Computational Statistics manuscript No.
(will be inserted by the editor)

The plumbing of interactive graphics

Hadley Wickham[1], Michael Lawrence[1], Dianne Cook[1], Andreas Buja[2], Heike Hofmann[1], Deborah F. Swayne[3]

Received: date / Revised version: date

1 Introduction

What is a pipeline, and why do we need one for interactive graphics? This conceptual paper attempts to answer these questions, building on previous work by Buja et al. (1988) and Sutherland et al. (2000). A pipeline controls the transformation from data to graphical objects on our screens, and we argue that the pipeline must be present, in some form, in all graphics software. The pipeline is made explicit in descendants of DataViewer (Buja et al., 1986).

The essence of a pipeline is data moving through a series of transformations. To allow updates from plot interaction data also must be able to move backwards along the pipeline, using the inverses of the original transformations. The metaphor of a pipe breaks down somewhat here, as pipelines typically flow in one direction, but also because there are multiple ways we can control the flow of data along the pipe. In the second part of this paper, we will discuss some possible means of control, and propose one design which we think is particularly appropriate for the needs of interactive graphics.

Finally, we will conclude with an discussion of how these ideas are being implemented in the next version of GGobi (Swayne et al., 2003), a tool for highly interactive and dynamic data visualisation.

2 Related work

Apart from the details of dataviewer (Buja et al., 1986) and its descendants, XGobi (Swayne et al., 1991), Orca (Sutherland et al., 2000), and GGobi (Swayne et al., 2003), little has been published about the flow of data within interactive graphics software. In this section we summarise related work, drawing on academic publications, conversations with software authors, and the source code, where possible.

The early work of Fisherkeller et al. (1975); McDonald (1982) and Becker and Cleveland (1987) started the field of statistical graphics, but they published the uses of their new tools, not the details of their data structures. We might surmise, due to the speed of computation at the time, that their software was written very specifically for the task at hand: focussing on adequate speed for interaction, not generality.

Using a symbolics lisp machine, Stuetzle (1987) created a interactive graphics system using the now familiar window-based interface on a personal computer. Linked brushing was mediated through the data, and could be performed interactively with a brush, or scripted with the command line: "`send county-44 :highlight`". Quail (Hurley, 1993; Hurley and Oldford, 1999), also written in lisp, provided a powerful and flexible linking system. This system worked by connecting plots directly, not mediated through the data, and also supported scripted manipulation.

XmdvTool (Ward, 1994) uses parallel coordinates, scatterplot matrices, glyphs and dimensional stacking to explore complex data simplified using hierarchical clustering. Recent work has given some insight into the data processing done behind the scenes: Doshi et al. (2007) describes a caching mechanism that takes advantage of the periods when the user is looking at and processing on screen graphics to precompute expensive operations that the user may wish to do next. In mmvis (Ardis et al., 2000), each user action is tied to a *component*, which is responsible for updating the appropriate views. DAVIS (Huh and Song, 2002), a java application, experimented with tours.

[1] Iowa State University, Ames, IA; [2] University of Pennsylvania, Philadelphia, PA; [3] at&t, Florham Park, NJ

The Augsburg impressionist software (Manet, Unwin et al. (1996); Cassatt, Winkler (2000); Mondrian, Theus (2003); Klimt, Urbanek (2004); Gaguin, Gribov (2007)) has focussed on new types of graphics and improved models of interaction, not on the data pipeline. While there are no published accounts of the data model that underlies these applications, personal communication with some the authors and inspection of the code has revealed some details. In particular, little data manipulation takes place, apart from the computation of multidimensional contingency tables necessary for categorical graphics. Wilhelm (2005), another Augsburg graduate, discusses general issues regarding interactive graphics, but does not deal with computational considerations.

It is even harder to find out how commercial software works. Conversations with experienced users and inspection of documentation indicates that Datadesk (Velleman, 1992) has a powerful and speedy pipeline, but the structure is unknown. SPSS (SPSS Inc, 2007) has implemented interactive extensions to the grammar of graphics, and uses the fixed pipeline outlined in (Wilkinson, 2005).

Outside of statistical graphics, there has also been work on these topics in the information visualisation community. The prefuse framework (Heer et al., 2005) is a Java framework for designing web-based interactions, based on the "information visualization reference model" Heer and Agrawala (2006). It is not very well documented, and supports very little data manipulation. Improvise (Weaver, 2006, 2007) provides a more flexible framework than outlined in this paper. The general model is a directed graph, and it is the responsibility of the user to prevent cycles from forming. Even less is known about the internals of the commercial products which grew out of the work from this community, for example, Spotfire (Jog and Shneiderman, 1994) and Tableau (Stolte et al., 2002).

3 What is a pipeline?

With the pipeline we want to make explicit the process of taking raw data, visualising it and then updating the data based on interactions with that visualisation. A pipeline takes the raw data that we want to display and after a series of steps produces explicit instructions on how to render the end product. In particular, from a vector of data coordinates, measured in any system of units, it generates a vector of screen coordinates, in pixels. This pipeline has to be present in every graphics program—all software must perform these transformations—although it may not be explicit. We argue that making it explicit has important advantages: it helps with computation and implementation, and it makes it possible to compare different interactive systems on a deeper and more objective level.

For this paper, we will consider the original data matrix to be augmented with extra columns which describe additional details about the observations. For example, extra columns give the colour, shape and size of the glyph that represents that point. The data matrix is also augmented with column level information such as the range of each variable. This augmentation is largely a computational and notational convenience.

A pipeline is composed of pipes, or stages, each of which encapsulates a task. At a high level, a pipeline stage is simply a function and its inverse, with a set of parameters that controls its operation. At a more detailed level, a pipeline stage is an actor which (1) responds to events, (2) generates and forwards events, (3) receives data, (4) applies some function to the data (selecting rows or columns, imputing values, applying a linear transformation, etc), and (5) passes the data on. Depending on what event triggers the change these changes either cascade up or down the pipeline.

A stage needs the inverse function because the pipeline also coordinates the flow of information in the other direction: from plot to data. For example, when a user interacts with a point on screen, we must be able to map backwards along the pipeline to figure out how to adjust that point in the original data. One type of interaction that is particularly useful is brushing, or selection. Brushing changes the colour (or size, or shape) of an observation. Because the change flows back to affect the original data set, it also flows downstream to all other plots so that we get linked brushing. Coordinating this flow of data is not simple, and the next section discusses some of the possible approaches for coordinating the pipeline.

4 Coordinating the pipeline

The simplest pipeline looks something like Figure 1. We have our augmented data, pipeline stages to transform and standardise the data, and then the final plot. Changes can occur for two reasons: a change to the data, or a change to the parameters of a stage. For example:

– The original data may change if we have streaming data, or if we are creating an animation from within another program.

– Changing which variables are transformed, or the type of transformation, is a change to the parameters of the stage and requires updating everything downstream.
– Similarly, changing the type of standardisation also requires downstream updates.
– Finally, interaction with the plot, such as brushing, requires updates to the original data.

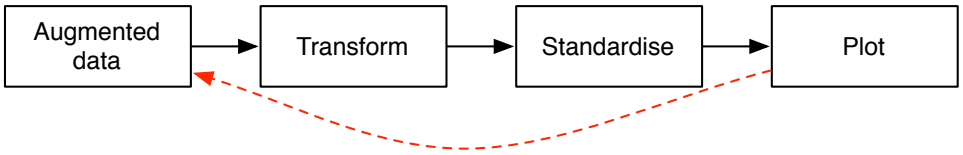

Fig. 1 Simple pipeline. Dashed red line indicates data flowing from the plot to the original data

The possibilities for change in this very simple example are already quite large, but we can make a simple rule which ensures everything remains consistent: changes to the parameters of a stage update all stages downstream, and changes from interaction with the plot flow upstream. Changes need to flow up one stage at a time so that any alterations can be reversed before updating the original data. This shown in figure 2

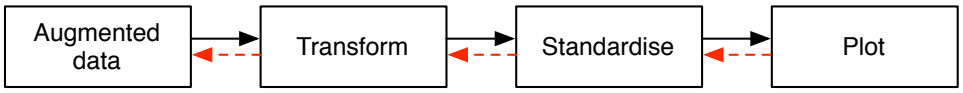

Fig. 2 Simple pipeline with reverse pipeline. Dashed red line indicates data flowing from the plot to the original data

Unfortunately life is not that simple, and in any non-trivial application, we will have multiple plots, as in figure 3. In this case, creating a consistent system of updates is much more complicated. We describe three possibilities below: plots update other plots, a central commander tells everyone what to do, or updates follow the pipeline. We will call the plot that is interacted with the active plot.

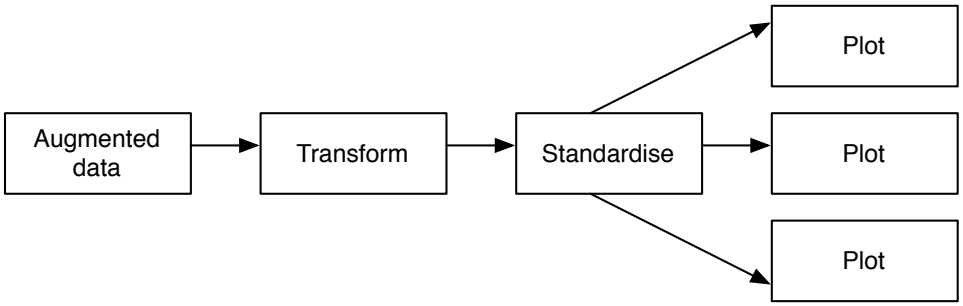

Fig. 3 Pipeline with multiple plots.

Plot-to-plot communication, figure 4, relies on the active plot knowing how to update the other plots. At the minimum, it must know what other plots exist, and then tell each of them what has changed. This design is simple, but has considerable drawbacks: each plot must know a lot about the other plots, and more importantly the original data is never updated. This makes this design increasingly complicated as the number of plot types increases, and we can never inspect the original data to see what changes have been made. Interactions between changing the plot and changing the parameters of the pipeline stages will be complicated.

A better design uses a central commander to take the burden off the plot objects, as shown in figure 5. This is a common object oriented design pattern (Gamma et al., 1995), and used in many software packages (Apache Software Foundation, 2007). It is useful because it is centralised, with a single object encapsulating all the control paths. The plumbing exists in one place and a single component "commands" the pipeline stages like a puppeteer: in essence, the commander *is* the pipeline. However, this design suffers a similar problem to the plot-to-plot design: the commander object needs to know how to coordinate everything that's going on, and as the number of components increases we can get a corresponding increase in complexity.

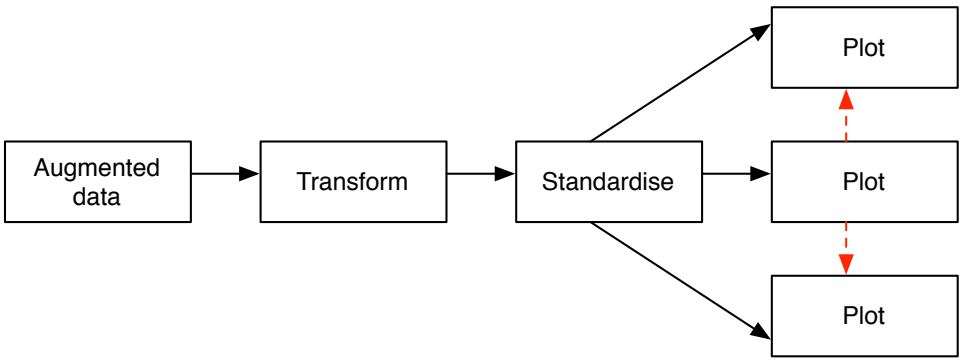

Fig. 4 Pipeline with plot-to-plot updates.

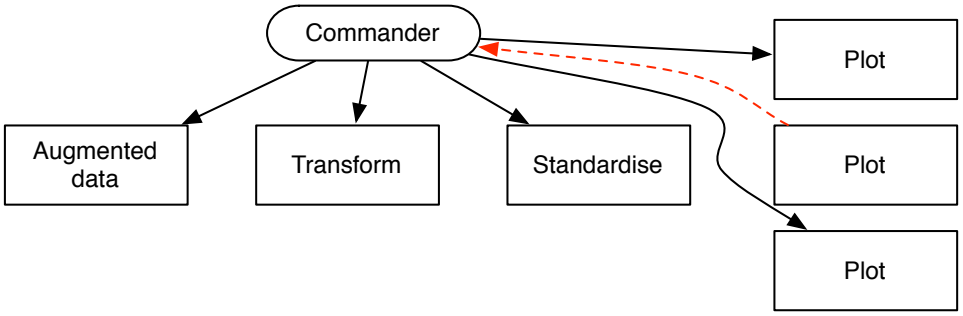

Fig. 5 Pipeline with central commander to control flow of updates..

In the final design, and the one that we think is most appropriate, events follow the same path as the pipeline. There is a two stage approach where first the data changed flows right back to the original data, and then the events flow back out to the plots. There are some complications in this design to ensure that we don't update things twice or get caught in infinite loops, but these are resolvable. This design is demonstrated in figure 6.

We favour our design because it is self-contained. The pipeline emerges as the stages are connected, just as in a real plumbing system. Because the operation of each stage is tightly constrained to be orthogonal to all others, we can treat new stages as black boxes and only need worry about how they are connected together. Just as plumbers have adopted as standard pipe size and set of connectors, we have adopted a common interface for pipeline stages.

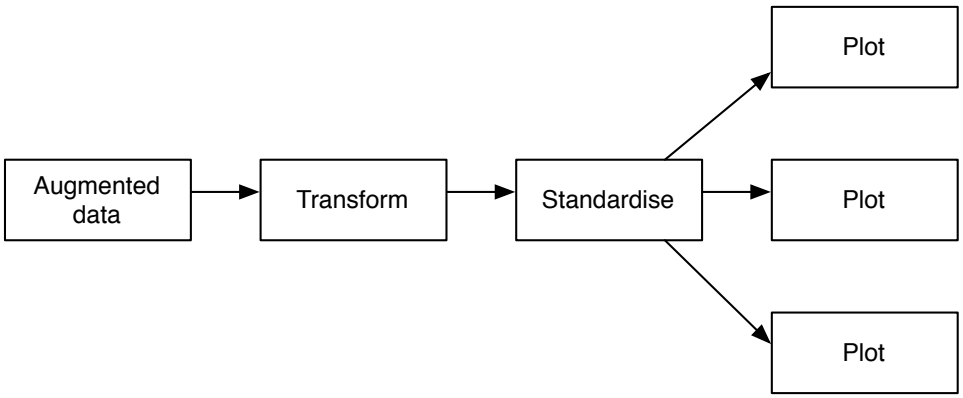

Fig. 6 Pipeline with two-stage event model, where events follow the pipeline.

Package	Type	Year released	Reference
Dataviewer	pipeline	1986	Buja et al. (1986)
XGobi	pipeline	1992	Swayne et al. (1991)
Orca	pipes	1999	Sutherland et al. (2000)
GGobi	pipeline	2000	Swayne et al. (2003)
GGobi 3	pipes, and pipeline	2007	

Table 1 Interactive graphics software with an explicit pipeline.

5 Pipes vs plumbing

It is important to reinforce the difference between pipes and the pipeline. Pipes, or stages, are the simple orthogonal components that when plumbed together form a pipeline. Some graphics software provides the pipes, while other software provides the plumbing. Software that provides only pipes is generally used as a component of other programs: it does not offer a data analysis environment. Software that offers a pipeline usually provides a tailored environment for data analysis, but if you want to extend or modify the underlying process you will need to call a plumber (a developer).

Table 1 provides a list of some interactive graphics software with an explicit notion of a pipeline. All the interactive graphics software with an explicit pipeline that we are aware of comes from DataViewer (Buja et al., 1986) ancestry.

6 Motivation and implementation

Over the past year, we have begun implementing the communication and pipeline model in GGobi. Previously, although the pipeline was an important part of GGobi, the code for the pipeline was not cleanly separate from the rest of the implementation, and changing the plumbing of the pipeline was near impossible.

The motivation for this change is simple: we want to make GGobi more flexible and powerful. Figure 7 provides an example of what we would like to be able to do with GGobi. Currently, the subset stage occurs immediately after the raw data. This means that every plot must display the same data. We would like to be able to shift this subset stage to a place much later in the pipeline, so that each plot can display different subsets of the original data. This will enable us to add features such as conditioning and shingles.

This is one advantage of the new pipeline: it makes it easy to recode the plumbing of GGobi because concerns are cleanly decoupled, and so makes it easy for developers to change how GGobi works. However, the new pipeline is more powerful yet: we can change the pipeline dynamically, at run-time, so that users could change how GGobi works. We have only just begun to explore the possibilities of this, but one simple consequence is that it becomes very easy to generate new pipelines from within R or other dynamic languages.

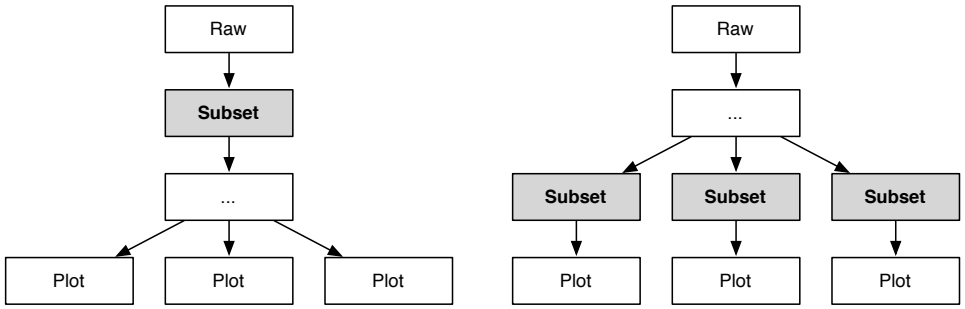

Fig. 7 What we want to be able to do with GGobi. Currently subsetting occurs as the first step in the pipeline, so that all plots show the same data, but moving it to a later point will allow different plots to show different subsets of the data.

Two other important aims are adding area plots (eg. mosaic plots) to GGobi and creating more complex displays. The original pipeline was created for scatterplots, and area plots require very different treatment. We need a better organised treatment of aggregation; currently it's handled as a sort of diversion from the pipeline. One more complex display we would like to experiment with is a 2D tour with the corresponding 1D tours displayed on the margins. This is not possible in the current framework.

The biggest concern with the new implementation is speed: will the new, more, general framework decrease the overall speed of the software? To date, there have been some slowdowns, but we are hopeful that the developer time saved with a cleaner, more flexible design, can be spent profiling and improving performance so that the net effect is positive. There are also some things that we do now that can be done much more cleanly and efficiently: subsetting simply for computational efficiency when working with large data is rather messy now, and we still store the full data longer than necessary.

7 Conclusions

This paper revisits the data pipeline for interactive graphics in light of recent work with GGobi. The major difficulty for any interactive graphics software is co-ordinating changes and ensuring that plots are updating in a timely and consistent manner. We discussed three possible designs for controlling the flow of updates, and suggest that the most appropriate design occurs when updates follow the same path as the pipeline. Implementation of these ideas in GGobi are still at an early stage, but development so far shows promise, and we have many ideas on how to use the new power and flexibility.

References

Apache Software Foundation. Package org.apache.commons.pipeline, 2007. URL `http://jakarta.apache.org/commons/sandbox/pipeline/apidocs/org/apache/commons/pipeline/package-summary.html`.

M. Ardis, K. Cox, S. Hibino, L. Hong, A. Mockus, and G. Wills. Building information visualizations: A commonality analysis. In *Information Visualization 2000*, 2000.

R. A. Becker and W. S. Cleveland. Brushing Scatterplots. *Technometrics*, 29(2):127–142, 1987.

A. Buja, C. Hurley, and J. A. McDonald. A data viewer for multivariate data. In *Computing Science and Statistics: Proceedings of the 18th Symposium on the Interface*, pages 171–174, Washington, DC, 1986. American Statistical Association.

A. Buja, D. Asimov, C. Hurley, and J. A. McDonald. Elements of a viewing pipeline for data analysis. In *Dynamic Graphics for Statistics*. Wadsworth, Inc., 1988.

P. R. Doshi, E. A. Rundensteiner, M. O. Ward, and D. Stroe. Prefetching for visual data exploration. Technical Report WPI-CS-TR-02-07, Worchester Polytechnic Institute, 2007. URL `http://davis.wpi.edu/~xmdv/docs/tr2002-Punit.pdf`.

M. A. Fisherkeller, J. H. Friedman, and J. W. Tukey. Prim-s, an interactive multidimensional data display and analysis system. In *Dynamic Graphics for Statistics*, pages 91–109. Wadsworth, 1975.

E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

A. Gribov. Gauguin (grouping and using glyphs uncovering individual nuances), 2007. URL `http://rosuda.org/software/Gauguin/gauguin.html`.

J. Heer and M. Agrawala. Software design patterns for information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), 2006.

J. Heer, S. K. Card, and J. A. Landay. Prefuse: A toolkit for interactive information visualization. In *Proc. of ACM Human Factors in Computing Systems (CHI'05)*, pages 421–430, Portland, OR, 2005.

M. Y. Huh and K. Song. Davis: A java-based data visualization system. *Computational Statistics*, 17(3):411–423, 2002.

C. Hurley and R. Oldford. Statistical graphics in quail: An overview. In *Biennial Meeting of the International Statistical Institute*, 1999.

C. B. Hurley. The plot-data interface in statistical graphics. *Journal of Computational and Graphical Statistics*, 2(4):365–379, Dec. 1993. ISSN 10618600. URL `http://links.jstor.org/sici?sici=1061-8600%28199312%292%3A4%3C365%3ATPIISG%3E2.0.CO%3B2-M`.

N. Jog and B. Shneiderman. Starfield information visualization with interactive smooth zooming. In *Proceedings of IFIP 2.6 Visual Databases Systems*, College Park MD 20742-3255 USA, 1994. URL `citeseer.ist.psu.edu/jog95starfield.html`.

J. A. McDonald. *Interactive graphics for data analysis*. PhD thesis, Stanford University, 1982.

SPSS Inc. *SPSS Base 16.0 for Windows User's Guide*. Chicago IL, 2007.

C. Stolte, D. Tang, and P. Hanrahan. Polaris: A system for query, analysis, and visualization of multidimensional relational databases. *IEEE Transactions on Visualization and Computer Graphics*, 8(1), 2002.

W. Stuetzle. Plot windows. *Journal of the American Statistical Association*, 82:466–475, 1987.

P. Sutherland, A. Rossini, T. Lumley, N. Lewin-Koh, J. Dickerson, Z. Cox, and D. Cook. Orca: A visualization toolkit for high-dimensional data. *Journal of Computational and Graphical Statistics*, 9(3):509–529, 2000.

D. F. Swayne, D. Cook, and A. Buja. XGobi: Interactive Dynamic Graphics in the X Window System with a Link to S. In *American Statistical Association 1991 Proceedings of the Section on Statistical Graphics*, pages 1–8, Alexandria, VA, 1991. American Statistical Association.

D. F. Swayne, D. Temple Lang, A. Buja, and D. Cook. GGobi: Evolving from XGobi into an extensible framework for interactive data visualization. *Computational Statistics & Data Analysis*, 43:423–444, 2003.

M. Theus. Interactive data visualiating using Mondrian. *Journal of Statistical Software*, 7(11):1–9, 2003.

A. R. Unwin, G. Hawkins, H. Hofmann, and B. Siegl. Interactive Graphics for Data Sets with Missing Values - MANET. *Journal of Computational and Graphical Statistics*, 5(2):113–122, 1996.

S. Urbanek. *Exploratory Model Analysis. An Interactive Graphical Framework for Model Comparison and Selection.* PhD thesis, Universität Augsburg, 2004.

P. F. Velleman. *Data desk. The New Power of Statistical Vision.* Data Description Inc, 1992.

M. O. Ward. Xmdvtool: Integrating multiple methods for visualizing multivariate data. In *IEEE Conf. on Visualization '94*, pages 326–333, Oct 1994.

C. Weaver. *Improvise: a user interface for interactive construction of highly-coordinated visualizations.* PhD thesis, University of Wisconsin-Madison, 2006.

C. Weaver. Patterns of coordination in improvise visualizations. In *Proceedings of the IS&T/SPIE Conference on Visualization and Data Analysis*, San Jose, CA, 2007.

A. Wilhelm. Interactive statistical graphics: the paradigm of linked views. In *Handbook of statistics 24: Data mining and data visualisation*, 2005.

L. Wilkinson. *The Grammar of Graphics.* Statistics and Computing. Springer, 2005.

S. Winkler. *Parallele Koordinaten: Entwicklung einer interaktiven Software.* PhD thesis, Universität Augsburg, 2000.