global context can be retained. This subsumes methods to drill down to lower levels of abstraction.

Filter: Filter out uninteresting items. Filtering by removing parts of the visualization will necessarily disturb the global context. Therefore, it is important to see whether the design supports some kind of abstraction of the removed parts.

Details-on-demand: Select an item or group and get details when needed. Getting details on a selected item is usually implemented by the embedding application. The detail representation is of less importance in large-scale software visualization therefore, priority will be given to easy and fast navigation and rendering. The visual metaphors are designed such that there is no loss of meaning while zooming in or out.

Relate: View relationships among items. For a hierarchical data structure, it is necessary that the visualization show parent-child relationships. This is one of the most important features of many software visualization systems. Software systems rely on many inter-related components, working together to solve problems.

History: Keep a history of actions to support undo, replay, and progressive refinement. A visitation path should be supported. That is a set of attributes, which describe the position of the camera, the light, and the zoom level. These viewpoints can be saved and reviewed. A sequence of such viewpoints can be played, thus representing a path within the visualization, which could represent the history.

Extract: Allow extraction of sub-collections and of query parameters. This task concerns saving the current state of the visualization. This is related only to the application and the underlying data set. How the data is visualized does not affect this.

In addition to these criteria, Roman and Price's taxonomies offer a detailed classification of the forms, methods, interfaces, interactions, and effectiveness of the visual representation (see table 1).

3.5. Medium

The *medium* is where the visualization is rendered. That is, the display medium. The ones generally used by software visualization systems today are: paper and (colored) pencil, black and white monitors, color monitors (21 inch), multiple monitors (2*21 inch), and high-resolution/large sized displays (e.g., plasma screens and projectors). Other mediums being investigated for use by software visualization systems are: stereo displays, immersive virtual reality environments, and multi-typed mediums (e.g., a laptop in an VR environment). Every and each of these mediums have different characteristics and in consequence are suited for different tasks. For example, paper and low-end monitors are well suited for small-scale, low-dimension, static representations, while virtual immersive environments offer expansive real estate for visualizations of large structures (e.g., connected graphs) and the ability to make use of other sensorial inputs (e.g., sound, smell, motion, haptics, etc.).

Most taxonomies of software visualization do not address the aspect of mediums. This is a wholly information visualization issue. The development of new mediums is driven by user needs and the market place. The recent advances in technology and continued reduction in cost of these technologies give rise to new mediums and thus new opportunities for better representations.

The medium is not explicit in the reference model but it is an implicit concept. The user must interact and perceive the visualization from some technology. We see this as an important aspect for software visualization in the future. The information visualization community has been taking advantage of mediums for quite some time and this research is starting to flow to the realm of software visualization.

4. Mapping Software Visualization Systems

This section presents a number of software visualization tools and systems that have very different features along the defined dimensions of our framework. Most notably they have different targets, some use different mediums, and they support different tasks. For the most part, these tools were developed after the publication of the previous taxonomies. Table 2 presents a summary of their attributes over the five dimensions of our framework.

SeeSoft [8] is a tool for visualizing software statistics about lines of code. It uses a thin colored line to represent each line of code in a file. The color (and brightness) of the line is calculated from the statistics that the tool has about that line. The indentation of the code can also be preserved. Since each line of code is shown as a single line of pixels it is capable of representing more than 50,000 lines of code one screen. Sections of code can be selected from the lines and viewed in a reading window. The visualization is interactive allowing for zoom actions, selection, and filtering. It is a versatile and general tool that can support a variety of tasks. On top of the source line information colors can encode various data such as testing data, data types, version control, etc. It is suited

for analyzing reasonably large software systems but without accommodating design or architecture level information. It has been used to analyze version control data, feature location in source code, etc.

SHriMP (Simple Hierarchical Multi-Perspective) [25] allows views of hierarchical software structures showing many levels from the actual source to classes and package views. It has been customized for browsing Java programs. The hierarchies are represented using nested graphs. The views allow zooming of various kinds with hypertext browsing over nested graphs. SHriMP is one of the few software visualization tools that is able to visualize aspects of design level information, it allows multiple views of the software system, and has drill-down capabilities from class hierarchy to source code and back, as well as documentation representation. The visualization is highly interactive, offering a multitude of feature to the user. It is useful in understanding activities for reverse engineering. Additionally, its underlying diagrammatic visual representation can represent data from other sources than software.

Tarantula [10] is a tool for fault location in source code. The information about defects and test suites is display with color and brightness. The results of the multiple tests are used to determine the color and brightness for each line of code. The user can quickly spot which lines of code were executed when test cases failed and gauge how much a specific line of code was responsible for the error. The representation is based on SeeSoft [8].

IMSOvision (IMmersive SOftware VISualizatION) [14] is a system that supports program understanding and development through software visualization. It uses a virtual environment as the medium for visualization and uses a specially designed visualization language that maps source code into the virtual environment. This language, LOOC (Language for OO software Comprehension), incorporates some of the features of UML and allows for a natural representation of certain source code level complexity metrics. LOOC maps heterogeneous data (classes, entities, relationships, and quantitative information) to the visual metaphors. Metric information is also incorporated into the visualization. Just like SHriMP, IMSOvision is suited to support understanding

Table 2. Mapping of the five software visualization systems along the five dimensions of the framework. Not all features along each dimension are represented.

Dimension / SV System	Task	Audience	Target	Representation	Medium
SHriMP	Reverse engineering, maintenance	Expert developer	Source code, documentation, static design-level information, medium Java systems	2D graphs, interactive, drill-down	Color monitor
Tarantula	Testing, defect location	Expert developer	Source code, test suite data, error location	Line oriented representation, color, interactive, filtering, selection	Color monitor
IMSOvison	Development, reverse engineering, management	Expert developer, team manager	Source code, static design information, metrics, large OO systems	Specialized visual language, 3D color objects, spatial relationships, drill-down, interactive, abstraction mechanism	Immersive virtual environment
SeeSoft	Fault location, maintenance, reengineering	Expert developer	Source code, execution data, historical data	Line oriented representation, color, interactive, filtering, selection	Color monitor
Jinsight	Optimization	Expert developer	Program bursts, Java, dynamically collected	Color coded line oriented, text, interactive, filtering, queries	Color monitor

activities related to reverse engineering, but also to development, as both are able to represent aspects of design-level information.

JInsight [6] is a tool for the analysis of running, multithreaded Java program behavior. It allows the user to deal with a large amount of trace information by carefully selecting what and when they want the information collected. The information collected is associated with a particular running task. It can be connected to a running program, collect the data, and then disconnect, thereby starting the process all over. This tool is an example that visualizes data about the software, rather than the structure of the software.

As seen in table 2, these tools cover a broad spectrum over the five dimensions of our framework. The classification and description of the systems is by no means complete. Many of the detailed attributes that were covered by previous taxonomies are left out on purpose. The goal of these examples is to highlight some issues that motivated our revisiting the issues of taxonomies for software visualization systems.

Each tool is built to support different tasks, the choice of features being motivated by this issue. We especially emphasize the fact that some of these tools are able to represent the structure of the software (e.g., SHriMP, IMSOvision), others represent data about the software execution (e.g, JInsight, Tarantula), the actual source code layout (e.g., SeeSoft, Tarantula), or multiple types of information (e.g., IMSOvision, SeeSoft). Due to the different nature of the representation, different mediums are chosen (e.g., VE, color monitors). Another important issue to note is that while some tools use general visual representations (e.g., timelines, graphs – in Jinsight and SHriMP), others define special visual metaphors (e.g., 3D objects, colored pixel-based lines – in IMSovision and SeeSoft).

5. Conclusions and future work

This paper revisits the issues of defining a taxonomy of software visualization systems. By realigning these taxonomies with the perspective of current software engineering problems we identify open research issues and how research in cognitive psychology and information visualization can aid the field of software visualization. Additionally, the framework we present highlights the strengths of individual tools and techniques with respect to their application to software engineering tasks.

This framework also puts into perspective the work being done on visualizing execution trace data with that of visualizing design and structural aspects of software. These visualization approaches are quite different because they address very different problems. The target of these systems is from different sources, one being large amounts of numerical data, the other source code. One approach uses specialized data visualization methods; the other must develop graph based and metaphoric visual representations.

The ultimate goal of this work is to iterate the key tasks for maintenance and development and determine the sets of dimensional values that are most appropriate. This would present us with a space of possible visualization systems with respect to software engineering tasks. As new visualization methods, mechanisms, and mediums become available, this ontology can serve as means to determine were they could be of use to software visualization.

References

[1] Baecker, R., DiGiano, C., and Marcus, A., "Software Visualization for Debugging", *Communications of the ACM*, vol. 40, no. 4, April 1997, pp. 44-54.

[2] Burd, E. L., Chan, P. S., Duncan, I. M. M., Munro, M., and Young, P., "Improving Visual Representations of Code", University of Durham, Technical Report 1996.

[3] Card, S. K., Mackinlay, J., and Shneiderman, B., *Readings in Information Visualization Using Vision to Think*, San Francisco, CA, Morgan Kaufmann, 1999.

[4] Chi, E. H., Barry, P., Riedl, J. T., and Konstan, J., "A spreadsheet approach to information visualization", in Proceedings of Information Visualization Symposium '97, 1997, pp. 17-24,116.

[5] De Pauw, W., Kimelman, D., and Vlissides, J., "Visualizing Object-Oriented Software Execution", in *Software Visualization*, Stasko, J., Dominque, J., Brown, M., and Price, B., Eds., Cambridge, MA MIT Press, 1998, pp. 329-346.

[6] De Pauw, W., Mitchell, N., Robillard, M., Sevitsky, G., and Srinivasan, H., "Drive-by Analysis of Running Programs", in Proceedings of ICSE 2001 Workshop on Software Visualization, Toronto, Ontario, Canada, 2001, pp. 17-22.

[7] Eick, S., "Maintenance of Large Systems", in *Software Visualization*, Stasko, J., Dominque, J., Brown, M., and Price, B., Eds., London MIT Press, 1998, pp. 315-328.

[8] Eick, S., Steffen, J. L., and Summer, E. E., "Seesoft - A Tool For Visualizing Line Oriented Software Statistics", *IEEE Transactions on Software Engineering*, vol. 18, no. 11, November 1992, pp. 957-968.

[9] Grundy, J. and Hosking, J. G., "High-level Static and Dynamic Visualisation of Software Architectures", in Proceedings of IEEE Symposium on Visual Languages (VL '00), Seattle, Washington, September, 10-14 2000.

[10] Jones, J. A., Harrold, M. J., and Stasko, J. T., "Visualization for Fault Localization", in Proceedings of ICSE 2001 Workshop on Software Visualization, Toronto, Ontario, Canada, 2001, pp. 71-75.

[11] Levoy, M., "Spreadsheet for images", *Computer Graphics*, vol. 28, 1994, pp. 139-146.

[12] MacKinlay, J. D., "Automating the design of graphical presentation of relational information", *ACM Transaction on Graphics*, vol. 5, no. 2, April 1986, pp. 110-141.

[13] Maletic, J. I., Leigh, J., and Marcus, A., "Visualizing Software in an Immersive Virtual Reality Environment", in Proceedings of ICSE'01 Workshop on Software Visualization, Toronto, Ontario, Canada, May 12-13 2001, pp. 49-54.

[14] Maletic, J. I., Leigh, J., Marcus, A., and Dunlap, G., "Visualizing Object Oriented Software in Virtual Reality", in Proceedings of International Workshop on Program Comprehension (IWPC01), Toronto, Canada, May 21-13 2001, pp. 26-35.

[15] Myers, B. A., "Taxonomies of Visual Programming and Program Visualization", *Journal of Visual Languages and Computing*, vol. 1, no. 1, March 1990, pp. 97-123.

[16] Parker, G., Franck, G., and Ware, C., "Visualition of Large Nested Graphs in 3D: Navigation and Interaction", *Journal of Visual Languages and Computing*, vol. 9, 1998, pp. 299-317.

[17] Price, B. A., Baecker, R. M., and Small, I. S., "A Principled Taxonomy of Software Visualization", *Journal of Visual Languages and Computing*, vol. 4, no. 2, 1993, pp. 211-266.

[18] Price, B. A., Baecker, R. M., and Small, I. S., "An Introduction to Software Visualization", in *Software Visualization*, Stasko, J., Dominque, J., Brown, M., and Price, B., Eds., London, England MIT Press, 1998, pp. 4-26.

[19] Rao, R. and Card, S. K., "Exploring large tables with the table lens", in Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'95), 1995, pp. 403-404.

[20] Reiss, S. P., "Bee/Hive: A Software Visualization Back End", in Proceedings of ICSE 2001 Workshop on Software Visualization, Toronto, Ontario, Canada, 2001, pp. 44-48.

[21] Roberts, J., "Display Models for Visualization", in Proceedings of International Conference on Information Visualization (IV'99), London, England, 1999.

[22] Roman, G.-C. and Cox, K. C., "A Taxonomy of Program Visualization Systems", *IEEE Computer*, vol. 26, no. 12, December 1993, pp. 11-24.

[23] Shneiderman, B., "The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations", in Proceedings of IEEE Visual Languages, 1996, pp. 336-343.

[24] Stasko, J. T. and Patterson, C., "Understanding and Characterizing Software Visualization Systems", in Proceedings of IEEE Workshop on Visual Languages, Seattle, WA, September 1992, pp. 3-10.

[25] Storey, M.-A. D., Best, C., and Michaud, J., "SHriMP Views: An Interactive Environment for Exploring Java Programs", in Proceedings of International Workshop on Program Comprehension (IWPC'01), Toronto, Ontario, Canada, May 12-13 2001, pp. 111-112.

[26] Tufte, E. R., *The Visual Display of Quantitative Information*, Graphic Press, 1983.

[27] Tweedie, L., "Characterizing interactive externalizations", in Proceedings of Conference on Human Factors in Computing Systems (CHI '97), 1997, pp. 375-382.

[28] Ware, C., *Information Visualization. Perception for Design*, Morgan Kaufmann Publishers, 2000.

[29] Ware, C. and Franck, G., "Evaluating stereo and motion cues for visualizing information nets in three dimensions", *ACM Transaction on Graphics*, vol. 15, no. 2, April 1996, pp. 121-140.

[30] Ware, C., Gobrecht, C., and Paton, M., "Dynamic adjustment of stereo display parameters", *IEEE Transactions on Systems, Man and Cybernetics*, vol. 28, no. 1, 1998, pp. 56-65.

[31] Wise, J. A., Thomas, J. J., Pennock, K., Lantrip, D., Pottier, M., Schur, A., and Crow, V., "Visualizing the non-visual: Spatial analysis and interaction with information from text documents", in Proceedings of Information Visualization Symposium (InfoVis'95), 1995, pp. 51-58.

[32] Wiss, U., Carr, D., and Jonsson, H., "Evaluating Three-Dimensional Information Visualization Designs A Case Study of Three Designs", in Proceedings of International Conference on Information Visualisation, London, England, July 29-31 1998.