Software Visualization

20.05. - 25.05.2001

organized by

Stephan Diehl (Saarbrücken)
Peter Eades (Sydney)
John Stasko (Atlanta)

It is often said that humans have never before created any artifacts which are as complex as todays software systems. As a result creating, maintaining, understanding and teaching software is a challenging task. Software is neither matter nor energy, it is just a kind of information. Sometimes the representation and the information itself are confused. Software visualization is concerned with visually representing different aspects of software including its structure, execution and evolution.
So far, research on software visualization was mostly motivated by its potential to support teaching. Many systems have been developed to facilitate the production of algorithm animations.
At Dagstuhl software enginieers and re-engineers repeatedly argued that there is a strong need for software visualization in their areas. Here further research includes the use of techniques from information visualization to display sofware metrics, graph layout and graph animations to show the structure and changes in software systems and program animation for debugging.
At the seminar more than 50 researchers from all around the world discussed the state-of-the-art as well as challenging questions for the future of software visualization. The program included 38 presentations and 15 system demonstrations, as well as several sessions for group discussions.
Participants of the seminar volunteered

- to compile a post seminar proceedings, which is to be published as a Springer LNCS state-of-the-art survey.
- to create a repository with algorithm animations and software visualization tools
- to initiate an international conference series on software visualization.

We feel that the seminar was a seminal event. The future will tell whether it reached its ambitious goals to form a community and raise awareness of software visualization as a challenging and important research field of its own.

Contents

1 Visualizing Software Changes

Stephen G. Eick
Visual Insights, USA

A fundamental problem in software engineering is changing the code, to add new functionality, support new hardware, accommodate new operating environments, and fulfill new increased user expectations. In an ideal world, software architecture would anticipate and facilitate future changes. In reality, the architecture is imperfect, and incorporate compromises forced by time and cost constraints. In real-world systems the scale and complexity become daunting. Software visualization is a natural, effective, and perhaps essential way to help programs overcome this complexity. We have developed a number of tools that facilitate rapid exploration of software change data, provide access to details, and support the maintenance and evolution of software systems.

2 Visualization for End-User Software Engineering

Margaret Burnett
Oregon State University, USA

Tools and environments to enable end users to "program" are becoming increasingly popular. The best known such environment is the spreadsheet, and the way users program in this type of environment is by providing formulas. Unfortunately, it has become clear that end users' programs are no more reliable than those written by professional software engineers. To try to help address this problem, we are developing a holistic approach to software engineering for end users. It incorporates support for testing, finding bugs, maintenance, and requirements specification. The software engineering knowledge needed is in the system, and the user is not expected to develop expertise at software engineering; instead, the strategy is for the system to provide guidance to the user. The way this is done is through an interactive visualization of the software engineering attributes of the spreadsheet, tightly intertwined with the spreadsheet display of cells, values, etc. Empirical studies show that with this approach, end users' ability to test and debug their spreadsheets is greater than has been true without the approach.

3 Structure in Sources, Structure in Systems and Structure in Between

Siegfried Wendt
Hasso-Plattner-Institut für Softwaresystemtechnik, Germany

It is one of the major tasks of engineers to spread the knowledge concerning the design of complex systems. In civil, mechanical and electrical engineering, optimal representations of such knowledge have been developed in a long evolutionary process. In software engineering, it is much more difficult to find adequate representations because of the abstract nature of software systems. The software itself should not be treated as the primary object to be described, but as a description which is the result of a mapping. The primary object to be described is the software controlled hardware. This system is dynamic, discrete, directed and informational, and it can be described using bipartite graphs for representing structures, ranges and processes. The software elements are obtained at the end of a sequence of mappings of system models.

4 Fun with Leonardo

Camil Demetrescu
Universita di Roma "La Sapienza", Italy

Joint work with:
Irene Finocchi, Universita di Roma "La Sapienza", Italy

Leonardo is an integrated environment for developing, executing and animating C programs. It provides two major improvements over a traditional IDE. First, it provides a mechanism for visualizing computations graphically as they happen by attaching in a declarative style graphical representations to key variables in a program. Second, code written with Leonardo can be executed both forwards and backwards, i.e., it is completely reversible: variable assignments will be undone, output sent to the console will disappear, graphics drawn will be undrawn, and so on. You have access to the full set of standard ANSI functions, and those are reversible too. Leonardo has been widely distributed over the Web (more than 15000 downloads over the last two years) and features a repository of more than 60 animated algorithms.

In this talk we survey the main features of Leonardo and we show examples of how to bind pictures to C code. In particular, we build in Leonardo a complete animation of the breadth-first visit of a graph starting from a plain C code.

URL: `http://www.dis.uniroma1.it/~demetres/Leonardo/`

5 Understanding Algorithms by means of Path Testing

Jorma Tarhio
Helsinki University of Technology, Finland

Joint work with:
Ari Korhonen, Helsinki University of Technology, Finland
Erkki Sutinen, University of Joensuu, Finland
University of Linköping, Sweden

Visualization of an algorithm offers only a rough picture of operations. Explanations are crucial for deeper understanding, because they help the viewer to associate the visualization with the real meaning of each detail. We present a framework based on path testing for associating explanations with a self-study visualization of an algorithm.
The algorithm is divided into blocks, and the system provides a description for each block. The system contains a separate window for code, flowchart, animation, explanations, and control. Students are given assignments based on the flowchart and coverage conditions of path testing. Path testing leads more exact evaluation of learning outcomes because of systematic instruction instead of common trial-and-error heuristics

6 Algorithm Animation for Constrained Domains

Ayellet Tal
Technion, Department of Electrical Engineering, Israel

A major goal in the design of an algorithm animation system is how to create a system that let others use it easily. One possible way to do it, the one proposed here, is to limit the domain the system supports. By constraining the domain we are able to incorporate into the system Knowledge regarding the objects and the operations which are prevalent in the domain. Built in the system are ways to visualize these objects and ways to animate the operations on them. As a result, large parts of the user's tasks can be automated.
A system for a constrained domain allows the user to be isolated from any concern about how graphics is being done. A typical animation can then be produced in a matter of days or even hours. This can be done regardless of the complexity of the algorithm being visualized. Even highly complex algorithms can be animated with ease.
We define a hierarchy of users: naive programmers, advanced programmers, end users, and groups of end users. The naive programmer cares solely about the contents of the visualization. Advanced programmers can also modify and extend various visualization aspects of the animation. End users experiment with an algorithm to understand its functioning. Finally, groups of end users are able to collaborate. In response to these needs, the visualization system should consist of libraries for the naive programmer, an external graphical user interface for the advanced programmer, an environment that lets end users run the animation, and tools for collaboration.
We have presented three algorithm visualization systems that realize this model in two specific domains. GASP and GASP-II were designed for the domain of computational geometry. VADE was designed for the domain of distributed algorithms.

7 Low Fidelity Algorithm Visualization

Chris Hundhausen
University of Hawaii at Manoa, USA

Computer science educators have traditionally used algorithm visualization (AV) software to create graphical representations of algorithms that are later used as visual aids in lectures, or as the basis for interactive labs. Typically, such visualizations are "high fidelity" in the sense that (a) they depict the target algorithm for arbitrary input, and (b) they tend to have the polished look of textbook figures. In contrast, "low fidelity" visualizations illustrate the target algorithm for a few, carefully chosen input data sets, and tend to have a sketched, unpolished appearance. Drawing both on the findings of ethnographic field studies I conducted in a junior-level algorithms course, and on the results of an experiment I conducted that compared the educational effectiveness of high and low fidelity visualizations, I motivate the use of low fidelity AV technology as the basis for an alternative learning paradigm in which students construct their own visualizations, and then present those visualizations to their instructor and peers for feedback and discussion. To explore the design space of low fidelity AV technology, I present a prototype language and system derived from empirical studies in which students constructed and presented visualizations made out of simple art supplies. The prototype language and system pioneer a novel technique for programming visualizations based on spatial relations, and a novel presentation interface that supports reverse execution and dynamic mark-up and modification. Moreover, the prototype provides an ideal foundation for what I see as the algorithms classroom of the future: the interactive "algorithms studio."

8 Algorithm Explanation: Focussing and Invariants

Reinhard Wilhelm
Universität des Saarlandes, Germany

Joint work with:
Tomasz Mueldner, Acadia University, Canada

We propose to preprocess software before visualizing it. Preprocessing is done

by shape analysis, a particular static program analysis technique. Shape analysis attempts to compute invariants at program points. Invariants both express structural as well as non-structural properties of the contents of the heap. Typical structural properties of heap elements are being pointed to by a specific pointer varibale, being reachable from a specific pointer variable or some pointer variable, being the target of at least two different heap pointers, and lying on a cycle.
Non-structural properties encompass properties such as having a data component wich is greater than that of the left neighbour (child) and smaller than that of the right neighbour (child). Structural properties can be used in abstracting concrete heap contents of arbitrary size to abstract heap states of bounded size. It can also be used to make software visualization focus on active parts of the heap contents, i.e. those parts of heap data structures where the algorithm currently works.
A particular problem connected to our approach of visualizing a program preprocessed by shape analysis is the necessity to deal with uncertainty. Our visualizations of algorithms do not work on concrete input data, but show abstract executions representing all concrete executions. Properties of some abstract data may not be known to hold, but may still be have to be represented. The best examples are programs working on totally ordered domains, e.g., sorting programs. Our visualizations may encounter abstract data that are incomparable to other data. Hence, only a partial order may exist. However, the visual representation of the abstract data should not induce the impression of comparability in the viewers mind. Specific solutions to this problem are presented.
Appropriate visualizations of structural properties have to be found and visualizations of non-structural properties have to be impressed on top of them. A visual calculus is envisioned in which the results of the shape analysis of a piece of software can be visulaized more or less automatically. Examples are given for binary search trees and red-black trees.

9 Jeliot as a program animation tool

Erkki Sutinen
University of Joensuu, Finland
University of Linköping, Sweden

Joint work with:
Moti Ben-Ari, Weizmann Institute of Science, Israel
Jorma Tarhio, Helsinki University of Technology, Finland

Jeliot is a program animation system which allows a Web user to write a Java

code, determine its visual appearance on the screen and have the system automatically produce an animation of the code according to the given visual guidelines. A related version, called preliminarily Jeliot 2000, is a stand alone system with a reduced set of features. For example, the animation will be compiled into the visual format in a fully automatic way, without users' preferences.
Although the history of Jeliot started from ready made animations for string algorithms, implemented by XTango, the development team soon understood that it is not seeing or observing existing animations but constructing them for one's own codes that contributes to positive learning outcomes. Both Jeliot and Jeliot 2000 serve this purpose.
To design an efficient visualization environment, one has to think carefully of the aimed target group. Especially in the case of Jeliot 2000, the intended audience consists primarily of beginning programmers, independently of the language they use. The emphasis is on understanding the basic language structures. Evaluation of the learning outcomes indicates that this approach is particularly useful for mid-performers.
The original Jeliot serves a bit more advanced students who already can manage the features offered by the environment. Moreover, they can focus the visualization into the variables which they are interested in. This helps in crossing the learning boundaries at the zone of proximal development (Vygotsky).

10 State Chart Visualization

Rym Mili
University of Texas at Dallas, USA

ViSta is a tool suite designed to support the requirements specification of reactive systems. It enables the user to prepare and analyze a diagrammatic description of requirements using the statechart notation. ViSta includes a template wizzard, a graphical editor and a statechart visualization tool. The template wizzard guides the user through the steps necessary for the extraction of relevant information from a textual description. The statechart visulization tool offers a framework that combines hierarchical drawing, labeling, and floorplanning techniques, designed to work in a cooperative enviroment.

11 GXL - Graph eXchange Language

Andreas Winter
Universität Koblenz-Landau, Germany

The fields of reverse engineering and program analysis have matured to the extent that there are many tools to extract information about programs, to manipulate and analyze this information, and to visualize it. What is missing is a generally accepted means to allow these tools to interoperate.
GXL offers a graph-based format supporting tool interoperability on the data interchange level. GXL has evolved from many discussions among groups developing reengineering tools. It was also influenced by similar activies on tool interoperability in graph drawing and graph transformation.
In GXL instance data and their according schema data is represented by typed, attributed, directed graphs. Both, instance data and schema data are exchanged as XML documents following ONE common DTD (document type definition). The repesentation of graphs and schemas follows the notation given for UML object and class diagrams.
At the Dagstuhl seminar on "Interoperability of Reengineering tools" (January 2001) GXL 1.0 was ratified as standard exchange format in the software reengineering community. GXL also defines the graph structure part in the standard exchange format for graph transformation systems (GTXL). Currently more than 15 groups from industrie and accademics are developing tools using and supporting GXL.

URL: `http://www.gupro.de/GXL`

12 Structure and constraints in interactive exploratory algorithm learning

Nils Faltin
University of Oldenburg, Germany

Traditionally an algorithm is tought by presenting and explaining the problem,the algorithm pseudocode and finally an algorithm animation or a sequence of static snapshots. My aim is to foster creativity, motivation and high level programming

concepts by providing the student an alternative route to algorithm understanding: exploratory learing. The algorithm is structured into several functions and this structure is presented to the student. The student is encouraged to device a pseudocode description himself. An instance of the problem is presented on the level of each algorithm function. A graphical simulation of the data structures and some of the algorithm functions are provided. It is the students task to find out a correct sequence of function calls that will solve the problem instance. The instructor can control the difficulty of the task by providing algorithm constraints. Each new constraint will shrink the solution space and thus ease the task. Algorithm structure together with algorithm constraints can be seen as an alternative way of describing an algorithm. More information can be found at:

URL: `http://www-cg-hci-e.informatik.uni-oldenburg.de/~faltin/SALA/int_vis_alg_e.html`

13 Visually Teaching Network Protocols

Pierluigi Crescenzi
Universita degli Studi di Firenze, Italy

In this talk we propose a general framework for building network protocol visualizations. To this aim, we start from the Abstract Protocol (AP) notation, which is a useful formal notation for specifying network protocols, and we propose a Java based implementation of this notation. By using our implementation, prototyping and implementing a network protocol turn out to be a very easy task (once its AP specification has been formalized). Moreover, by suitably adding visualization components to the implementation, the visualization of the network protocol (mainly for teaching purposes) can be done in a completely automatic way.

14 Developing GATO and OATBox with Python: Teaching Graph Algorithms through Visualization and Experimentation

Alexander Schliep
Universität Köln, ZAIK, Germany

CATBox (Combinatorial Algorithm Toolbox) is an interactive course on discrete mathematics using Gato (Graph Animation Toolbox) to supply algorithm animation and graph visualization allowing students to enrich the learning experience through experimentation. Using Python as the language to represent and implement algorithms together with the novel concept of animated data structures, which allows to easily encode visualization rules linking cause and visual effect, provides a (semi-)automatic visualization environment. This allows to extend experimentation also to algorithms.

15 Animating Algorithms Post Mortem

Carsten Görg
University of Saarland, Germany

Joint work with:
Stephan Diehl, University of Saarland, Germany
Andreas Kerren, University of Saarland, Germany

In the first part of our presentation we introduce two new generative approaches of animated computational models. These approaches are applied in context of educational software systems for compiler design. We shortly describe the implementation of the first approach. Based on the experiences with this prototype implementation we have developed the second approach and the GANIMAL framework was designed. This framework consists of a generic algorithm animation system, which offers a unique set of possibilities because of its graphical base package, concurrent runtime system with graphical user interface and its programming and animation specification language GANILA. In the second part we present a generic algorithm which computes a Foresighted Layout for dynamically drawing a sequence of evolving graphs while preserving the mental map. The algorithm is generic in the sense that it takes a static graph drawing algo-

rithm as a parameter. Furthermore we discuss some applications of Foresighted Layout including a 3D grapher which uses the third dimension as a history.

URL: `http://www.cs.uni-sb.de/GANIMAL`

16 Graph Animation

Carsten Friedrich
University of Sydney, Australia

Enabling the user of a graph drawing system to preserve the mental map between two different layouts of a graph is a major problem. Whenever a layout in a graph drawing system is modified, the mental map of the user must be preserved. One way in which the user can be helped in understanding a change of layout is through animation of the change. In this talk, we present clustering-based strategies for identifying groups of nodes sharing a common, simple motion from initial layout to final layout. Transformation of these groups is then handled separately in order to generate a smooth animation.

17 Visualization for Fault Location

John Stasko
Georgia Institute of Technology, USA

Large test suites are frequently used to evaluate the correctness of software systems and to locate errors. Unfortunately, this process can generate a huge amount of data that is difficult to interpret manually. We have created a system called Tarantula that visually encodes test data to help find program errors. The system uses a principled color mapping to represent how particular source lines act in passed and failed tests. It also provides a flexible user interface for examining different perspectives that show the effects on source regions of test suites ranging from individual tests, to important subsets such as the set of failed tests, to the entire test suite.

URL: `http://www.cc.gatech.edu/gvu/softviz`

18 Interactive Visualization of Java Programs for performance analysis and debugging

Wim De Pauw
IBM T. J. Watson Research Center, USA

Jinsight is a tool that displays a Java program's behavior at execution. It displays object population, messages, garbage collection, bottlenecks for CPU time and memory, thread interactions, deadlocks, and memory leaks. Jinsight can also take repetitive execution behavior and boil it down to its essentials, eliminating redundancy and uncovering the highlights of an execution. By displaying program behavior and hot spots from several perspectives, Jinsight strengthens your ability to understand, debug, and fine-tune your program.
Jinsight advances the analysis of dynamic, object-oriented (OO) programs in a number of ways:

- It is fully object-oriented. Most performance-tuning tools for OO languages do little more than profile methods as they do procedures in non-OO languages. Some go as far as showing the total number of objects per class. But the OO programming model is fundamentally different from the procedural model, and much of the power of OO is lost on conventional tools. Jinsight's views of program execution use metaphors that are both natural and consistent with the OO model. These views let you visualize both objects and messages explicitly. The Histogram view lets you see calling and reference relationships among objects. The Execution, Invocation Browser, and Execution Pattern views show sequences of messages among objects as a function of time. The Reference Pattern view displays patterns of references among objects. These views work together seamlessly to reveal the inner workings of your program.

- Jinsight has powerful pattern extraction capabilities that let you deal with large, real-world traces. It presents recurring patterns of run-time behavior in a single, compact view. Pattern extraction takes what is often an overwhelming and highly redundant mass of execution information and reduces it to its fundamental interactions. It lets you peruse vast areas of the execution space without sifting through it message by message, object by object. Pattern extraction greatly simplifies run-time analysis.

- Jinsight has a unique memory leak finder. Where other tools claim to find memory leaks by "displaying an abnormal volume of instances," Jinsight can reveal the causes of the memory leaks with far greater precision. Its Reference Pattern view shows which objects are holding references that are hampering garbage collection, thereby drawing attention to the code responsible for those references.
- We have an efficient way to visualize program on-line, by using task-oriented tracing. This live visualization allows the user to carve out pieces of the execution that are related to a given task.

19 An Artistic Approach to Model and Software Engineering Design

Paul A. Fishwick
University of Florida, USA

As we obtain less expensive and more visually oriented toolkits for software engineering, we find that the structure of software evolves into a "model structure." The Unified Modeling Language provides one solution to modeling for requirements specification and design, but there are many others, each with their own symbology. Along with this movement toward modeling, the increasing emphasis on personalization in consumer products, which includes interfaces, suggests that art can play a significant role in the design of future programs. We present our research using our "rube Methodology" that encourages the application of aesthetically-based metaphors to traditional diagrammatic models to produce 3D executable software in VRML (Virtual Reality Modeling Language). We have built several dynamic model worlds, including 1) the Dining Philosophers Petri net model, 2) an Operating System kernel, 3) numerous finite state automata examples, and 4) a System Dynamics model. The current reusability aspect of rube lies in the VRML Prototype library needed for model construction.

20 On the Visualization of Java Programs

Holger Eichelberger
J. Wolff von Gudenberg University of Würzburg, Germany

The Unifed Modelling Language (UML) has become the standard language for object-oriented analysis and design. Static structure diagrams like class diagrams visualize design aspects, serve as a guideline for the implementation and can be used to document a concrete implementation. This process is automated in modern CASE tools which do not retrieve all the information provided by the source code and which still have problems in automatic layout of the generated diagrams. Further information like the participation of model elements in design patters and the visualization of various versions of the same program are not considered by current CASE tools. Information retrieved from source code or exported by software development tools has to be stored into a format which re the items to be visualized. We have taken three languages into account: XMI (XML Metadata Interchange), UMLscript and TA (Tuple-Attribue-Language). XMI is an extendible OMG standard based on XML which is currently implemented by different tool vendors. It provides metamodel exchange and stores all information of all kinds of UML diagrams but it is complicated to read and not human-writable. Vendor specific extensions to store layout information add additional complexity for the use in external tools. UMLscript is a programming language for object oriented design and can be used for the specification of UML class diagrams. One of the design goals of this language is that it has to be human readable and writable. Because it is implemented by a generated parser it is difficult to adopt to new versions of UML. Currently it is the language which is used in our tools. TA, invented by R. Holt, combines the advantages of XMI and UMLscript. The basic ideas of TA combined with other features have been implemented in GXL (Graph Exchange Language) which is an upcoming standard for the exchange of graph information. An extension can be used to specify class diagrams. In order to retrieve information from source code we have implemented a common parsing API for source code in Java called JTransform. All structural information including comments are represented in a highly configurable parse tree. On the parse tree additional binary relations like containments, references, dependencies, downcasts, instance creations and method categorizations into predefined categories (like constructors) and user defined categories can be calculated. Using these relations it is easy to retrieve associations and aggregations, Filter classes and enable certain marked comments to be included in the output format. By applying a visitor class which generates the output we can transform java source code into UMLscript, XMI and GXL. Other applications are consistency checking on javadoc comments against method signatures, comment transformations,

Makefile generation and checking, if the source code is pretty- printed. The output produced by the java source code to UMLscript compiler (javac2UMLscript) might be used directly as input to the visualization framework SugiBib. SugiBib is a pure Java framework which was developed to implement a general Sugiyama algorithm (hierarchical layout). It has been refined and
J. Seemann (steps 6,9,10,12 and 14 in the following enumeration) which can be applied to layout UML class diagrams. Our current algorithm circumvents the problems of the original Seemann algorithm: diagrams with no proper hierarchy were not laid out aesthetically, nested model elements like packages and classes and comments were not regarded. The following enumeration gives an overview of the algorithm restricted to reverse engineering applications:

1. Identify the edges of the hierarchy (preference to the inheritance subgraph).
2. Order the set of nodes with respect to packages, nested nodes, clusters and collaborations. Within a subsequence like a package order the top level nodes by the weighted number of edges. This step releases the dependency between predefined sequences in the input graph and the layout result.
3. Insert additional edges to the hierarchy in order to represent the containment hierarchy of inner classes and nested packges.
4. Remove the comment nodes and attach them to the related model element.
5. Remove reflexive edges (edges which have the same node as start and end point) by inserting them into the node, which is furthermore responsible for the correct layout of its reflexive edges.
6. Remove non-hierarchical edges from the graph temporarily.
7. Transform the graph to an acyclic graph in order to get a proper hierarchy.
8. Insert a virtual root if the graph consists of more than one component.
9. Apply the network simplex algorithm to get the ranking of the graph.
10. Reduce the number of crossings by reordering the nodes of each rank.
11. Remove the edges inserted in step 3.
12. Execute the incremental extension proposed by Seemann. Therefore reinsert the non-hierarchical edges which have been removed temporarily in step 6.
13. Rearrange the graph and eliminate edge crossings which occured because of the reinsertion of edges in the last step.

14. Iteratively calculate the coordinates of the nodes with respect to the space needed by nodes. Nested structures are treated by a frame layout approach. Then calculate the orthogonal layout of the non-hierarchical edges.

15. Reintegrate the comment nodes by searching minimum displacements for the affected nodes.

We presented a way to visualize Java source code. The combination of javac2-UMLscript and SugiBib can be applied in education as well as in documentation of source code by enriching the HTML documentation generated by javadoc with automaticly generated diagrams. Latest information about our work can be obtained from our website:

URL: `http://www2.informatik.uni-wuerzburg.de/WURST`

21 Automatic Program Visualization with Sequence and Object Diagrams Using the Java Debug Interface (javavis)

Rainer Oechsle
FH Trier, Germany

The goal of javavis is to help students to understand what is happening in a Java program during execution. The focus of the first release is on sequential Java programs. The system uses the Java Debug Interface (JDI), so there are no modifications needed in the Java source code for the extraction of information. The system shows the dynamic behavior of a running program by displaying several object diagrams and one sequence diagram. There is one object diagram for each active method on the call stack. All modifications in the diagrams are made by smooth transitions.

22 Visual debugging of concurrent Java programs with UML

Katharina Mehner
Universität Paderborn, Germany

Concurrent programming is getting more and more important. While this is taken into account by modern programming languages like Java debugging facilities fall behind. Understanding errors in concurrent programs requires to deal with several threads and their execution history. Usually, textual traces, i.e., protocols from a program run, are the starting point for finding the cause of an error. However, a graphical visualization can improve the understanding of the interactions between threads. In addition, automated support is needed to find errors in traces such as deadlocks or dormant threads.
We have developed an extension to UML interaction diagrams to visualize traces with deadlocks from concurrent Java programs. Using UML, the standard language for object-oriented modeling, allows a better integration of debugging with other activities during the software development from a language perspective and minimizes the overall number of languages.
Our tool supports tracing of concurrent Java programs using a non-invasive approach, i.e., the source code is not changed. Traces can be analyzed for deadlocks. Traces and deadlocks are visualized post-mortem using the UML extensions. The tracing is based on the Java Platform Debugger Architecture. The visualization has been implemented as an extension to the UML CASE tool TogetherJ.

23 Visualizing Memory Maps

Andreas Zeller
Universität des Saarlandes, Germany

Joint work with:
Thomas Zimmermann, University of Passau, Germany

To understand the dynamics of a running program, it is often useful to examine its state at specific moments during its execution. We present *memory graphs* as a means to capture and explore program states. A memory graph gives a comprehensive view of all data structures of a program; data items are related

by operations like dereferencing, indexing or member access. Although memory graphs are typically too large to be visualized as a whole, one can easily focus on specific aspects using well-known graph operations. For instance, a greatest common subgraph visualizes commonalities and differences between program states.

24 Algorithm Animation for Teaching

Rudolf Fleischer
Hong Kong University of Science and Technology, HK

Throwing C-code onto poor students is not the right way to teach algorithms. It is necessary to explain high-level concepts. Algorithm animations should support these conceptual explanations, they should show *why* an algorithm works and *why* it is fast. Current algorithm animation tools are not more than sophisticated graphical editors driven by program events, i.e., graphical debuggers that can only show *how* an algorithm runs. Automatic animation of algorithms does not work, each algorithm must be animated individually.

25 Graphical Liquid Miro: Dynamic Graphical JavaDocs

Aaron Quigley
University of Newcastle, Australia

Much of the interest in Software Visualization stems from the need to support the initial development effort. However as software engineers know, the maintenance and evolution of a software system typically constitutes the vast majority of effort in the lifetime of any software system. Graphical Liquid Miro is a Visualization system that captures the "logical navigation" software maintainer makes through a software system, design and documentation. This talk outlines the back-end liquid Miro system and how this can be coupled to a front-end graphical exploration tool. The case study presented will be centered on Graphical JavaDocs;

the visualization is used to show not only the syntactic structure of the information but also the dynamic semantic structure as introduced by software engineers using the system.

26 Visualization for the Mind's Eye

Wolfram Luther
GMU Duisburg, Germany

Joint work with:
Nelson Baloian, Universidad de Chile, Chile

Today there is a growing interest in enhancing standard visual interfaces by aural or haptic components and in complementing usual approaches to make aware logical structures or data types through different perception channels To achieve a better comprehension, we deal with new or augmented interfaces added to standard systems for data visualization and algorithm animation. As a consequence, modern information and learning systems are designed in such a way that not only sighted but also blind users can navigate within these systems.

ACM classification: `K.3.1, H.5.2`
Key words: `Software visualization`, `Tutoring systems`, `Sensory disabilities`, `User adapted interfaces`

27 Opportunities for Software Visualization in Reverse Engineering

Rainer Koschke
Universität Stuttgart, Germany

Because reverse engineering is a highly interactive and incremental process, in which results of automatic analyses need to be presented to the reverse engineer that are then validated, augmented and fed back to following automatic analyses, software visualization plays a key role in reverse engineering. Research in

reverse engineering focuses on extracting and storing information, locating specific things, reducing the amount of unnecessary information for a particular task, analyzing the extracted data and to build useful abstractions of the system under analysis. Presenting the data to the reverse engineer in a suitable manner is a main issue here and the reverse engineering research community struggles with finding solutions to this problem.
Specific problems of software visualization for reverse engineering are the following:

- graphs that are used to represent the data have semantics; automatic layouts should take the semantics of nodes and edges into account
- the amount of data that need to be visualized can be rather large; graphs with 4,000 nodes and more are typical
- reverse engineering activities require weeks, months, or even years, i.e., there is not just one graph, but many graphs that are derived from each other such that visualizations evolve and one has to keep track of this evolution
- reverse engineering activities require team-work and, hence, visualizations need to support multiple users
- reverse engineering activities are often in parallel to normal maintenance and changes are made while one is analyzing; consequently, visualizations need to be incremental
- reverse engineering needs multiple views; different dimensions of the data need to be visualized:
  - time,
  - different users,
  - same / subset / overlapping / different information,
  - and different levels of granularity,

  which raises the questions how are these views integrated, how can one navigate within and between views and how can the context be maintained during navigation?

Summarizing the topics that have been discussed at this Dagstuhl seminar, we can identify the following classes. From my perspective, most of them are specifically interesting for the domain of reverse engineering while some others deal with problems that are more relevant to other domains.

- Visual languages, for instance, are useful to express structural properties of a software system and may be used to capture the actual architecture of the system and to specify the idealized architecture.
- Algorithm animations have their strength in teaching algorithms; it is not clear to me whether these techniques scale to large programs; moreover, they seem to require some preknowledge on the algorithm in order to produce useful animations, which cannot necessarily be assumed in reverse engineering.
- Metaphorical visualizations are useful to introduce novices to a new domain; however, generally, reverse engineers understand the domain well enough and metaphors involve the danger of allowing misleading conclusions.
- Problem-driven visualizations are visualization for specific concrete purposes, like drawing UML models or state charts.
- Mental map preserving techniques attempt to help the observer of the data to maintain the context while changing from one view to the other; these techniques are extremely important as reverse engineers need to browse the system from different perspectives.
- Metric visualizations helps to identify certain spots where reengineering needs to be targeted.
- Recognition of repeated patterns in the data to be visualized may help to reduce the visual complexity.

Interestingly enough, there is surprisingly little overlap between the communities for reverse engineering and software visualization in terms of people despite of the large overlap in terms of topics. And - to my further suprise - it turned out that the visualization community is very heterogeneous, too. It is high time for our communities to team up since our common goal is to help programmers understand programs.

28 3D Visualization of Large Object-Oriented Programs

Claus Lewerentz
BTU Cottbus, Germany

Joint work with:
Frank Simon, BTU Cottbus, Germany
Frank Steinbrückner, BTU Cottbus, Germany

The size and complexity of object-oriented programs permanently grows. Frameworks and component technology allow to quickly build very complex and large systems which have to be maintained and which undergo incremental evolution. To support such maintenance and re-engineering processes program comprehension and quality assessments are key factors.
In order to understand programs and to communicate them, traditionally different forms of hierarchical diagrams are used to represent the static program structure. Examples are UML package and class diagrams, or different graph structures as inheritance trees. While these notations are useful for depicting small systems, parts of larger systems or abstract views of them, they do not well scale-up for real-world program systems. Furthermore, these notations often are designed to support the construction process rather than the analysis process of existing code.
Our approach to visualize existing object-oriented programs on the basis of extracted structure and metrics data uses attributed 3D graphs. Nodes in such graphs represent structure entities like classes or packages. They are visualized by simple geometric objects (as spheres or cubes) with geometrical properties (as color or size) representing software metrics values. Relations are displayed as straight lines colored according to their relation type (method usage, inheritance). A central idea for drawing these graphs is the use of a generic similarity and distance concept that allows to calculate metric distances for each pair of nodes. The distances may be calculated from arbitrary common property sets. The 3D graph layout algorithm is based on a spring-embedding method. It takes these distances and produces a layout which approximately preserves the node distances on an ordinal scale in the 3D space. In the resulting graph layouts the spatial relationships are meaningful and can be interpreted in the problem domain. Interaction and navigation mechanisms allow for an interactive exploration of the graphs using 3D display devices.
The visualizations are used as part of a software analysis tool environment which we use for code comprehension and re-engineering for large (MLOC) object-oriented programs. The first results from case studies together with industrial software developers are very encouraging. The 3D visualization proved to be a

very effective means to quickly detect typical design weaknesses and to give restructuring recommendations on the basis of simple visual patterns.

URL: `http://www.software-systemtechnik.de`

29 Visual Modeling and Model Visualizations

John Hosking
University of Auckland, New Zealand

This talk describes experience in constructing environments that support modeling of software systems using multiple visual and textual notations, together with reuse of the notations to provide dynamic visualization of the realised models. The background research for this work is described, together with three industrial applications of our approach: a process control modelling environment; a message translation specification environment; and a business simulation game.

30 Multi-sensory metaphors for interacting with abstract data within a Virtual Environment

Keith Nesbitt
University of Newcastle, Australia

With the advent of Virtual Environment technology it is now possible to construct new styles of user interfaces that provide multi-sensory interactions. For example, interfaces can be designed which utilise 3D visual spaces and also provide auditory and haptic feedback. Many information spaces are multivariate, large and abstract in nature. It has been a goal of Virtual Environments to "widen the human to computer bandwidth" and so assist in the interpretation of these spaces by providing models that map different attributes of data to different senses.
While this approach has the potential to assist in understanding these large in-

formation spaces what is unclear is how to choose the best metaphors or models to define these mappings between the abstract information and the human sensory channels. My research examines the features of multi-sensory virtual environments, in an attempt to define generic guidelines for designing interaction metaphors.
This work is applied in a number of case study areas including...

- Interpreting "Technical Analysis" data for trading finanical instruments on the stock market.
- Understanding complex software designs.
- Investigating logicstical data.

31 Using JFLAP for Visualization and Interaction in the Automata Theory Course

Susan H. Rodger
Duke University, USA

We present a suite of tools for teaching the automata theory course in an interactive and visual manner. JFLAP is a tool for creating and simulating deterministic and nondeterministic automata, pushdown automata, and 1-tape and 2-tape Turing machines. In addition, one can interactively convert representations of languages from one form to another. Examples include converting a regular grammar to an NFA to a DFA to a minumum state DFA to a regular expression. Pate is a tool for parsing restricted and unrestricted grammars, showing either the textual derivation or the graphical parse tree. In addition one can interactivly transform a context-free grammar to CNF. JAWAA is a tool for easily developing animations on the web, which can be used in any course. All of these tools are used in CPS 140 at Duke University and many other universities to provide hands-on experimentation of automata theory concepts.

Tools web page: `www.cs.duke.edu/~rodger/tools/`
Slides of Talk: `www.cs.duke.edu/~rodger/talks/softvis01/talk.html`

32 Integrating Animation into Comprehensive Teaching and Learning Resources for the Web

Rockford J. Ross
Montana State University, USA

Software visualization has long been heralded as a means to enhance the learning of difficult concepts. Many interesting software visualizations systems have been developed in computer science for educational purposes, notably algorithm animation systems. In spite of their appeal, however, most of these systems languish and are not widely used in the classroom. We have identified a number of reasons for this, including technical issues surrounding the use of a new visualization system, the learning curve associated with the effective use of visualization software, and the problem of integrating new visualization software into an existing course. Technical issues include the downloading, installation, and maintenance of the visualization software in the local environment, issues which may become insurmountable if the software in question is platform dependent. The learning curve for a new software visualization system is not insignificant, as the instructor must first become very familiar with the system, and then this knowledge must be imparted to each new group of students. Course integration is also a serious impediment. If visualization software is to be used in a course, where is its use most appropriate? How can it be woven seamlessly into the existing fabric of a course? All of these issues can be summed up in one word: time. Faculty simply do not have the time to deal with all of these issues.

We have begun a concerted effort to realize the promise of software visualizations in enhancing teaching and learning by implementing an integrative approach that addresses each of the issues raised above. The result is what we call active learning hypertextbooks. Such hypertextbooks are envisioned as replacements for (or at least equal companions to) traditional textbooks; in any case, they are to be considered to be the major teaching and learning resource for a course. They incorporate traditional textual material enhanced for learning through hyperlinks, sound, visual information, and embedded, interactive applets that engage the student in active learning of important concepts. Hypertextbooks are also organized through hyperlinks to address different levels of maturity and different learning styles.

In our scheme, hypertextbooks are HTML- and Java-based so that they run in standard web browsers, eliminating most of the technical problems associated with downloading, installation, and maintenance, as well as unpleasant platform dependency problems. There is only a small learning curve associated with

well-done hypertextbooks, as both the instructor and the students are already comfortable with the use of browsers and the web. Course integration issues also pale, because a hypertextbook is the main resource for the course and as such is the integration of the course materials. As a result, the issue of faculty time becomes a moot point: a hypertextbook should be as easy to use as a traditional textbook. We are exploring these ideas while constructing a hypertextbook called Snapshots of the Theory of Computing in an ongoing project in the Webworks Laboratory at Montana State University.

URL: `http://www.cs.montana.edu/webworks`

33 aiCall - Call graph and control flow graph visualization for developers of embedded systems

Alexander A. Evstiougov-Babaev
AbsInt, Germany

aiCall is a software visualization tool which helps programmers to better understand their software, generally improving learning, speeding up development and saving considerable effort and expense. aiCall visualizes the call graph and the control flow graph of embedded application code. Currently supported targets are Infineon C16x and STMicroelectronics ST10. These microcontroller families are very popular and widely used in consumer goods (cellular phones, CD-players, washing machines) and in safety-critical environments (airbags, navigation systems, automotive and aircraft controls).
The complexity of embedded software increases continuously. Typical applications have to handle many sources of inputs which often requires complex interrupt handling code. Furthermore, embedded applications are usually time-critical and/or safety-critical. Due to the 'embedded' aspect the use of debuggers is often restricted. Advanced software visualization and static program analyses tools can help the developers of embedded systems to master the increasing complexity.
aiCall reads assembly files in .src format as produced by the Tasking C compiler for C16x/ST10 and generates a .gdl (graph description language) representation of the call graph and the control flow graph. GDL is the input format for the graph layout software aiSee, which is integrated into aiCall. aiSee provides for fast layout calculation and excellent graph readability, and supports recursive

subgraph nesting which is mandatory for hierarchical representation of software structures.
The top-level graph is the call graph. Each routine is represented by a node. Edges show the calling relationship between routines. Subgraph nesting operations enable the user to view the content of a routine, i.e. the routine's control flow graph, where nodes represent basic blocks and edges represent the control flow. The basic blocks are labeled with C source code snippets. Subgraph nesting operations enable the user to view the content of a basic block: the sequence of assembly instructions corresponding to the particular C source code snippet. Additional information windows enable viewing source code comments, internal addresses and source file destinations.
aiCall supports multi-page printer output, 15 basic graph layout algorithms, fisheye views and animation after relayout. aiCall is easily retargetable and provides for smooth integration of other static program analysis components, e.g. for stack usage analyses. Stack height differences can be shown as annotations in the call graph and the control flow graph. Critical program parts can be immediately recognized thanks to color coding.

URLs to aiCall:
Homepages: `www.aicall.de` and `www.absint.com/aicall`
Downloads: `www.aicall.de/download` and `www.absint.com/aicall/download`

URLs to aiSee:
Homepages: `www.aisee.de` and `www.absint.com/aisee`
Downloads: `www.aisee.de/download` and `www.absint.com/aisee/download`

34 Algorithm Animation Based on Data Flow Tracing

Jaroslaw M. Francik
Silesian University of Technology, Poland

Successful algorithm animations usually go beyond isomorphic mappings of program data or code to graphical representation of program semantics, and they provide a high level of abstraction, supplying an extra information on the semantics and meaning that is behind the code. The goal of the presented work is to introduce into an animation some elements that significantly increase the level of abstraction - in a strictly automatic mode, without any additional effort on the

part of the visualiser. An original method of algorithm animation based on data flow tracing has been proposed. The key features of this method are as follows:

- Focusing on data flow rather then temporary values of data structures. Data flow operations are considered to be operations of change of data addresses, and are visualized by smooth changing the selected attributes of graphical objects; usually these attributes are spatial coordinates, giving an impression of smooth motion. In the same time the changes of data values are also visualized, usually using such attributes, as width, height and colour of graphical objects.

- Suppressing information on the variables considered to be unimportant (e.g. detailed auxiliary variables). It is enough to omit the definition of graphical representation for those variables to make the machine automatically exclude them from the final animation; all the data flows are automatically reconstructed even if the omitted variables were their intermediary stages.

- Suppressing information on temporal order of execution - where applicable. Some operations are shown as if they were executed in parallel, synchronously.

All the three paradigms simplify the image of some complex operations in a way which is usually obtained in other systems manually.
To discover what information could be suppressed, the proposed animation system architecture applies the dynamic information delivered from the program being visualized. It is then analysed using Petri net formalism.
The proposed method has been practically verified in an algorithm animation system called *Daphnis*. In its current, functional but still prototype version, *Daphnis* is a general purpose, language-independent and easy-to-use system. The system is suitable for both didactic and engineering applications. Further information is available at:

URL: `http://www-zo.iinf.polsl.gliwice.pl/~jfrancik/aa`

A Presentations without Abstract

SV Work within the Knowledge Media Institute

John Domingue
Open University, Knowledge Media Institute, GB

Visualization of Algorithms

Ludek Kucera
Charles University, CZ

A Comprehensive Framework for Defining Software Visualizations

Steve Reiss
Brown University, USA

Visualizing Object-Oriented Systems

James Noble
Victoria University of Wellington, New Zealand

B Software Demonstrations

KIEL

Rudolf Berghammer
University of Kiel, Germany

LEONARDO

Camil Demetrescu
Universita di Roma "La Sapienza", Italy

Algorithm Animations

Markus Eiglsperger
University of Tübingen, Germany

VAM

Peter Ziewer
University of Trier, Germany

GANIMAM

Andreas Kerren
University of Saarland, Germany

GUPRO

Andreas Winter
Universität Koblenz-Landau, Germany

GEOWIN

Stefan Näher
University of Trier, Germany

BLOOM

Steve Reiss

Brown University, USA

JELIOT

Erkki Sutinen
University of Joensuu, Finland

HeapSort/BinomialHeap

Nils Faltin
University of Oldenburg, Germany

UML-Layout

Holger Eichelberger
J. Wolff von Gudenberg University of Würzburg, Germany

3D-OO Systems

Claus Lewerentz
BTU Cottbus, Germany

JAVAVIS

Rainer Oechsle
FH Trier, Germany

AGD-Library

Petra Mutzel
Technical University of Vienna, Austria

Multi-Sensory Environment (Video)

Keith Nesbitt
University of Newcastle, Australia