NISS

WEB-Based Text Visualization

Stephen G. Eick, Audris Mockus, Todd L. Graves,
and Alan F. Karr

Technical Report Number 64
September, 1997

National Institute of Statistical Sciences
19 T. W. Alexander Drive
PO Box 14006
Research Triangle Park, NC 27709-4006
www.niss.org

WEB-Based Text Visualization

Stephen G. Eick[1], Audris Mockus
Bell Laboratories
Todd L. Graves
National Institute of Statistical Sciences and Bell Laboratories
Alan F. Karr
National Institute of Statistical Sciences
{eick,audris,graves}@research.bell-labs.com; karr@niss.rti.org

[1]Correspondence contact: Room 1G-351, Bell Laboratories, 1000 East Warrenville Road, Naperville, IL 60566

Funding for this research was provided in part by NSF-SBR-9529926.

Summary

We describe two prototypical World Wide Web-based tools for visualization of text data. By using the Web, the tools enrich the interaction between the analyst and the data. Both are portable: they are written in Java and run within a standard Web browser. They access data from a very large central data base, instead of requiring that it be downloaded; this leads to significant gains in efficiency, as well as in maintenance (timeliness and security) of the database. Decreased performance resulting from use of interpreted rather than compiled code can be mitigated by means of clever programming.

One tool, SeeSoft™, can display thousands of lines of text on a single screen, allowing detection of patterns not discernible directly from the text.

The second tool, *Live Documents*, replaces static statistical tables in ordinary documents by dynamic Web-based documents, in effect allowing the "reader" to customize the document as it is read.

1 Introduction

In this paper we use two prototype tools to outline and illustrate a strategy to deal with two essential issues: the needs for

- Visualization as a means for dealing with text and other non-traditional forms of data;
- Analysis of data drawn from increasingly commonplace, multi-gigabtye archives, especially when multiple copies are not feasible (for example, because of frequent updates or security considerations).

Our motivating example is the multimillion line source code from a large real-time software system. The source code for this system, developed over the last two decades, is partitioned into tens of thousands of files, thousands of modules and tens of subsystems. The code is dynamic and constantly changing: modifications are submitted daily by the thousands of engineers involved in the project. The database contains the complete change history, including virtually every modification made during the project, as well as many related statistics.

Often, corporate data assets are warehoused centrally, because of their strategic nature and interest to many organizations, and at the same time, these organizations have differing

needs in connection with the data, differing skill levels, and varying tasks to perform. This is true of the source code database, which must be accessed by multiple constituencies for multiple reasons. First, since documentation is often out of date, the code itself is the most reliable source of information about the product. Second, programmers making changes need access to ensure that proposed changes do not affect existing functionality. Third, project managers monitor code changes to assess whether promised features are on schedule and whether delivery deadlines will be met. Finally, engineers study the code to identify bug-prone regions, unused dead code and inefficiencies.

Our approach exploits recent advances in computing and networking technologies, in particular the explosive growth of the World Wide Web, by allowing the data to reside in a central location, but analyses to be conducted locally. The tools we describe are implemented as applets written in Java (Flanagan, 1996) and run within Netscape's Navigator™ Web browser. (They would run equally well in Microsoft's Internet Explorer™). The compelling advantages of browsers are their widespread availability, low cost and robust implementation, coupled with the rapidly forming world-wide infrastructure utilizing Web technology. We believe that Web browsers have the potential to become widely used platforms to combine access to networked databases with the technology to conduct analyses of the data.

One prototype, described in §2, is a applet-based version of SeeSoft™. SeeSoft is a software visualization system for showing thousands of lines of code on a single screen (Eick, 1994) (Ball and Eick, 1996). §3 describes *Live Documents*, a suite of prototype applets for publishing statistical summaries in `HTML` Web pages. Both access data directly from a central code database, so that analyses are updated automatically as changes occur. In addition, the applets incorporate many useful ideas from dynamic statistical graphics.

2 SeeSoft On The Web

Initially written in C++, to be compiled and run on local databases, the SeeSoft system adapts naturally to the Web, where it runs as a Java applet (under Netscape or another browser) on local or remote databases.

Before discussing the Java version, we introduce SeeSoft (Eick, 1994) (Ball and Eick, 1996), a system designed to display large volumes of text. Although developed to visualize source code for large software systems, SeeSoft can be used equally effectively for other kinds of text data. SeeSoft incorporates several representations for text, the most popular of which is the line representation:

- Each file (or other text unit) is labeled and shown in a box. Comparing sizes of boxes allows a user to identify unusually large or small files.
- Each line of code is represented as a row of pixels, with the length of the row encoding the number of characters (preserving both indentation and line length), so that the nesting structure is perceived readily (see Figure 1).
- Lines are color-coded to show a statistic of interest. Examples for source code are the age of the line, its software release, the identity of the programmer who wrote it and an indicator specifying whether the line fixes a bug or adds new functionality.[1]

[1]Proprietary information has been blurred in the figures.

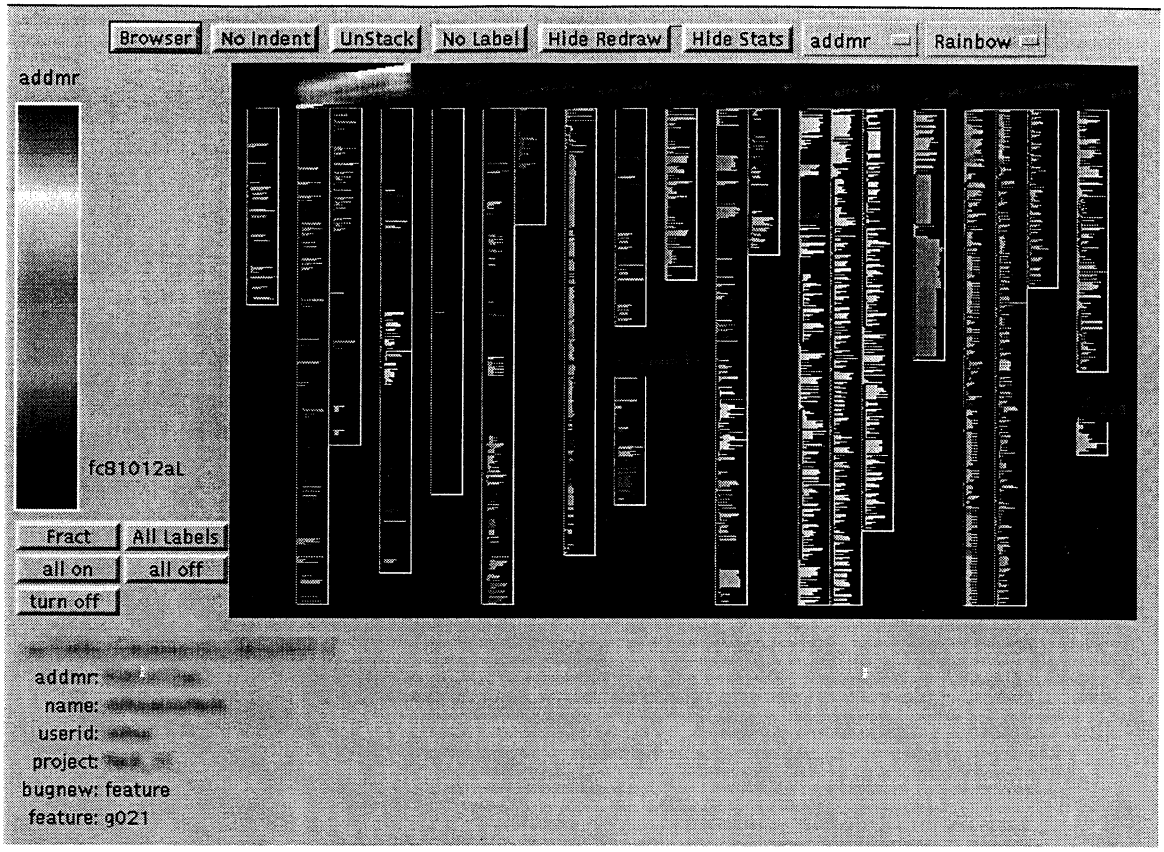

Figure 1: The SeeSoft text view showing code age according to a rainbow color scale.

SeeSoft displays such as Figure 1 are interactive, and employ techniques from dynamic statistical graphics (Becker et al., 1987). For example, a user may turn lines on or off by brushing the mouse over the color scale (see Figure 2), in order to reduce the visual complexity and focus attention. As the mouse touches any line of the code, the line itself and statistic values corresponding to it appear on screen. (This form of dynamic identification is similar to that used in the S language (Becker et al., 1988) for identifying points in a scatterplot.) The statistic used to color lines is user-selected (in Figure 1 it is the age of the code), as is the color palette (`Rainbow` in Figure 1). Other important options include `Browser`, which opens a window (see Figure 2) that shows in a readable font the text beneath its controller (in the middle of the third file).

The WEB interface to SeeSoft, like many Web interfaces to databases, consists of four principal components: a large database maintained on a server, a `CGI-bin` access program written in `Perl` (Schwartz, 1993), a `JavaScript` (Goodman, 1996) interface running in a Web browser, and the SeeSoft applet.

The visual display, illustrated in Figure 2, includes three Netscape `HTML` frames (Musaciano and Kennedy, 1996). The left and bottom frames allow the user to select a subsystem and module for display (the code and statistics for a particular subsystem and module are retrieved by executing the `Perl` running as a `CGI-bin` command (Gundavaram, 1996)), to select from two different versions of the code and to control whether the SeeSoft window is displayed in the right frame or in its own window.

Because Java applets are interpreted within a Web browser, they execute much more

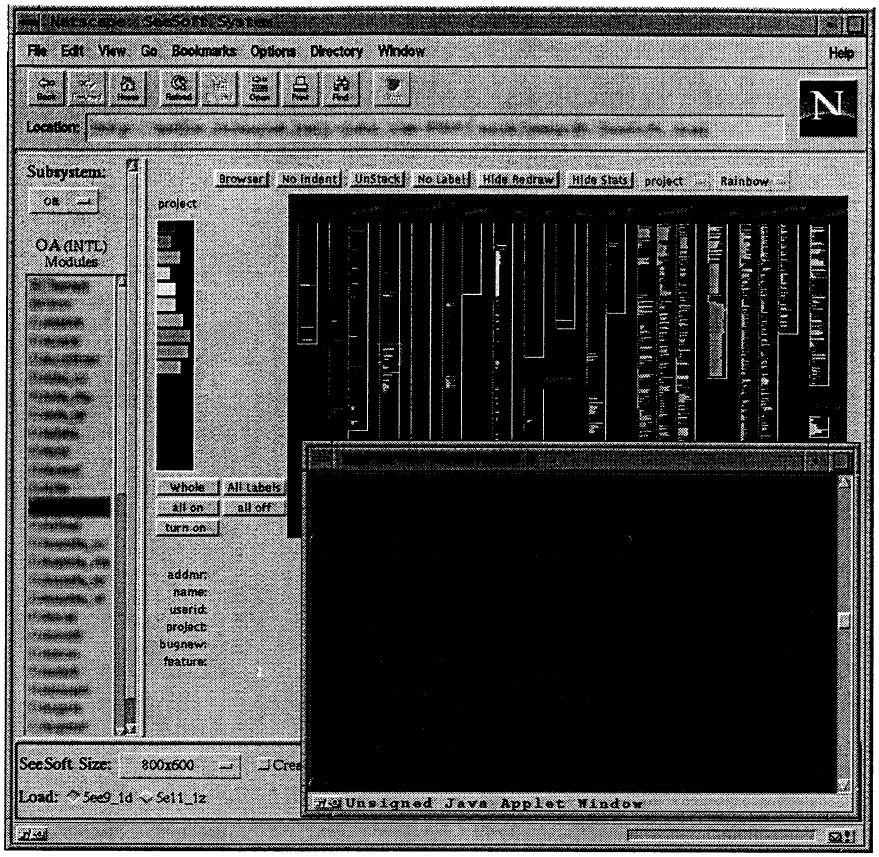

Figure 2: SeeSoft running within Netscape Navigator as an applet with a browser window showing the code text. The color of each line is tied to its version with the middle versions deactivated.

slowly than does compiled C code. Three techniques were used to make the Web interface perform acceptably. First, for run-time efficiency, the `Perl` command precomputes many of the data structures on the server that will be needed by the SeeSoft applet, and caches them for future access. Second, for network efficiency, data is transmitted in compressed binary form. Finally, for a responsive user interface, the applet uses two independently executing threads, one to read the data as it streams in through the network, and the other to display data as soon as it is available, as well as to respond to user requests.

3 Live Documents

Put simply, *Live Documents* are to ordinary, static documents as pictures are to words: if a picture is worth a thousand words, a dynamic picture — the essential feature of a Live Document — may be worth a million words.

3.1 The Concept

Live Documents weave together two key threads:

- The necessity to use graphical displays (such as bar charts, tables, histograms and density plots) to present the results of statistical analyses, through careful selection of

quantities and views that illustrate the salient points;

- The widespread recognition that dynamic displays of statistical data can be more informative than static displays (Cleveland, 1993).

Live Documents, therefore, publish the results of the analyses as Web pages, but replace static figure- and table-based graphics with interactive applets. Readers may manipulate the applets dynamically to obtain detailed, customized information from the graphics. Through this interactive capability, writers of a Live Document may target a wider audience than is possible with traditional publications. Readers can even explore hypotheses not envisioned by the author.

To make Live Documents effective, we apply the following principles from Computer-Human-Interaction (Mullt and Sano, 1995):

- Interactions are simple, intuitive, and reversible;
- Graphical controls are embedded as their use is described;
- Suggested scenarios answer relevant and engaging questions, and suggest further investigations.

To date, we have constructed applets for drawing scatterplots, densities and tables. The table applet (see Figure 3), perhaps the most interesting, follows Rao's Table Lens (Rao and Card, 1994): it shows tables of numeric data with the variable names across the top and values for each observation in row-ordered cells. Three representations of data values are possible, depending on available screen space: as textual numeric digits, as thin bars with lengths proportional to the values, and as a combination of these two, with the digits overplotted on the bars. (The applet chooses the data representation that displays all of the observations most efficiently; at the extreme, each bar is one pixel thick, as in the SeeSoft line representation.) The rows of the table can be sorted to show correlations among the variables. The scrollbar on the left side of the view controls the available screen space and scrolls the table.

3.2 Live Documents in Action

We illustrate Live Documents in the context of changes to code in large software projects.

In very complex software systems deleting code may be seen as risky, because the person performing the deletion may be unaware of dependence of other parts of the system on it. To address this question, Figure 3 shows a table with one row for each developer and columns for (1) the annual number of changes made by a developer (`Count`); (2) the number of lines added in each change (`Added`); (3) the number of lines deleted (`Deleted`), and (4) the number of lines unchanged (`Same`). The table is sorted by `Deleted`.

In Figure 3, the top two bars in the `Deleted` column are considerably longer than the bars below them, indicating that most large deletions have been performed by only two developers. Furthermore, these developers have small values of the `Count` variable, so they made relatively few changes.

Using the table applet allows us interactively to restrict attention to the developers with at least five changes (`Count` ≥ 5). To do this we sort by `Count`, zoom in, read textual values

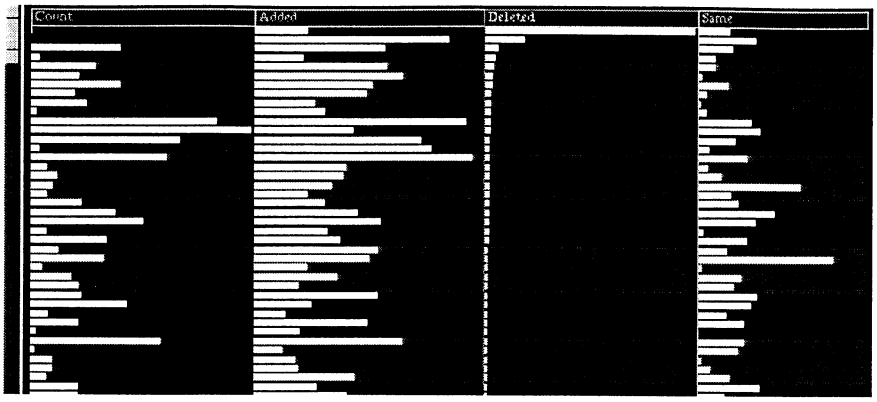

Figure 3: Numbers of lines added and deleted by different developers.

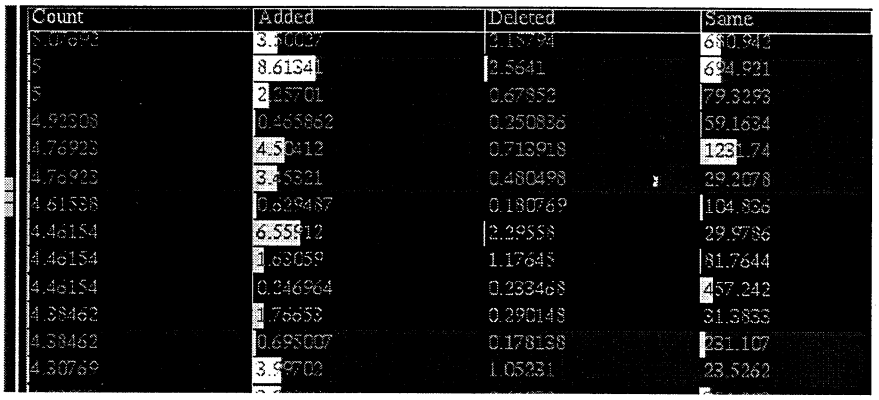

Figure 4: Interactively Sorted and Zoomed Portion of Figure 3.

from the display and select developers making approximately five changes per year, which are highlighted in yellow in Figure 4.

Using similar methods, we can select the subset with `Deleted` at least five, and find that there are twenty such developers within the reduced data set (see Figure 5). By then zooming out and sorting in decreasing count order, we see that many of these twenty developers (still highlighted in yellow) cluster around the top of the count distribution (see Figure 6).

From this analysis we have learned that developers who delete code tend to be those making large numbers of total changes, who are likely, of course, to be among the most experienced and able engineers. More important, there is no need for the author of a report containing the data to have answered or even thought about the questions we have addressed: the Live Document allows us the freedom to ask and answer our own questions. Nor have we ever required physical possession of the data, which can remain secure and timely in a central location.

4 Conclusions

We have presented two prototypes — SeeSoft and Live Documents — involving text visualization within a Web browser. They work on virtually all computing platforms. Given availability of such tools, authors can use the Internet as a mechanism to distribute their results, by publishing the relevant `URL`s.

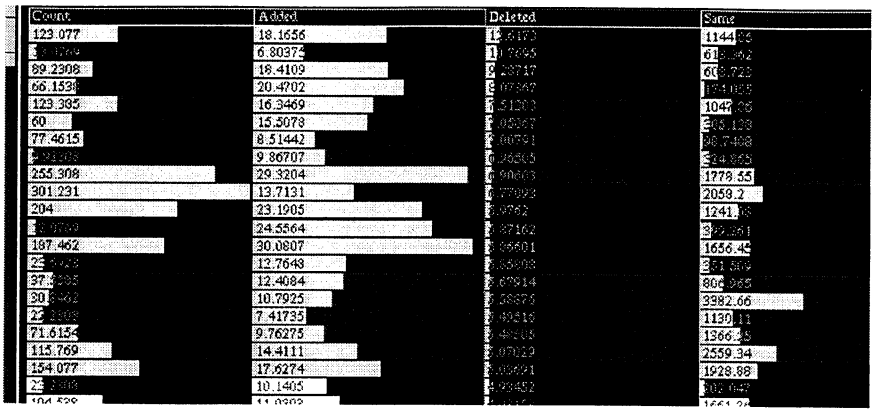

Figure 5: Statistics for Developers Who Make Many Deletions.

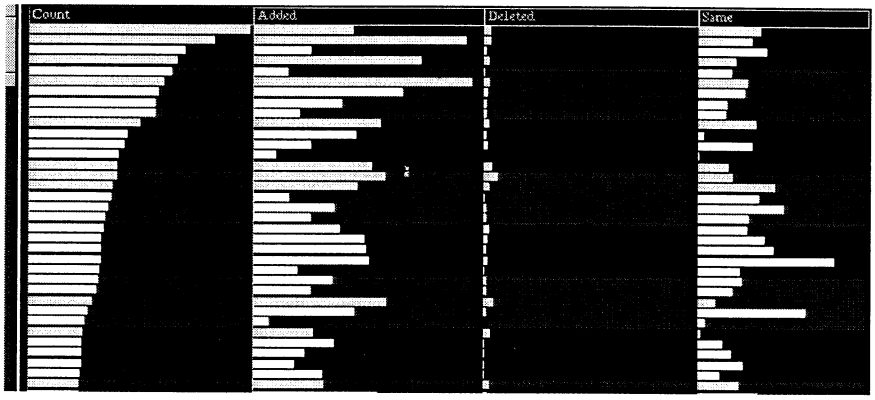

Figure 6: Developers Making Many Deletions are those Making Many Changes.

We speculate that in the future many statistical analyses will be performed in a Web environment. Results will be published in Live Documents, allowing readers to replicate, expand or refine the work done by the authors. Such an environment would prove beneficial for science as a whole and statistics in particular. Especially, questionable interpretations of the data will be more readily subjected to scrutiny. Statistical reasoning will be more in demand, to answer the question, "Is that what the numbers *really* mean?"

Acknowledgments

Thomas J. Ball wrote early versions of some of the applets described in Section 3, based on Graham Wills' C++ code. This research was supported in part by NSF grant SBR–9529926 to the National Institute of Statistical Sciences.

References

Ball, T. A. and Eick, S. G. (1996). Software visualization in the large. *IEEE Computer*, 29(4):33–43.

Becker, R. A., Chambers, J. M., and Wilks, A. R. (1988). *The New S Language*. Wadsworth & Brooks/Cole, Pacific Grove, CA.

Becker, R. A., Cleveland, W. S., and Wilks, A. R. (1987). Dynamic graphics for data analysis. *Statistical Science*, 2:355–395.

Cleveland, W. S. (1993). *Visualizing Data.* Hobart Press, Summit, NJ.

Eick, S. G. (1994). Graphically displaying text. *Journal of Computational and Graphical Statistics*, 3(2):127–142.

Flanagan, D. (1996). *Java in a Nutshell.* O'Reilly & Associates, Sebastopol, CA.

Goodman, D. (1996). *JavaScript Handbook.* IDG Books Worldwide, Inc., Foster City, CA.

Gundavaram, S. (1996). *CGI Programming on the World Wide Web: On-the-Spot Information.* O'Reilly & Associates, Sebastopol, CA.

Mullt, K. and Sano, D. (1995). *Designing Visual Interfaces.* SunSoft Press, Sun Microsystems, Inc., Mountain View, CA.

Musaciano, C. and Kennedy, B. (1996). *HTML The Definitive Guide.* O'Reilly & Associates, Sebastopol, CA.

Rao, R. and Card, S. K. (1994). Table lens: Merging graphical and symbolic representations in an interactive focus plus context visualization for tabular information. In *Proceedings of ACM Conference on Human Factors in Computing Systems (CHI'94)*, pages 318–322, Boston, MA.

Schwartz, R. L. (1993). *Learning Perl.* O'Reilly & Associates, Sebastopol, CA.