Supporting intellectual work through artifact rendering and group review

Lars Yde
Department of Computing, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
larsyde@diku.dk

Jyrki Katajainen
Department of Computing, University of Copenhagen
Universitetsparken 1, DK-2100 Copenhagen East, Denmark
jyrki@diku.dk

Abstract: Intellectual teamwork, such as that done by software development teams, is characterized by the intangibility of its subject matter. This can make it difficult for team members and outside stakeholders to gain an overview of the product being built. Many software tools of various designs have addressed this problem, often by using graphs and charts to model on-going projects. In this paper, we suggest a design based on artifact rendering and group review and argue that it can promote overview and communication. We also present software built from that design and describe its potential uses and audience as well as its implementation.
Keywords: Computer-supported collaborative work, component-based development
Category: H.4.3, H.5.3, K.4.3

1 Introduction

Two persistent problems in software engineering are inadequate understanding and poor communication among the participants in software development projects [Sommerville 1996]. These problems have been known for decades [Naur and Randell 1969] and numerous development methodologies and tools have been offered as solutions to them. Among the tools have been computer-aided software-engineering (CASE) software and groupware for supporting various aspects of the development process. A central goal of the CASE category has been to create overview of the structure and progress of software projects, often by means of abstract, static representations such as charts and graphs — a choice of form that has several consequences. Firstly, users must be familiar with the conventions of the tool and its notation to use and understand it properly. Secondly, abstraction trades overview for completeness in that the more abstractly a project is modelled, the less detail is available. Thirdly, by abstracting away detail, CASE tools typically remove access to the content underlying the model, thus wasting opportunities for use, for example, in code inspection and group discussion. Lastly, by using static or manually updated models, many tools report not the actual, on-going state of a project but rather an intended or perceived state whose accuracy is dependent on how well the model is updated by the users.

Furthermore, one reason why software engineering has not achieved the predictability, stability and success rate of conventional engineering disciplines is lack of feedback. Consider construction engineering where, put crudely, the status of a project at any given moment is reflected by the status of the physical structure being built. Or take an electrical engineering project where the degree of correlation between blueprint and, say, a piece of circuitry can be continuously monitored by simply comparing the two, in addition, of course, to testing and inspecting the system under construction. This relationship between conception, design, and realization, where an idea is translated into detailed specification and then gradually realized as a physical entity, is not found in software engineering as the end product, the software itself, is inherently intangible. Software engineers, and, indeed, everyone engaged in intellectual work, therefore lack the overview that other engineers are often provided by the laws of physics, so to speak.

This suggests a potential need for software tools or systems that can give developers an accurate and concurrently updated representation of the artifact being built so that project status and progress can be continuously monitored. In [Section 2] we describe a prototype of such a tool which is intended to support *intellectual teamwork* in general. That is, we aim at supporting any process in which a symbolic product, an *artifact*, is being produced by individuals in group collaboration, so the use is not limited to software developers. Our main design goal was to promote overview by artifact rendering and communication by group review. By *artifact rendering* we mean the representation of intellectual artifacts by some medium to facilitate processing by humans, and by *group review* the activity through which team members exchange information and commentary on the artifact being built. Our prototype uses visualization to render an artifact that consists of textual and graphical *documents*, but other forms can be envisaged, i.e. auditory representation. Also, discussion fora can be associated with parts of the artifact rendered.

Our prototype was developed using the Java programming language and a set of compatible component technologies. In [Section 3] we discuss both and argue that implementation was possible only through such component-based development, given our limited resources. Next, in [Section 4], we propose some potential uses of the prototype, both in its present form and in subsequent versions. We then go on to discuss earlier related work in [Section 5] and finally, in [Section 6], we outline our plans for a more advanced system.

2 Description of the Prototype

The prototype aimed at fulfilling the basic needs of artifact rendering and group review was named *PeerView* and work on it was begun in the middle of May 2000 as part of Lars Yde's M. Sc. thesis. At the time of writing (mid October), a beta release has been completed and a stable version is planned for release before the end of the year. The system is released under a freeware license both in its binary and source form. In this section, we outline PeerView's design.

2.1 System Overview

PeerView provides users with a dynamic representation of on-going work by rendering the artifact of that work in a form which allows easy overview as well as inspection and discussion of detail. PeerView accomplishes this by allowing groups of users to share documents and have them rendered in a scalable visual panorama. The set of documents is constantly kept updated so as to provide a real-time window onto the artifact (e.g., a collection of code files) being developed. With each document is associated a discussion forum that enables a group to have discussions on the document in a manner similar to a USENET [WebMagic 2000] newsgroup having discussions of a set topic.

2.2 Architecture

PeerView is a client/server application with "fat" clients and a "thin" server, meaning that most system functionality is placed in the client application and relatively little in the server. Each PeerView user is assumed to be running a single instance of the client program on his or

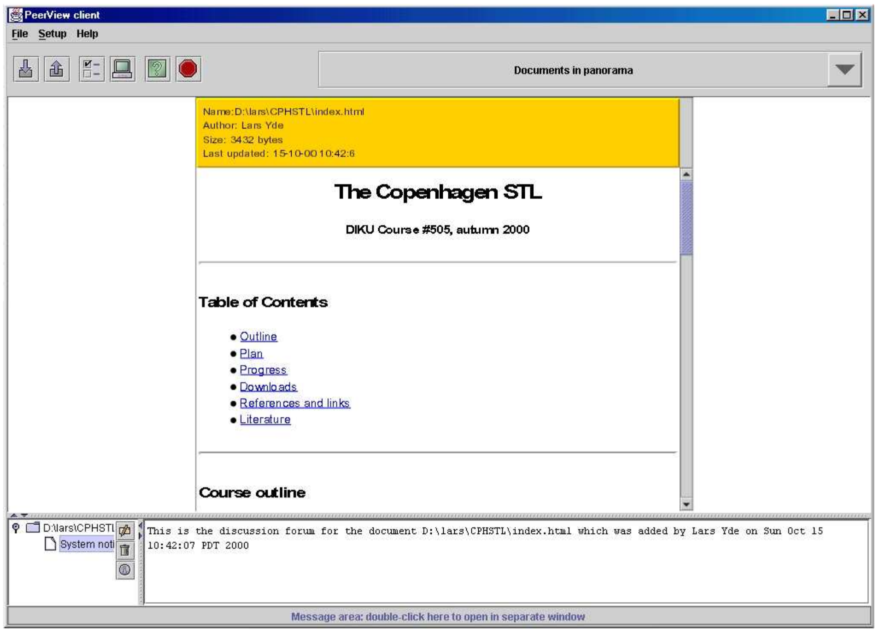

Figure 1: The PeerView client window.

her machine, but multiple instances are possible, provided they are running from separate locations (different directories). The server may run on one of the users' machines or it may run on a separate host. Currently, PeerView supports two types of connection: TCP/IP sockets and HTTP. The former is intended for local or closed networks, the latter for use over the Internet, but both may of course be used differently.

2.3 The PeerView Client

The client application provides a window onto a set of documents via a zoomable *panorama* [Fig. 1, centre] where documents can be arranged in an arbitrary pattern and zoomed to arbitrary scale as well as moved and sized individually. By centring a document, the user can access its *discussion forum* [Fig. 1, bottom] and read as well as add contributions to the on-going discussion on that document. The top of the client window is occupied by a selection panel consisting of standard *drop-down menus* and *toolbar buttons*. The drop-down box in the right-hand side of the toolbar panel lists the documents currently displayed in the panorama so

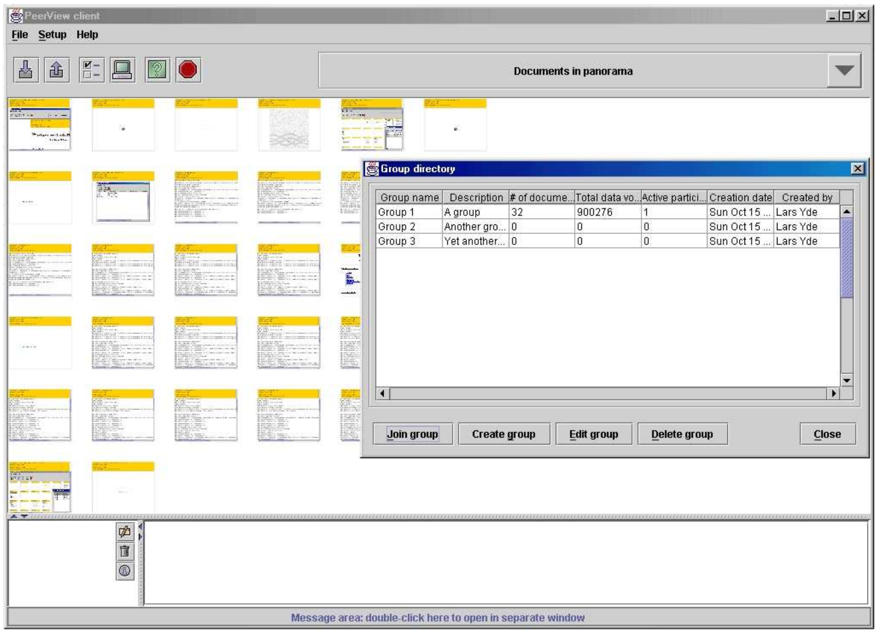

Figure 2: Panorama overview and group directory.

that each can be selected by name as well as by navigating the panorama using a mouse.

From the selection panel, the user can choose to add or remove documents, customize the client using a preferences dialog or access and modify the *group directory*, i.e., the list of groups currently available. The directory appears on top of the main window as shown in fig. 2 where the panorama is zoomed to overview, displaying in this case 32 documents arranged in a grid pattern which can be customized by the user from the preferences dialog available from the drop-down menus. Each group is listed with its current number of participants, the number of documents it comprises and their total volume in bytes. Using the button panel at the bottom of the group directory box, the user can join, create, edit and delete groups. After joining a group, the other group members, if any, are instructed by the client application to submit their documents so that they can be added to the panorama of the new group member. The new member's own documents are then broadcast to the other group members so that their panoramas can be similarly updated.

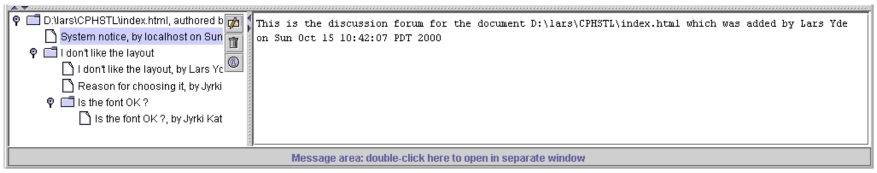

Figure 3: Discussion forum.

As indicated earlier, centring a document (by double-clicking it) causes the PeerView client to request the discussion forum for that document from the server and, upon receipt, display it in the bottommost section of the main window. From there, the user can select a position in the *topic tree* [Fig. 3, left] and using the small vertical toolbar panel choose to compose a new message, delete an existing message that has not yet been responded to, or view the *property sheet* of the selected message.

2.4 The PeerView Server

The central purpose of the server is to maintain a database of clients, groups and discussion fora which is reflected in its user interface [see Fig. 4]. It consists of little else than a listing of the groups supported, a toolbar and a menu panel from which a few, basic functions (setup and help) can be selected.

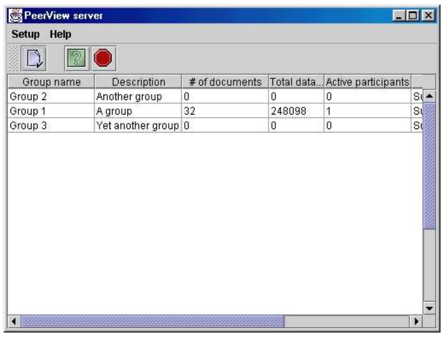

Figure 4: The PeerView server window.

2.5 Design Rationale

The choice of a client/server architecture was made based on the responsibilities of the server. A peer-to-peer architecture would have been equally feasible but would merely have placed the server functionality in an arbitrary client, resulting in little discernible difference to the user. More significant is the use of a zoomable panorama. As explained earlier, the underlying motivation is to provide overview without loss of content. The user can achieve this by zooming the arrangement of documents to his or her preferred scale, of course, but by combining zooming with a linear setup, an effect similar to that found in work by Eick et al. on the visualization tool SeeSoft [Stasko et al. 1998, p. 315] can be produced. SeeSoft uses a linear arrangement of multi-coloured columns to render an overview of a collection of code files and their associated statistics (such as time of modification), thus giving users access to large amounts of information at a glance.

On a more theoretical level, the idea of a customizable panorama that can be viewed at arbitrary levels of detail is in keeping with the notion of micro/macro readings as described by Tufte in [Tufte 1990, p. 38 ff.]:

> "...the power of micro/macro designs holds for every type of data display as well as for topographic views and landscape panoramas. Such designs can report immense detail, organizing complexity through multiple and (often) hierarchical layers of contextual reading."

Tufte's focus is on the presentation of cartographical and scientific data but his observations seem to apply equally well here.

3 Implementation Overview

The most important considerations in choosing implementation technology for PeerView were: broad user appeal, ease-of-use, high degree of portability, extensible structure and reliability. The Java programming language seemed the obvious choice given these requirements, but implementing all of the planned functionality bottom-up within the set timeframe of four months was almost surely infeasible since implementation was in the hands of a single developer (Lars Yde). The solution lay in using component libraries to implement some of the low-level features such as data distribution and visualization that would otherwise have demanded weeks of development, testing, and fine-tuning to function satisfactorily. The specific choice of components and the reason for choosing them are detailed below.

3.1 The User Interface

The PeerView interface is implemented using the Swing classes found in the Java Development Kit (version 1.2.2) [Sun Microsystems 2000a]. These are light-weight graphical components meaning that they make no (or only very limited) use of operating-system-specific code, thus ensuring maximum portability and a minimum of visual variability across platforms. The zoomable panorama which occupies the centre portion of the client window is built using the Jazz toolkit (version 1.0) [HCIL 2000], which provides primitives for building scenegraphs and scaling text and/or graphics. (A *scenegraph* is a tree structure for organizing graphical objects.) Jazz interacts smoothly with other Java technologies and its structure facilitates future extensions and modifications. In that sense, it allows development time to be shortened without imposing arbitrary restrictions on future development efforts.

3.2 Communication and Data Distribution

The PeerView communication infrastructure is implemented using the Java Shared Data Toolkit (JSDT, version 2.0) [Sun Microsystems 2000b]. It is a relatively low-level toolkit compared to some of the alternatives surveyed, such as NCSA Habanero [NCSA 1999], TANGO Interactive [WebWisdom.com 1999], and DreamTeam [LPI2 2000], all available online and free of charge. However, it has the distinct advantage of being 100% pure Java, meaning that the risk of incompatibility problems — a frequent stumbling block to developers — seemed minimized.

The JSDT requires a server to maintain a directory of clients and groups, and allows all members of a group to communicate and exchange data using an arbitrary number of *channels*. Currently, PeerView uses only a single channel per group, but extending the design later on would be easy as would adding a number of advanced features such as data encryption, access restriction and communication statistics gathering, all of which are supported in part or whole by the JSDT.

As mentioned earlier, PeerView currently supports two communication protocols, namely TCP/IP sockets and HTTP, but can easily be extended with additional protocols since the JSDT design is completely independent of its underlying implementation. Such an extension is, however, a low priority as other, more interesting features (suggested in the previous paragraph) are higher on the agenda.

4 Potential Uses for PeerView

PeerView was designed for a specific purpose, namely support of intellectual teamwork, but was deliberately made flexible through simple design and portable technology. It is therefore suited for other uses than that of facilitating intellectual work. Some possible applications of PeerView are listed below. Many of these are inspired by examples found at the Jazz website [HCIL 2000].

Repository inspection: PeerView could be used for the inspection of document repositories such as those maintained by the version management systems that are often used in software development. It is planned to try this in practice in the near future by allowing users to download the PeerView client software from a website and then access a CVS repository using that software (CVS is short for Concurrent Versions System — a popular open source version management system, see [OpenAvenue 2000]).

Web browsing: PeerView can display HTML documents in its present form and can be extended to support hyperlinks in future versions, and could, after such an extension has been implemented, function as an inter-/intranet browser.

Presentation: Documents can be displayed and arranged in PeerView's panorama and subsequently used in a presentation much like conventional slides but in a more manageable way, for example, by projecting the panorama onto a canvas and then using a mouse to navigate while giving the presentation.

Authoring: PeerView could be used as an authoring tool by researchers who could have separate copies of a shared manuscript placed on the panorama and then edit and annotate their version while conferring with others about their copies. This could also be used for commentary and proof-reading and would scale well since the zoomable PeerView panorama need not consume more user screen space as the number of authors increases.

Education: Due to its simple design and graphical interface, PeerView is accessible to most users, including children and some groups of disabled people, and so could be used both for distance learning in geographically dispersed communities and as a support tool in conventional education. One obvious example of the latter is letting visually impaired students study teaching materials at their preferred magnification using the zoomable PeerView panorama.

The above points to some general areas of application, but the number of specific uses is potentially quite large because of PeerView's extensible, open-ended architecture and design.

5 Earlier Work

The extensive directory of computer-supported collaborative work and groupware resources at [Diamond Bullet Design 1998] lists a large number of groupware applications for shared editing, conferencing, e-mailing, and other forms of group collaboration. In theory, such functionality can serve the same purpose as PeerView, but in practice, many of these products are designed for different purposes and are thus ill suited for artifact rendering and group review. Two examples are Cybozu Office [Cybozu 2000] and Lotus Notes [Lotus Corporate 2000], both of which are large suites of tools for collaborative work that facilitate resource sharing and communication. However, their size and complexity makes it impractical to use them for the narrowly defined and specific purpose that PeerView has.

Several academic initiatives have explored areas related to artifact rendering and group review and have produced software that addresses the same underlying problems as PeerView. Examples are the TeamRooms software [Roseman et al. 1996] created for supporting team collaboration and group awareness, and the WORLDS software [DSTC 2000] used for distributed access to one or more repositories of information. Both are similar in architecture to PeerView and are aimed at facilitating collaborative work through object sharing and group communication, but their designs differ from PeerView's in key areas. Both TeamRooms and WORLDS are based on spatial metaphors, i.e., the notion of shared electronic spaces. TeamRooms realizes this idea by letting users define *rooms* which are persistent fora where objects can be shared and communication can take place between the visitors of the room. Each room is equipped with facilities for collaborative work in the form of applets for exchanging information and carrying out other tasks. The TeamRooms user interface is dominated by a panorama onto which the contents of the room (i.e., the shared objects) are projected for inspection and manipulation by all users "present" in the room. An implication of such a design is of course that the set of shared objects can easily occupy more than the visible screen space and TeamRooms addresses this problem by offering a room overview radar that gives an outline, non-scalable bird's eye view of the room (the work on radar overview has been continued at the University of Calgary [GroupLab 2000] where TeamRooms was developed).

On the conceptual level, PeerView differs from TeamRooms by being a tool for artifact rendering and group review whereas Teamrooms provides shared electronic spaces for collaborative work. This is the central difference since the conceptual nature of a system usually dictates future design and development efforts. In terms of design, PeerView has fewer features than TeamRooms and consequently a simpler interface. It also addresses the issue of artifact overview differently, namely through its scalable, configurable panorama, but it is similar in its use of object sharing by projection onto a panorama surface. Technologically, the most important difference is that PeerView is written in Java while TeamRooms is a Tcl/Tk application which obviously impacts portability and potential audience since Java is platform-independent and Tcl/Tk is not, although it is widely supported. The WORLDS collaboration environment, which is called *Orbit*, differs from PeerView in much the same way as TeamRooms, but has a more complex architecture and, consequently, more stringent system requirements than both TeamRooms and PeerView.

6 Future Plans

Given the open design and implementation of PeerView, we expect little difficulty in integrating new features suggested by future users and ourselves. A handful of useful extensions have already been brought to our attention and are listed below:

- New display managers to allow advanced document formats such as RCS (the format used by CVS [OpenAvenue 2000] for storing the revision history) and streaming video in addition to the existing formats which include plain text, HTML, RTF, GIF and JPG.
- A selection of layout managers, i.e., user configurable program components for automated control of the documents in the client panorama, in addition to the default manager (the grid layout manager) now available.
- An editable and extensible property sheet associated with each document.
- Discussion fora for groups of documents organized by owner, topic, or some other shared characteristic.

The above features are for integration into future releases along with any additional suggestions deemed worthwhile.

The long term plan is to gather feedback about the use of PeerView and from that assess the viability of its philosophy and design. Provided this assessment is favourable, we plan to develop a more ambitious system for real-time artifact rendering and *process rendering*. That is, in addition to an artifact, the organization and history of a project is rendered. The system is to assemble global statistics about the state of a project by collating information and documents gathered from individual clients distributed in a network. Relevant statistics would be the number of documents worked on by each user, the number and position of changes made to those documents and, for code files, profiling and debugging information. That information should then be organized by the server and transmitted around the network to an arbitrary number of consumers who would render it, either visually or by some other medium determined by the preferences of the user. Moreover, the system could offer the review facilities of PeerView along with a set of more advanced features. Ideally, the latter would include "scribbling pads" to communicate in pictures/diagrams in addition to words, audio and video conferencing to allow real-time communication among group members, and playback facilities for observing the evolution of a project over time. Thus, by combining artifact rendering with process rendering, a both spatially and temporally accurate representation of on-going work is given.

7 Conclusion

The problems that PeerView seeks to address are not new, but its approach and design are relatively untried. Only user feedback and continued research will tell if such an approach is justified and has enough substance to bear larger initiatives. If so, the potential benefits from a more ambitious project as outlined seem significant. On a more philosophical level, the potential merit of the PeerView approach comes from the fact that it is a response to a real and pervasive problem in software engineering, namely how to get a cognitive grip of an intrinsically intangible and usually highly complex subject matter. In that sense, PeerView can be seen as a small step in the evolution towards CASE tools that can help software engineering attain the maturity of other engineering disciplines and thereby, hopefully, stem the pandemic of failed projects.

Software Availability

The latest PeerView binaries are available online at:

`http://www.diku.dk/research-groups/performance-engineering/PeerView/`

The source code and accompanying documentation will be made available at the above address before the end of this year (2000). Future developments and release plans will also appear there as will related resources. Please feel free to download PeerView and test its usefulness for yourself.

Acknowledgements

This work was supported by the Danish Natural Science Research Council under contract 9801749 (project Performance Engineering).

References

[Cybozu 2000] Cybozu, Inc.: "Cybozu"; Website accessible at http://www.cybozu.com/.

[Diamond Bullet Design 1998] Diamond Bullet Design: "Usability First™"; Website accessible at http://www.usabilityfirst.com/.

[DSTC 2000] Distributed Systems Technology Centre Pty. Ltd.: "The WORLDS Project"; Website accessible at http://archive.dstc.edu.au/TU/worlds/.

[GroupLab 2000] GroupLab, University of Calgary: "GroupLab: Laboratory for Computer Supported Cooperative Work & Human Computer Interaction"; Website accessible at http://www.cpsc.ucalgary.ca/projects/grouplab/.

[HCIL 2000] Human-Computer Interaction Lab, University of Maryland: "Welcome to Jazz!"; Website accessible at http://www.cs.umd.edu/hcil/jazz/.

[Lotus Corporate 2000] Lotus Development Corporation: "Lotus® Notes"; Website accessible at http://www.lotus.com/home.nsf/welcome/notes.

[LPI2 2000] Lehrgebiet Praktische Informatik II, FernUniversität Hagen: "DreamTeam Homepage"; Website accessible at http://carmen.fernuni-hagen.de/dreamteam/dreamteam_eng.html.

[Naur and Randell 1969] P. Naur and B. Randell (Editors): "Software Engineering, Report on a Conference Sponsored by the NATO Science Committee"; Scientific Affairs Division, NATO (1969).

[NCSA 1999] National Center for Supercomputing Applications, University of Illinois at Urbana-Champaign: "NCSA Habanero®"; Website accessible at http://havefun.ncsa.uiuc.edu/habanero/.

[OpenAvenue 2000] OpenAvenue, Inc.: "Concurrent Versions System: The Open Standard for Version Control"; Website accessible at http://www.cvshome.org/.

[Roseman et al. 1996] M. Roseman and S. Greenberg: "TeamRooms: Network Places for Collaboration", Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work, ACM (1996), 325–333.

[Sommerville 1996] I. Sommerville: "Software Engineering"; 5th Edition, Addison-Wesley Publishing Company, Inc. (1996).

[Stasko et al. 1998] J. Stasko, J. Domingue, M. H. Brown, and B. A. Price (Editors): "Software Visualization: Programming as a Multimedia Experience"; The MIT Press (1998).

[Sun Microsystems 2000a] Sun Microsystems, Inc.: "Java™ 2 SDK, Standard Edition"; Website accessible at http://java.sun.com/products/jdk/1.2/.

[Sun Microsystems 2000b] Sun Microsystems, Inc.: "Java™ Shared Data Toolkit"; Website accessible at http://java.sun.com/products/java-media/jsdt/index.html.

[Tufte 1990] E. R. Tufte: "Envisioning Information"; Graphics Press (1990).

[WebMagic 2000] WebMagic, Inc.: "USENET.org™"; Website accessible at http://www.usenet.org/.

[WebWisdom.com 1999] WebWisdom.com, Inc.: "TANGO Interactive™"; Website accessible at http://www.webwisdom.com/tangointeractive/.