

# GRAPHICALLY DISPLAYING TEXT

Stephen G. Eick  
AT&T Bell Laboratories - Room IHC 1G-351  
1000 E. Warrenville Road  
Naperville, Illinois 60566  
  
eick@research.att.com

**Key Words:** software visualization, dynamic graphics, brushing, linking, text statistics

## SUMMARY

There are many examples of text databases including literary corpora and computer source code where statistics are associated with each line. A visualization technique for this class of data represents the text lines as thin colored rows within columns. The position, length, and indentation of each row corresponds to that of the text. The color of each row is determined by a statistic associated with each line. The display looks like a miniature picture of the text with the color showing the spatial distribution of the statistic within the text. Using this technique, *SeeSoft*<sup>TM</sup>, a dynamic graphics software tool, can easily display 50,000 lines of text simultaneously on a high-resolution monitor.

## 1. INTRODUCTION

Statistical graphics techniques have long been used for visualizing data. The traditional techniques are not always useful for depicting non-traditional data, particularly for statistics associated with lines of text. To make this information visually accessible, a graphics technique has been developed embodied in a dynamic graphics system *SeeSoft*<sup>TM</sup>.

Examples of text to which this technique may be applied include (a) a literary corpus, such as the Bible or Shakespeare; (b) a technical paper with multiple authors; (c) records in a database; (d) computer log files; (e) lines in a computer program (the motivating example for the development of the technique). This type

of data consists of files containing lines of text that have values associated with each line. The “lines” may be literally the lines of text or logical entities such as sentences, verses, or records. For a work of literature, the statistics of interest may include word usages or the locations of items in an index. In analyzing a paper jointly written by several authors, it might be helpful to know which author was responsible for a given sentence and the revision number in which it first appeared.

The statistics associated with text may be continuous, categorical, or binary. For a line in a computer program, when it was written is a continuous statistic, who wrote it is a categorical statistic, and whether or not the line executed during a regression test is a binary statistic.

The motivation for this technique comes from studying the change history of the computer source code from AT&T’s 5ESS® switch. This system contains several million lines of code, written by thousands of programmers, over the past decade. The complete history of every change has been captured in a version management database (Rochkind, 1975 and Tichy, 1985). For each modification, the database contains the modified code lines, reason for the change, the pertinent release, responsible developer, type of change (bug fix or new feature), etc.

To keep a large software system functioning, the code must be maintained. This involves reorganizing the code, fixing bugs, and adding new functionality. Analysts involved with code restructuring studies and code archeology must determine when the files have become too complex and should be rewritten. Programmers working on the code would like to know whether the code has been stable, since instability and frequent changes indicate the need for caution. Much of the software maintenance effort involves code discovery, where programmers rediscover how the code actually works. A visual method showing the code history can be a useful tool for helping programmers understand code.

A graphical technique for displaying text represents each file as a vertical column and each line as a color-coded row within the column (see Figure 1). The row indentation and length tracks the corresponding text and the row color is tied to a statistic. If the row tracking is literal as with computer source code, the display looks as if the text had been printed in color and then photo-reduced to fit on a single figure. The spatial pattern of color shows the distribution of the statistic within the text.

SeeSoft is a system implementing this graphical technique and applying dynamic graphics (Becker and Cleveland 1987), particularly brushing and linking, and high-interaction (Shneiderman 1983) to increase its effectiveness. Note that color is critical for this method and the intent is that this paper be read with reference to the accompanying color figures.

The following sections describe this graphical method and display manipulation techniques in more detail. For concreteness, Sections 3, 4, 5, and 6 apply it to visualizing the change history of source code, while Section 7 is general and Section 8 describes SeeSoft's implementation.

## 2. RELATED WORK

For source code, SeeSoft's display looks somewhat like that of Baecker and Marcus (1989, p. 235), but there are several differences. Baecker and Marcus show a reduced representation of C language (Kernighan and Ritchie 1978) code that has been typeset. SeeSoft works with arbitrary text, not necessarily C code, and represents the text lines as rows with no typesetting. The biggest differences involve color, statistics, and interaction. SeeSoft colors the representation to show the distribution of statistics, and makes the technique more effective with interaction.

Each of SeeSoft's columns have a similar appearance to the enhanced scrollbars proposed by Hill, et al. (1992) and Chimera (1992). Chimera proposes "Value Bars" as an enhancement to the usual directory (ls -l). Value Bars place thin vertical strips in a scrollbar on a window containing a directory listing. The bar width codes an attribute of each file, such as its age or size. Only the largest bars are shown, giving a ladder effect with unevenly spaced rungs. Hill, et al. also suggests an improvement to scrollbars in which there are thin lines in the scrollbar, with the length coding the number of times the corresponding text line has been read or modified. The idea is that the lines in the scrollbar indicate the amount of "edit wear and read wear" in the document.

The gray scale version of SeeSoft was first introduced by Eick, Steffen, and Sumner (1992) as a software engineering tool. This paper generalizes the previous results to arbitrary text databases and emphasizes the use of color.

Turning off indentation gives each row the same visual prominence, making it easier to see the colors of the short rows. In Figure 2 indentation is turned off and the mouse has moved. The red lines were written in

1992, the yellow lines in the middle of 1991, and the blue lines in 1990. Files in a single color have been stable and those in many colors have been unstable. The left-most file `TM15min_ex.c`, for example, has some old blue code, medium aged yellow code, and some recent code in red.

Browser windows (see Figures 1 and 2) enable a programmer to read the actual code and move smoothly between the condensed SeeSoft representation and the code. For each browser window there is a small colored rectangle on the graphics display that functions as a virtual “magnifying” box. In Figure 1 the magnifying box is positioned toward the bottom of file `TMdpm_trg.c` and in Figure 2 the box is tracking the mouse. The code underneath the magnifying box is displayed in the browser window. The height of the magnifying box is proportional to the size of the browser window, telling the programmer what fraction of the total is visible. Multiple browser windows may be created and independently positioned. For easy identification, the border color on each browser window is the same as its corresponding magnifying box.

#### 4. DYNAMIC GRAPHICS TECHNIQUES

There are five interactive views of data on SeeSoft’s display:

1. the rows corresponding to the text lines,
2. the values in the color scale,
3. the file names above the columns,
4. the browser windows, and
5. the bar charts underneath the color scale.

The views are linked, united through the use of color, and activated by touching with the mouse. As the user moves the mouse around the display, the computer tracks its position. Whenever the mouse touches a row, a value in the color scale, or a file name, all rows corresponding to this entity become active. This reduces the traditional point and click interface to just pointing. The activation may be permanent or temporary, but only active rows are displayed. All rows are active in Figures 1 and 2 since all are visible. The thin red horizontal bars underneath `64/64` and `Lines: 13589/13589` are bar charts showing the fraction of active changes and active lines (100% in both cases).

This mode of manipulating the display, called “brushing” by Becker and Cleveland (1987) and Becker, Cleveland, and Wilks (1987), is particularly effective for exploratory data analysis (Cleveland and McGill 1988). General techniques for linking multiple views are explored by Buja et al. (1991). An extension to SeeSoft might include other linked views such as time-series plots or scatter plots. A practical difficulty in adding more linked views to SeeSoft is a lack of screen real-estate. When displaying large bodies of text, the current SeeSoft display utilizes nearly every pixel on a high-resolution monitor, leaving few pixels available for additional views.

#### 4.1 Linking Color Scale, Rows, And Names

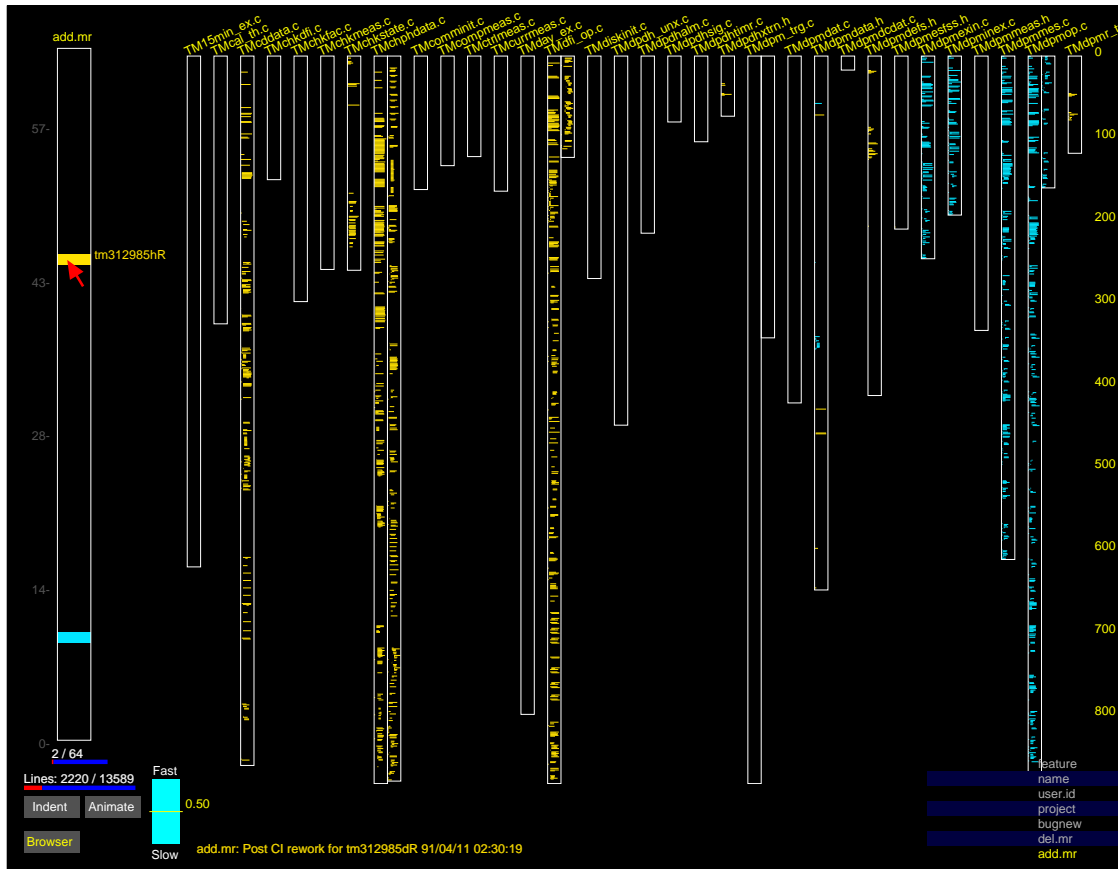
Touching a value in the color scale with the mouse activates the value and all of its corresponding rows. Conversely, touching a row activates the row, its corresponding value in the color slider, and all other rows associated with this value. Touching a file name activates all rows in the file, any values in the color scale corresponding to any of these rows, and any rows in other columns associated with the active values.

Figure 3 shows an example where two values in the color scale are active. The lower value is permanent and the upper is temporary. These MRs are two major enhancements done during different years, affecting different files, and in total involving 2,220 lines of code. In Figure 4 the mouse is touching a line in file `TMcommunit.c`. The MR adding this line touched 423 lines of code in 6 files. In Figure 5 the mouse is touching a file name, activating all rows in this file and rows in other files added by the active set of MRs.

#### 4.2 Multiple Statistics and Conditioning

A user selects one of the available statistics by clicking on its name in the lower right-hand corner. In Figure 6 the statistic determining the row color is the *user.id* of the programmer adding each line. Seven different programmers have worked on this code. An interesting pattern is displayed in file `TMdpmmeas.h`, the fourth file from the right. There are alternating patterns of red and blue rows, indicating that this file has been jointly written by two *user.id*'s, `bkershaw` and `shardy`. Four of the seven programmers have changed file `TMdpmop.c`, the second file from the right, although most of the code is written by `hvmpla!pplant`.

An interesting statistics is *bugnew*, a binary statistic indicating whether a change fixes bugs or adds new



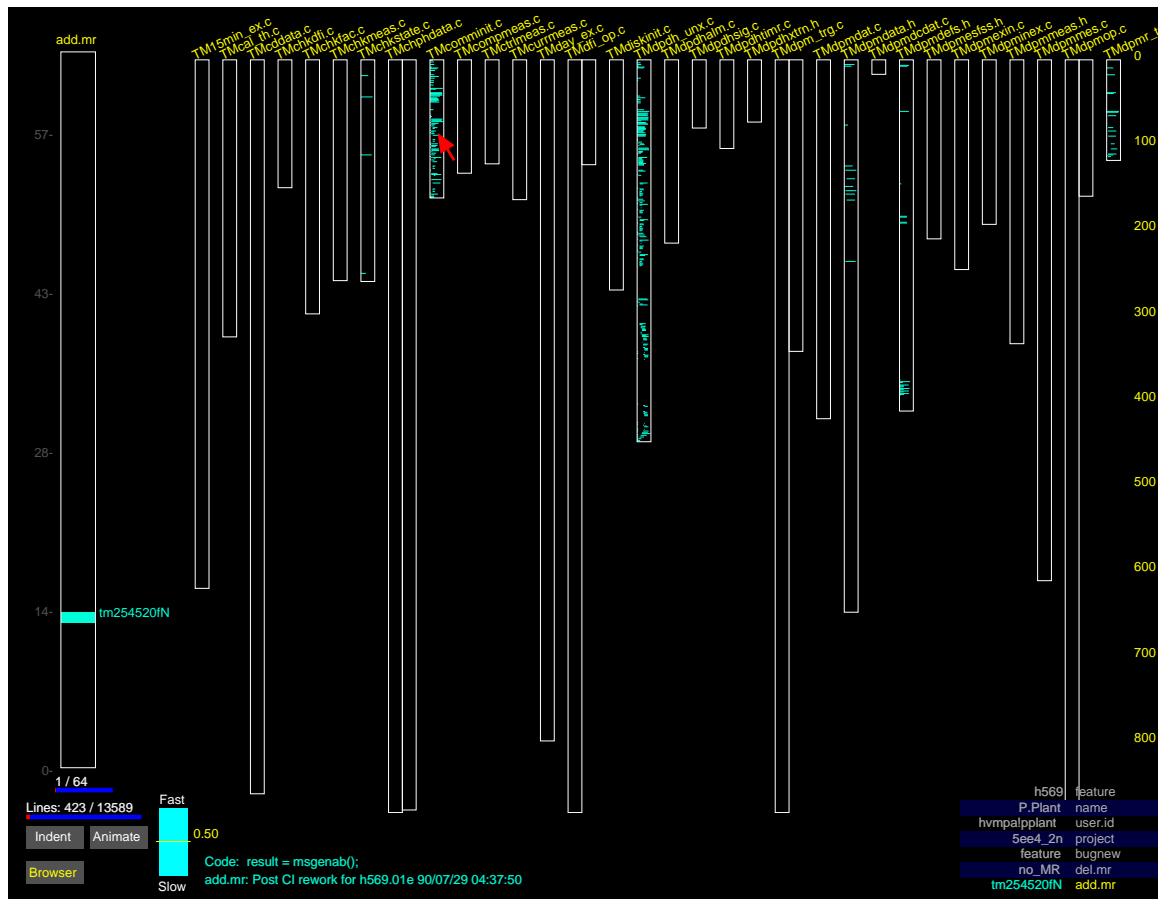
**Figure 2.** Brushing The Color Scale

As the mouse moves along the color scale, the color corresponding to the MR underneath the mouse is momentarily activated. This turns on all rows corresponding to the lines of code added by this MR.

functionality. Figure 7 shows the 431 lines added for bug fixes and the locations of each fix. The row color is determined by the *user.id* of the programmer fixing the bug. The four distinct colors on Figure 7 indicate that four of the seven programmers working on this code have fixed bugs. File `TMchphdata.c` has had the most bugs, and they have all been fixed by programmer `pplant`.

Some information involves two statistics such as “Which programmer added the most code to fix bugs?” In Figure 7 *bugnew* is a conditional statistic and its name is shown in white. Conditional statistics restrict the displayed rows. Depending on the complexity of the information being sought, there may be more than one conditional statistic. The active rows are those rows that are active in the statistic and all conditional statistics.

Causing a conditional statistic to blink is often an effective display manipulation technique. This is known as alternagraphics (Tukey 1973) and has been used by astronomers to identify new comets and asteroids.



**Figure 3. Brushing The Rows**

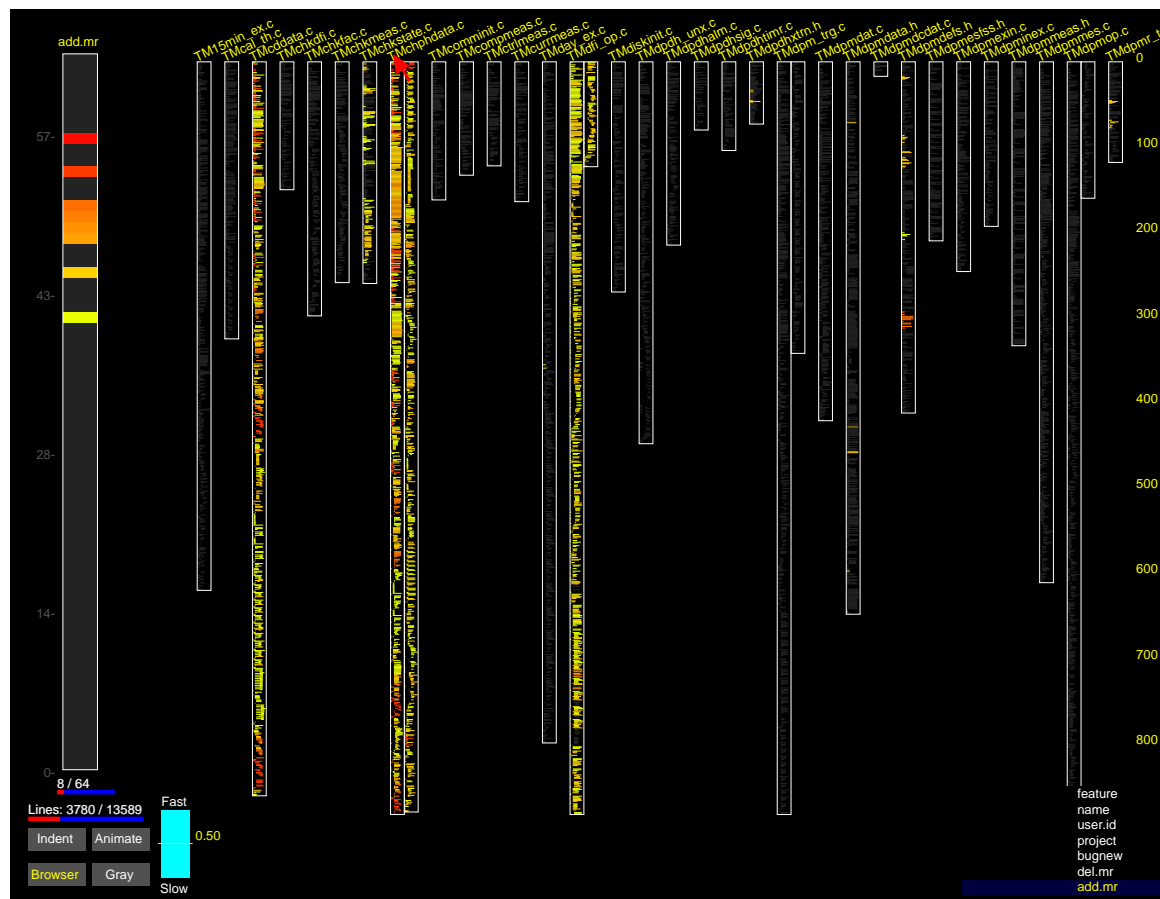
As the mouse touches a row, the MR that added the corresponding line of code is activated. Activating this MR activates all other lines added with the same change. This change added lines in six files.

Clicking the right mouse button on a statistic causes it to toggle on and off as a conditional statistic. This causes the rows conditionally affected by this statistic to blink, giving a vivid visual display.

### 4.3 Animation

Animating over the values in the color scale is useful for scanning across large numbers of statistic values in search of interesting patterns. The Fast-Slow slider controls the animation speed. There are two animation modes: *sequential* and *drag*. In the sequential mode each value is activated and deactivated in turn, and in the drag mode each value is activated and left on until all values are active. In the sequential animation mode, large changes are visually apparent from the bursts of colors that appear during the animation. Drag animation mode is particularly helpful when the statistic is related to time, and shows one





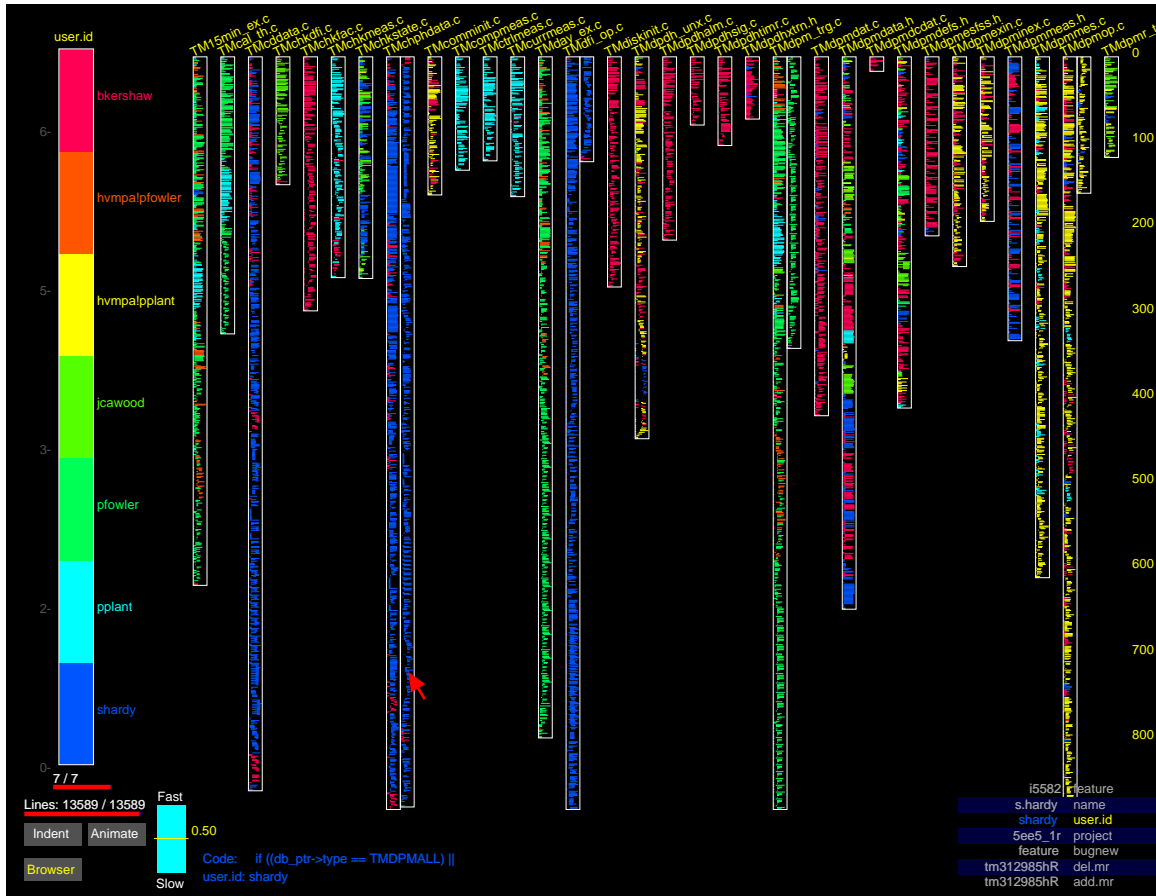
**Figure 4.** Brushing The Names

As the mouse touches a file name, all rows in the file are activated, any MR adding any of these lines, and any other rows added by any of these MRs.

view of the code evolution.

## 5. DATA ANALYSIS USING SEESOFT

The following is a sample SeeSoft data analysis session. Figures 1 and 2 provide an overview of the code. They show the sizes of the files, number of changes in each file, which code is old, which is new, and what code is stable. In addition, they show where the recent changes have been and what files have been changed the most. Several are “rainbow” files, indicating that they have been changed many times. File `TMchphdata.c` has the most bug-fixing code. The heavily changed files may be candidates to be rewritten in the future and are likely spots for future bugs.

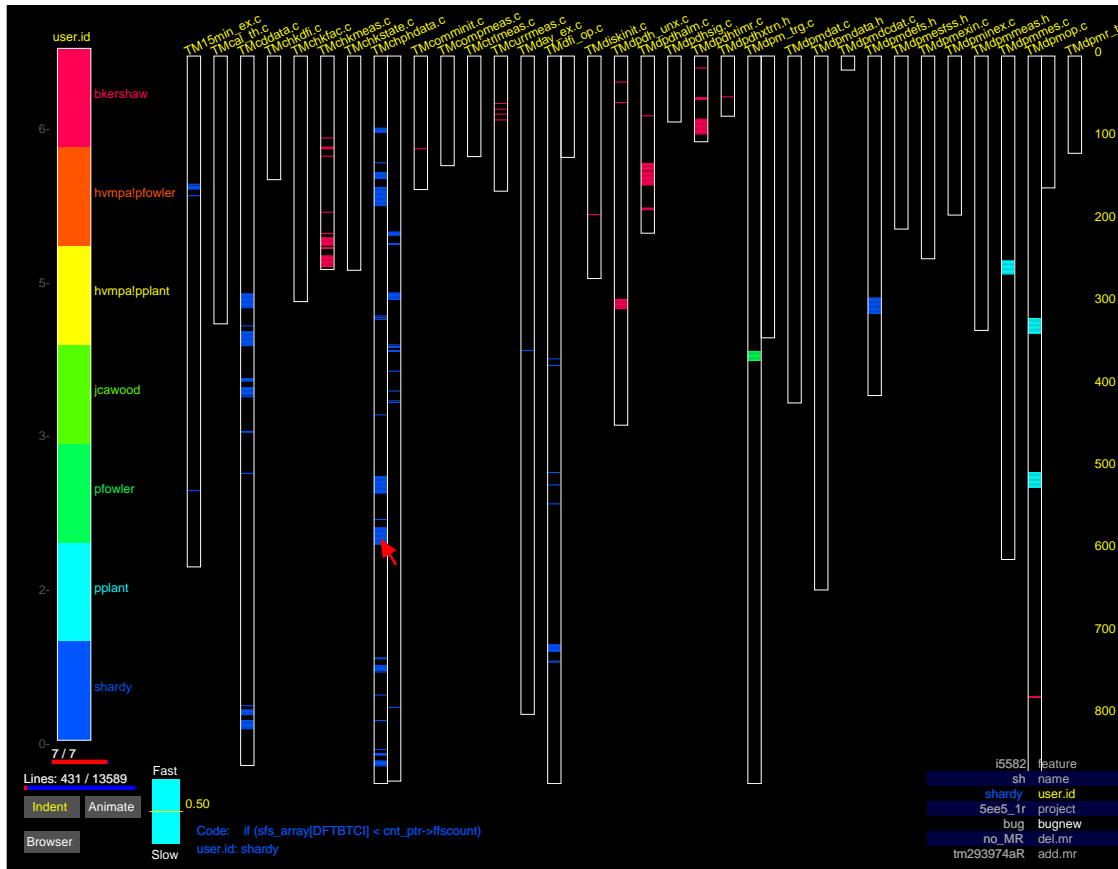


**Figure 5.** User IDs of Programmers Making Changes

Each of the seven *user.ids* in this directory is assigned a color. The rows corresponding to the lines that each programmer added are shown in that programmer's *user.id* color.

When animation is applied over all the MRs, the flashes of color show that there have been three major enhancements in this directory. These enhancements correspond to three successive releases of the software for versions 5ee4\_2n, 5ee5\_1r, and 5ee5\_2f. The blue files are from the earliest release, and each release required between three and six thousand lines of code and added new functionality.

Where have the bugs been? Figure 7 shows that there have been a few bugs in this code that are distributed out over several of the files. Who wrote the code? Brushing the mouse over the *user.ids* in Figure 6 shows that shardy wrote 4,110 lines and bkershaw wrote 3,159 lines. Together they have written more than half of the code. Shardy's code is almost entirely for release 5ee5\_1r, the middle release, with a few lines for 5ee4\_2n. Bkershaw, however, wrote much of the code for release 5ee4\_2n, a few lines for 5ee5\_1r, and much of the code for 5ee5\_2f.



**Figure 6.** Code Fixing Bugs

The rows corresponding to lines added to fix bugs are active. The color of each row indicates the *user.ids* inserting the corresponding bug-fixing line.

Six facts have been learned from this quick analysis:

1. All the code in this directory is new since the first files were added in May, 1990. (This original project started back in the late 1970s.)
2. It was written by seven programmers, but two of the programmers wrote more than half of the code.
3. Code was added to this directory in chunks for three major releases.
4. For each of these releases, new files were added to implement new functionality, and there were some changes to the base code.
5. The code has had a few bugs that have been fixed in each release.
6. Some of the early files have been heavily changed and might be trouble spots in the future.

## **6. GRAPHICAL PRINCIPLES**

Nine aspects of the SeeSoft display make it effective for analyzing text data.

### **6.1 Reduced Representation**

The representation of code lines as rows in columns preserves the context of the original code. It enables programmers to see large volumes of the code on a single screen, avoiding the overlap problem that occurs with scatterplots when plotting tens of thousands of points. Because the display looks like code, programmers are already familiar with it and understand it immediately with no training. Effective visualizations preserve context by presenting the data so that analysts with problem domain understanding can apply their knowledge to interpret the display.

### **6.2 Line Coloring By Statistic**

The line coloring layers information onto the base display. The color is unobtrusive and does not interfere with understanding the rows.

### **6.3 Dynamic Graphics Interface**

Using dynamic graphics, a data analyst presents queries graphically using the mouse and they are resolved instantly and the results presented visually. This direct-manipulation, human-computer interface is natural and effective (Shneiderman 1983).

### **6.4 High-Interaction Color Scale**

The SeeSoft color scale is a generalization of traditional sliders. Each value of the statistic is associated with a particular region on the scale. As the analyst moves the mouse along the scale to select values, the rows corresponding to the values become active in the display. This scale extends the traditional slider since continuous regions, discontinuous regions, or even single values may be selected. The color scale could be enhanced to show the distribution of the statistic, either as a barplot or as smoothed density (Eick, 1993), or could be made two dimensional.

## **6.5 Browser Windows**

The browser windows provide a transparent link between the “birds eye” view and the code. As with many statistical datasets, the ultimate interest is in the text itself. The SeeSoft display functions as a “road map,” making it easy to find certain parts of the code. By opening multiple browser windows, it is easy to compare different parts of the text.

## **6.6 High Information Density**

The information density of a SeeSoft display approaches that of a map (Tufte, 1990). One objective of the representation is to use every available pixel to display information. With SeeSoft’s compact representation of text, 35 files containing 50,000 lines of code can comfortably fit on a standard high-resolution (1280x1024) workstation color monitor. As many as 100,000 lines can be displayed, but the columns become very thin. For each row it is possible to code three statistics in the row indentation, length, and color. It is natural for text that the indentation and length of each row correspond to that of its associated line. This technique scores well on Tufte’s data-ink ratio (1983). The high information density makes the displays visually interesting.

## **6.7 Multiple Linked Views**

SeeSoft presents multiple views of the data and links the views using color. The views are the row display, text in the browser window, file names, the values in color scale, and bar charts underneath the color scale. The views give different insights into the data and each gives insight into the others.

## **6.8 Graphical Conditioning Between Statistics**

Conditioning allows an analyst to obtain information involving two or more statistics.

## **6.9 Animation**

SeeSoft animates over the values in the color scale. Animation may be performed automatically or manually with the mouse.

## 7. EXAMPLES OF TEXT ANALYSIS USING SEESOFT

SeeSoft can visualize any data stored as records within files for which there are statistics associated with each record. The following three examples show SeeSoft displays of code coverage, word usage in the King James Bible, and SeeSoft itself.

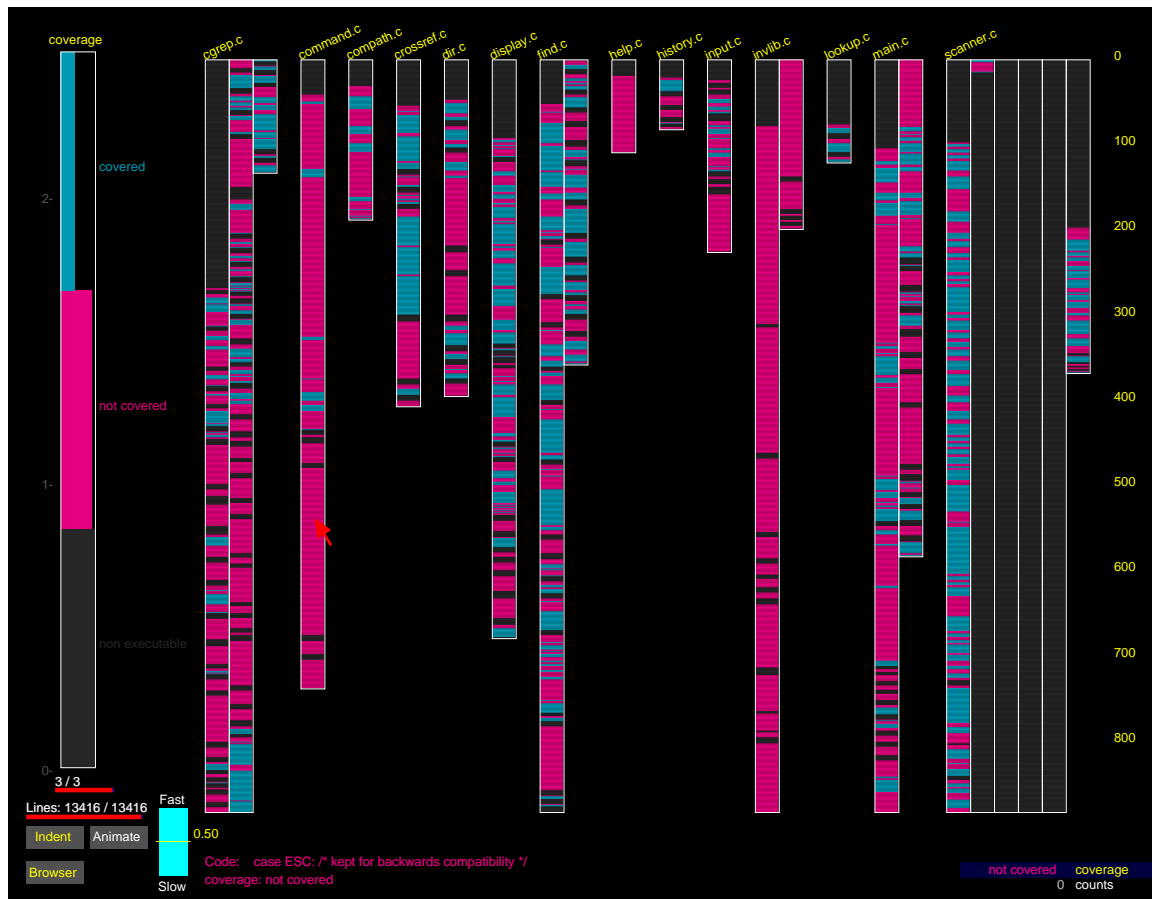
### 7.1 Test Suite Code Coverage

In software testing, the fraction of the executable statements that a test plan executes is called the coverage ratio. A high coverage ratio is a necessary but not a sufficient requirement for a good test plan. Figure 8 shows the statement coverage for the UNIX utility `cscope` (Steffen, 1985), when run through a typical run (Eick and Steffen, 1992). The thicknesses of the colors in the color scale represent the number of lines in each category. The gray color (nonexecutable lines) has full thickness and corresponds to 5,852 lines. The red color (noncovered lines) extends almost the full width and corresponds to 5,198 lines. The blue color (covered lines) extends about 1/3 of the full width and corresponds to 2,375 lines. Thus the test executed only 2,375 of the 7,564 executable statements, a poor coverage ratio of less than 1:3.

### 7.2 A Text Corpus: The King James Bible

Figure 9 shows word usage patterns in the King James Bible. Each row corresponds to one verse in the New Testament. The color of each row indicates whether any one of a particular set of preselected words appears in the corresponding verse. The browser window is showing some verses in 1 Corinthians.

What is interesting about Figure 9? The whole New Testament fits easily onto a single screen. The first five books in the new testament, Matthew, Mark, Luke, John, and Acts are all about the same length. These books were originally written to fit on a single scroll. The other books are much shorter. By using the interactive capabilities of SeeSoft, one discovers that the word *angel*, for example, occurs 108 times. It is used extensively at the beginning of Luke and in Hebrews. *Adultery* is used 17 times and appears in Mathew, Mark, Luke, and John, but not in Acts. A concordance of any extent desired can be easily created with this technique.



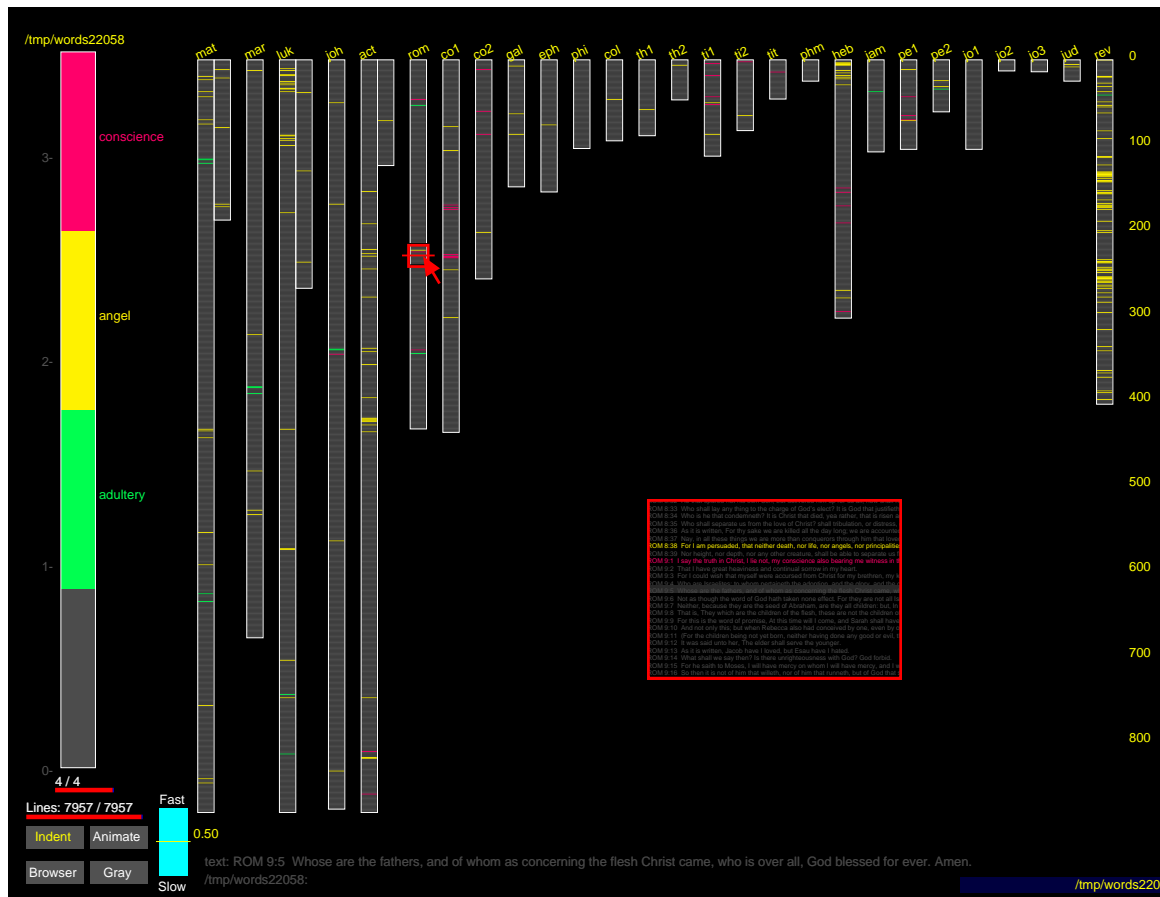
**Figure 7.** Test Suite Coverage

Lines executed by the test suite are shown in blue, nonexecuted lines are shown in red, and nonexecutable lines are shown in gray. The thickness of the colors in the color scale on the left corresponds to the number lines in each category.

### 7.3 SeeSoft Showing Itself

Figure 10 shows SeeSoft displaying itself. The color of each row is determined by the C++ language type of the line.<sup>1</sup> Each line is classified as a symbol reference, preprocessor statement, function definition, variable or function declaration, function call, comment or white space. At the top of each file there is a gray block of comments containing copyright information. The largest file is `seesoft.c` which contains SeeSoft's redraw loop. Other large files are `postseenet.c` and `main.c`. File `postseenet.c`

1. The original version of the awk program calculating the C++ line type was written by Joe Steffen.



**Figure 8.** New Testament

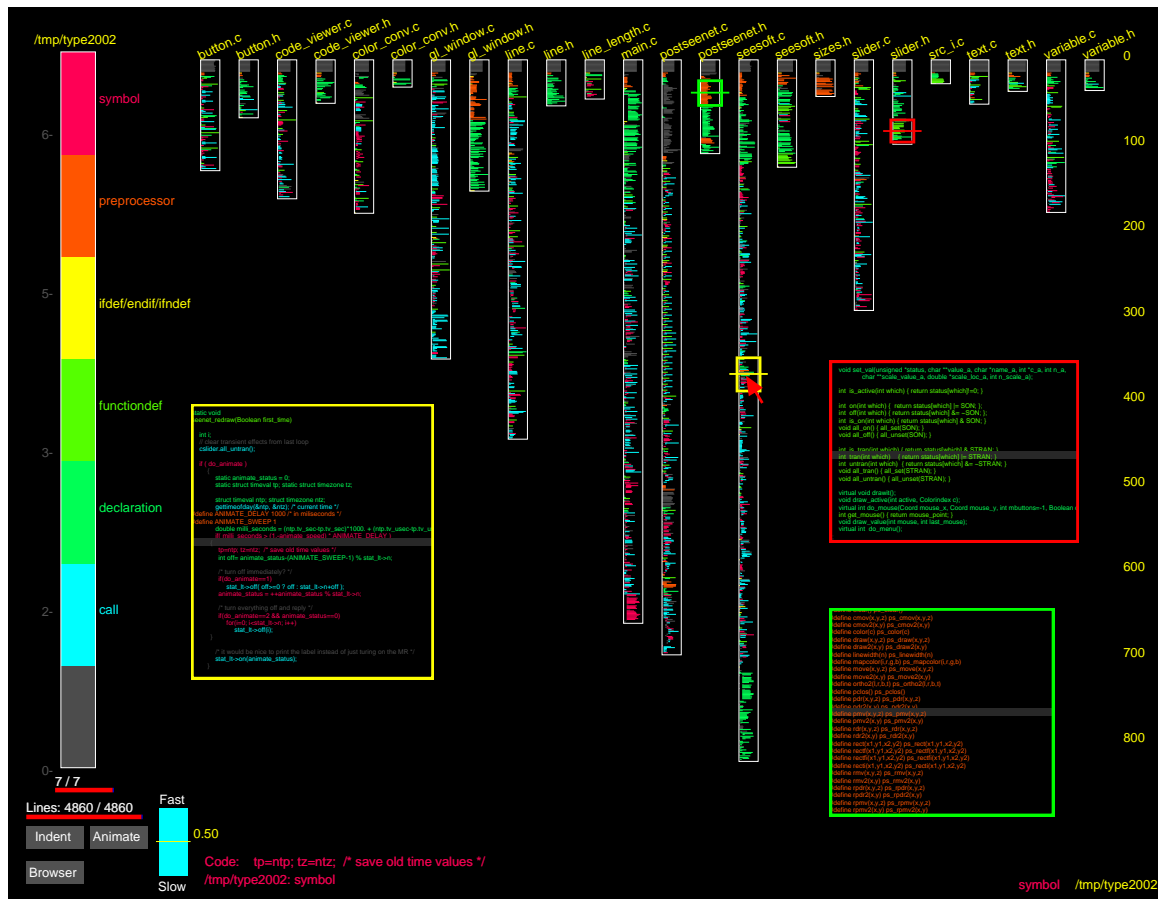
Each line corresponds to one verse in the King James Bible. The locations of selected words are indicated by color. The word `angel`, for example, occurs 108 times. Angel is used throughout the first five books and extensively at the beginning of Hebrews.

creates the PostScript (Adobe Systems 1987) files for the figures and `main.c` allocates memory and handles command line arguments. The orange and yellow lines are preprocessor statements. The `.h` files contain light green function declarations. The dark green lines are variable and function declarations, and the blue lines are function calls.

## 8. SEESOFT'S IMPLEMENTATION

SeeSoft is currently a 4,860 line C++ program running on Silicon Graphics Iris workstations using the GL graphics library (Silicon Graphics, Inc. 1990) and on X workstations using the MOTIF <sup>2</sup> widget set.





**Figure 9.** SeeSoft Displaying Itself

A SeeSoft display of C++ code with the color showing the C++ language line type. The C++ code is SeeSoft itself.

Similar code could run on any color workstation or personal computer, having a large, high-resolution color monitor.

For high performance, SeeSoft uses color map animation (Foley, et al. 1990). SeeSoft draws each row in its appropriate color and then manipulates the system color map. The colors for the activated rows are turned on and those for the deactivated rows are turned off. Color map manipulation is fast because it is done in hardware. Standard workstations have 8 bit frame buffers and can display 256 different colors. Displaying more than 256 statistic values requires a large color map, but in practice, 256 is plenty as it is

2. The port to X and MOTIF was done by Graham Wills.

hard to discern more than 256 colors on a single display. If there are more values than available colors, color zooming can help. The analyst can activate selected regions of the color scale, and SeeSoft will reallocate its color scale assigning colors only to the currently active statistic values. In this way the analyst can “zoom” in on interesting parts of the color scale.

SeeSoft uses C++’s object oriented capabilities (Coplien 1992) to simplify programming. All entities on the screen are represented as C++ classes and inherited from a base entity class. The redrawing and mouse manipulation is done by executing the virtual member function for the appropriate entity on the screen.

The figures in this paper were created from PostScript files. When SeeSoft’s PostScript option is selected, SeeSoft redraws the display and simultaneously writes color PostScript code into a file for each GL graphics call. Then, by prepending a PostScript preamble, it creates an encapsulated PostScript file that is an exact replica of the current SeeSoft display. The mouse is drawn last as a PostScript arrowhead. The browser windows are separate GL graphics windows that can be independently resized, positioned, and iconified. The figures showing both the reduced row representation and the browser windows were created by hand merging the separate PostScript files.

## 9. SUMMARY

This paper describes a new graphical technique for visualizing statistics associated with text. The graphical technique involves displaying lines of text as colored rows withing columns. The position, indentation, and line length of the rows tracks the text. The technique is useful for visualizing text databases of computer source code, including the software change history, test suite code coverage, and word usages in a text corpus. In general, it can be used for visualizing any ordered database of records.

## Acknowledgments

I would like to gratefully acknowledge the help of Richard A. Becker and Eric E. Sumner Jr, and to thank several of my colleagues and some anonymous referees for reviewing early drafts of this paper.

## REFERENCES

1. Adobe Systems Incorporated (1990). *PostScript Language Reference Manual*, Second Edition,

Addison-Wesley, Reading, Massachusetts.

2. Baecker, R. M. and Marcus, A. (1989). *Human Factors and Typography for More Readable Programs*, Addison-Wesley, Reading, Massachusetts.
3. Becker, R. A., Cleveland, W. S., (1987). “Brushing Scatterplots,” *Technometrics*, Vol. 29, pp. 127–142.
4. Becker, R. A., Cleveland, W. S., and Wilks, A. R. (1987). “Dynamic Graphics For Data Analysis,” *Statistical Science*, No. 2, pp. 355–395.
5. Buja, A., McDonald, J. A., Michalak, J., Stuetzle, W. (1991). “Interactive Data Visualization using Focusing and Linking,” *Visualization '91 Conference Proceedings*, San Diego, California, pp. 156–163.
6. Cleveland, W. S. and McGill, M. E. (Eds.) (1988). *Dynamic Graphics for Statistics*, Wadsworth & Brooks/Cole, Pacific Grove, CA.
7. Chimera, R. (1992). “Value Bars: An Information Visualization and Navigation Tool for Multi-attribute Listings,” *Chi '92 Conference Proceedings*, Monterey, CA, pp. 293–294.
8. Coplien, J. O. (1992). *Advanced C++ programming styles and idioms*, Addison-Wesley, Reading Massachusetts.
9. Eick, S. G., Steffen J. L., and Sumner, E. E. (1992). “ISesoft—A Tool For Visualizing Software,” *IEEE Transactions On Software Engineering*, Vol. 18(11), pp. 957–968.
10. Eick, S. G., Steffen (1992). “Visualizing Code Profiling Line Oriented Statistics,” *Visualization '92 Conference Proceedings*. Boston, Massachusetts, pp. 210–217.
11. Eick, S. G. (1993). “Interactive Color Scales,” Submitted for publication.
12. Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F. (1990). *Computer Graphics Principles And Practice*, Second Edition, Addison-Wesley, Reading Massachusetts.
13. Hill, W. C, Hollan, J. D., Wroblewski, D., McCandless, T. (1992). “Edit wear and read wear: their theory and generalizations,” *CHI '92 Conference Proceedings*.

14. Kernighan, B. W. and Ritchie, M. (1978). *The C programming language*, Prentice-Hall, Englewood Cliffs, New Jersey 07632.
15. Rochkind, M. J. (1975). “The Source Code Control System,” *IEEE Transactions on Software Engineering*, Vol. SE-1(4), pp. 364–370.
16. Shneiderman, B. (1983). “Direct Manipulation: A Step Beyond Programming Languages,” *IEEE Computer*, Vol. 16(8), pp. 57–68.
17. Silicon Graphics, Inc. (1990). “Graphics Library Reference Manual,” Document 007-1203-040.
18. Steffen, J. L. (1985). “Interactive Examination of a C Program with Cscope,” *USENIX Dallas 1985 Winter Conference Proceedings*, USENIX Association, El Cerrito, California, pp. 170–175.
19. Stroustrup, B. (1987). *The C++ Programming Language*, Addison-Wesley, Reading Massachusetts.
20. Tichy, W. F. (1985). “RCS—A System for Version Control,” *Software—Practice and Experience*, Vol. 15(7), pp. 637–654.
21. Tufte, E. R. (1983). *The Visual Display of Quantitative Information*, Graphics Press, Cheshire, Connecticut.
22. Tufte, E. R. (1990). *Envisioning Information*, Graphics Press, Cheshire, Connecticut.
23. Tukey, J. W. (1973). “Some thoughts on alternagraphic displays,” Technical Report Number 45, Series 2. Princeton University Department of Statistics, Princeton, New Jersey.