Seesoft—A Tool For Visualizing Line Oriented Software Statistics

Stephen G. Eick, *Member, IEEE,* Joseph L. Steffen, and Eric E. Sumner, Jr.

Abstract— **The Seesoft® software visualization system allows one to analyze up to 50 000 lines of code simultaneously by mapping each line of code into a thin row. The color of each row indicates a statistic of interest, e.g., red rows are those most recently changed, and blue are those least recently changed. Seesoft displays data derived from a variety of sources, such as**

- **version control systems that track the age, programmer, and purpose of the code, e.g., control ISDN lamps, fix bug in call forwarding;**
- **static analyses, e.g., locations where functions are called; and**
- **dynamic analyses, e.g., profiling.**

By means of direct manipulation and high interaction graphics, the user can manipulate this reduced representation of the code in order to find interesting patterns. Further insight is obtained by using additional windows to display the actual code. Potential applications for Seesoft include discovery, project management, code tuning, and analysis of development methodologies.

Index Terms—**Change management systems, code browsing, interactive graphics, line oriented statistics, scientific visualization.**

I. Introduction

A DIFFICULT problem in software engineering is understanding statistics collected at the source code line level of detail. This class of statistics includes information such as who wrote each line, when it was last changed, whether it fixes a bug or adds new functionality, how it is reached, how often it is executed, and so on. The problem is hard for large systems because of the volume of code. A moderately sized system may have thousands of lines of code and a large system may have millions of lines resulting in a large statistical data set. This paper describes a remarkable visualization technique to analyze line oriented data.

Line level data is available on all large software systems. It comes from version control systems, static analyzers, code profilers, and project management tools. This data, however, is underutilized because it is difficult to analyze. Version control systems such as the Revision Control System (RCS) [1], Source Code Control System (SCCS) [2], Change Management System (CMS) [3], Extended Change Management System (ECMS) [4], or SABLE [5] contain a complete history of the code. For each change to the software they typically capture information such as the affected lines, reason for the change, date, and responsible programmer. Static analyzers such as CIA [6] and cscope [7] capture the definitions of functions, types, macros, external variables, etc., and where they occur in the code. Profilers such as lcomp [8] perform basic block counting, indicating how often individual lines are executed.

Because of the volume of code it is difficult to gain insight from line oriented statistics or to get a perspective on the whole system. Statistical analysis techniques often involve aggregation. For many purposes, however, there is a need for finer grain detail. In addition, aggregation techniques discard the familiar and rich textual representation of the code. Code browsers, code formatting techniques [9], and version editors [10] are useful, but none of these generalizes to study arbitrary line oriented statistics.

Our approach to studying this class of data is to apply Scientific Visualization techniques [11]. We refer to this as *Software Visualization*. There is a distinguished history of visualization research starting with Tufte's seminal work [12]. Previous visualization work has involved traditional statistical data. Some notable examples include *MACSPIN* [13], scatter plot brushing [14], and dynamic graphical methods for analyzing network traffic [15]. Unfortunately, none of these methods is tailored for studying line oriented software data. We know of no techniques for studying this class of data that takes advantage of the underlying textual representation of software.

This paper describes a new technique for visualization and analysis of source code, and a software tool, Seesoft, embodying the technique. There are four key ideas: reduced representation, coloring by statistic, direct manipulation, and capability to read actual code. The reduced representation is achieved by displaying files as columns and lines of code as thin rows. The color of each row is determined by a statistic associated with the line of code that it represents. In several of our examples the statistic will be the date that the line was created. The visual impression is that of a miniaturized copy of the code with color depicting the age of the code. Then, using direct manipulation and high interaction graphics, a user manipulates the display to find interesting patterns. To display the actual code text the user opens up reading windows and positions virtual magnifying boxes over the reduced representation.

Fig. 1 shows a display of a directory containing 20 source code files containing 9 365 lines of code. The height of each column tells the user how large each file is. Files longer than one column are continued over to the next column. For the display, the line color[1] shows the age of each line using a rainbow color scale with the newest lines in red and the oldest

Manuscript received October 1, 1991; revised August 1, 1992. Recommended by R. Selby and K. Torii.

The authors are with AT&T Bell Laboratories, Naperville, IL 60566.

IEEE Log Number 9203763.

[1] In black and white versions of this paper color is to be interpreted as gray level. Red is equivalent to dark grey, green to medium gray, and blue to light gray.

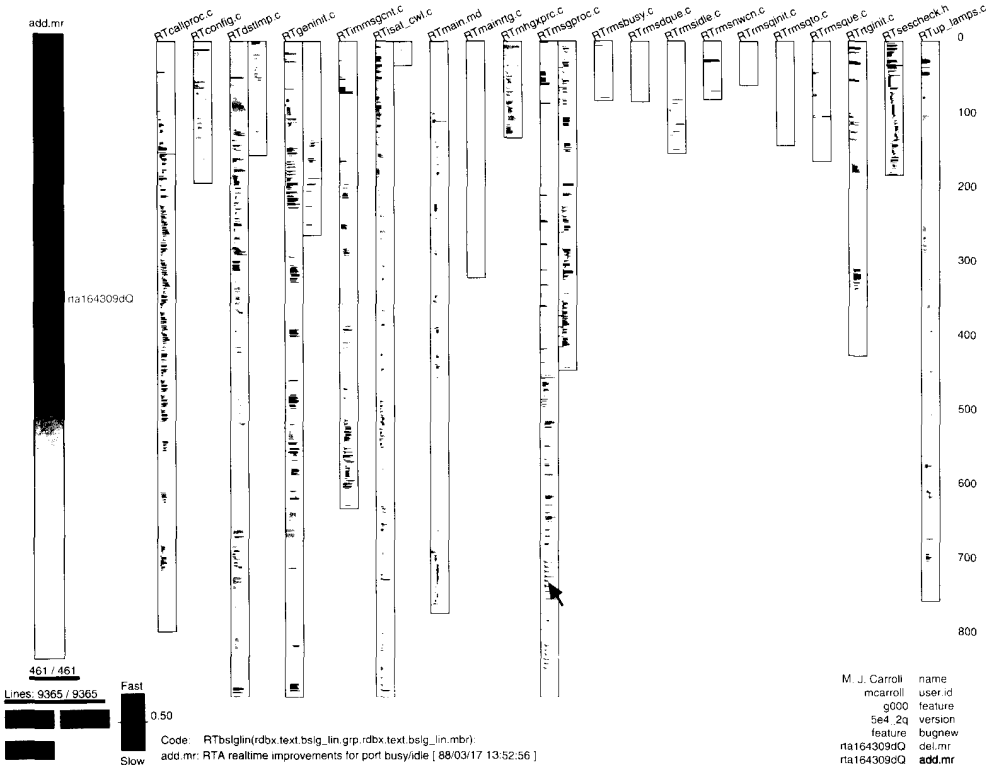

Fig. 1. Sample Seesoft display. A Seesoft display of a directory with 20 files and 9 365 lines of code. Each file is represented as a column and each line of code as a colored row. The files are either C code (.c), header (.h), or configuration management (.md) files. The color of each line is determined by the modification request (MR) that created the line. All MR's touching any of these files are shown on the left using a color scale with the oldest in blue and the newest in red.

in blue. On the left there is a scale showing the color for each of the 461 changes or modification requests (MR's) to this directory. The visual impression is that of a miniature picture of all of the source code with the indentation showing the usual C control structure and the color showing the age.

In the remainder of this paper we describe Seesoft and our visualization ideas in more detail. Section II describes the code display techniques and computer interaction methods used in Seesoft. Section III walks through a sample code analysis session to illustrate the types of insights that we have obtained while using Seesoft. Section IV discusses the general principles we use in our visualization techniques. These visualization techniques may be applied to the display of any ordered database. Section V discusses some software engineering applications. Section VI talks about our field experiences using Seesoft to solve some actual problems. Section VII describes how we implemented Seesoft, and Section VIII summarizes and concludes.

II. The Seesoft Software Visualization Tool

The Seesoft visualization tool displays line oriented source code statistics by reducing each file and line into a compact representation. The statistics are displayed with color. Then, using high-interaction graphics and direct manipulation techniques [16], the user manipulates the display to discover interesting patterns in the code and statistics.

For this approach to be effective the initial display must be informative and clear. With our display, programmers immediately recognize the files and lines of code because the display looks like a text listing viewed from a distance. The statistics are obvious from the row colors as is the spatial distribution of the statistic in the code. Next there must be easy and intuitive human interface techniques for the user to manipulate the display. We find that using direct manipulation techniques, in particular updating the screen in real-time in response to mouse actions, allows the user to manipulate Seesoft easily to find interesting patterns. Finally, there must be a technique to allow users to read the code. In our system users may open *code reading* windows that display the actual code corresponding to the rows underneath "magnifying" boxes that track mouse movement. This technique works well because it allows the user to have both an overview of the statistic and also read the interesting parts of the code. We now describe the Seesoft visualization tool in more detail.

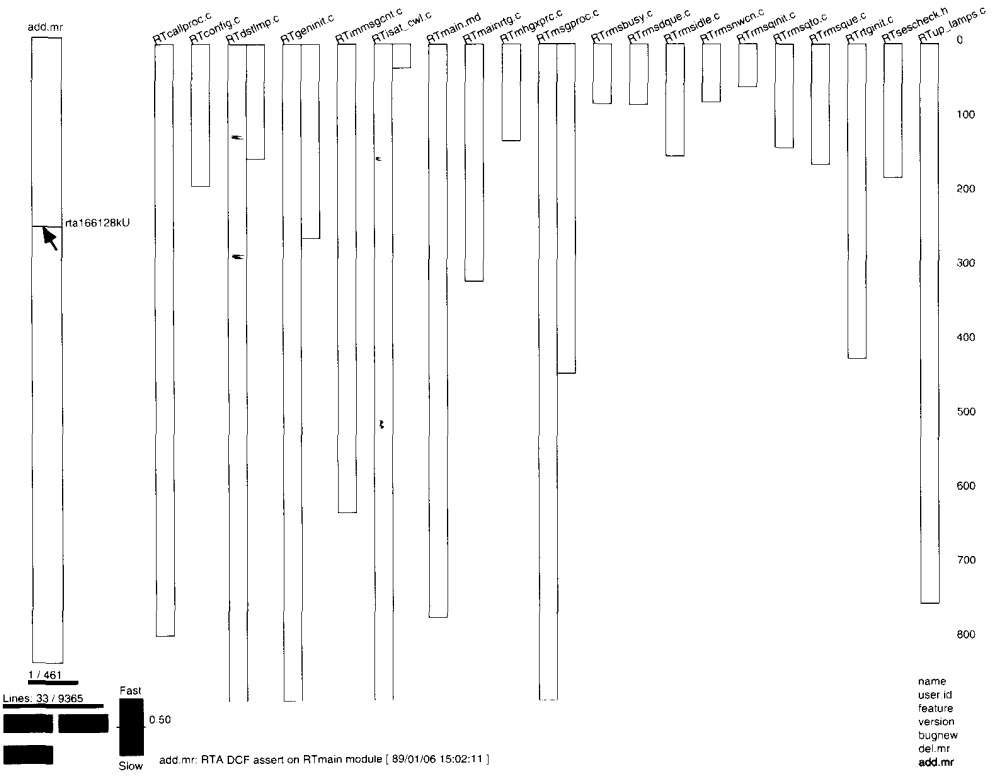

Fig. 2. Example MR activation. As the user positions the mouse over an MR, the lines of code that MR created are activated.

2.1. Screen Description

The Seesoft screen layout consists of a file display, a mouse sensitive color scale, buttons, toggles, and a list of statistic names. Fig. 1 shows that the largest portion of the screen display consists of files shown as columns containing lines of code shown as colored rows. Using a 1280 × 1024 standard high-resolution monitor we can display about 900 lines of code per column. Files longer than 900 lines wrap and are displayed as multiple columns. For example, file `RTmsgproc.c` in the middle of Fig. 1 has 1 300 lines and is displayed in two columns. The name of each file is printed above it for easy identification. The row representation shows clearly the indentation and length of each line of code. The color of each line is tied to a line oriented statistic. This statistic is highlighted on the list of statistic names in the lower right-hand corner. The rows are just large enough so that block comments, functions, and control structures such as *case* and *if* statements are visible just by their indentation.

On the left side of the Seesoft display is a mouse sensitive color scale. Each color on the scale represents one value of the statistic associated with each line of code. The statistic might be age, programmer, feature, type of line, number of times the line was executed in a recent test, and so on. We often use the MR (modification request to a version control system) number for our statistic. The MR number is an interesting statistic because it comes in date order, is the smallest unit of program change, and is associated with programmers, developers, and features. The MR's are displayed sequentially with the newest at the top and oldest at the bottom. Underneath the scale, the number of activated statistic values and the total are shown, 461/461 in Fig. 1, as well as the number of activated code lines and total, 9365/9365. The color scale is a generalization of traditional sliders controlling thresholds. The user may select discontinuous threshold ranges, as well as the traditional continuous ranges that normal sliders may select.

At the bottom of the screen there is space for Seesoft to print the current code line and the statistic value. As the cursor is positioned over any color in the color scale the value of the statistic represented by the color is printed at the bottom of the Seesoft display. In Fig. 1 this is the MR number, abstract (a short description of the purpose of the MR), and date. If the cursor is over a row, Seesoft also prints the line of code associated with that row. We have additional methods of viewing that we describe below.

2.2. Linking Between the Color Bar and Code Lines

Each statistic value is linked to the lines of code having that value through a common color. When the user activates

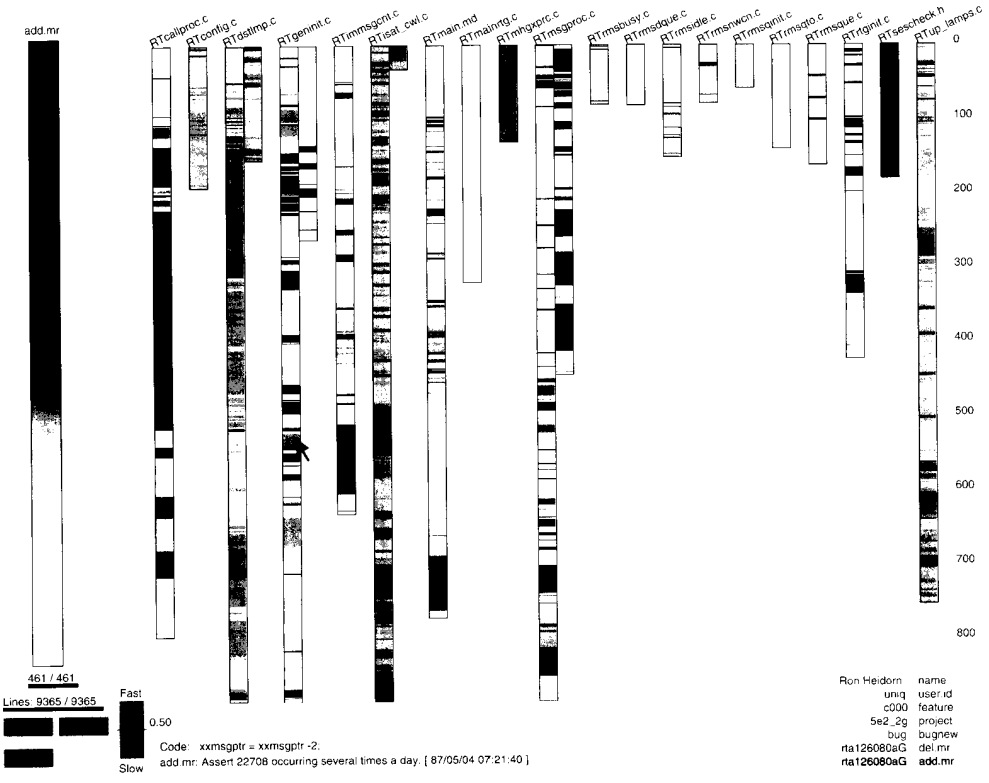

Fig. 3. Source code with no indentation. By turning off indentation it is easier to see the age of the code. The oldest lines are displayed in dark blue and the newest in red. The display shows the relative size of the files, age of the code, and how many times each file has been changed.

a value by positioning the cursor over it, the associated color is activated and the corresponding code lines are turned on. In Fig. 1 all MR's are activated and thus all code lines were visible. To obtain Fig. 2 from Fig. 1, first all MR's are deactivated and then the cursor is positioned over one MR in the scale. Activating a line of code activates its MR and thus any other lines of code created by that MR. Activating a file activates all of the lines in the file and thus all of the MR's used to create them.

2.3. Mouse Operations

The screen is mouse sensitive. As the user moves the mouse around the screen the entity under the mouse is automatically activated and the corresponding color and lines turned on. Activating MR's and code lines is described above. Activating a file name activates all lines within that file. When the mouse is moved away from an entity it is automatically deactivated. Depressing the left mouse button causes the activation to be permanent and the middle deactivates previously activated items. This style of user interaction is called brushing [17].

2.4. Buttons, Toggles, and Code Reading

At the bottom of the display there are mouse sensitive buttons and an animate slider. The user may turn off line indenting by clicking on the *Indent* button. This causes the rows to be drawn the full width of each column (see Figs. 3 and 4). The *Animate* button causes the computer to sequentially display all statistic values, one at a time.

Depressing the *Reading* window button causes two actions to occur. A window for displaying C code text is opened, and a small colored "magnifying" box is created. As the magnifying box is moved over the colored rows the actual code is displayed in the reading window (see Fig. 5). The size of the magnifying box is proportional to the amount of code that is displayed in the reading window. This enables the user to understand what fraction of the total code is visible in the code reading window. The border color on each reading window matches the color of its corresponding magnifying box.

III. Sample Code Analysis

This section describes a quick analysis of the change history in a sample directory using Seesoft. From the version control

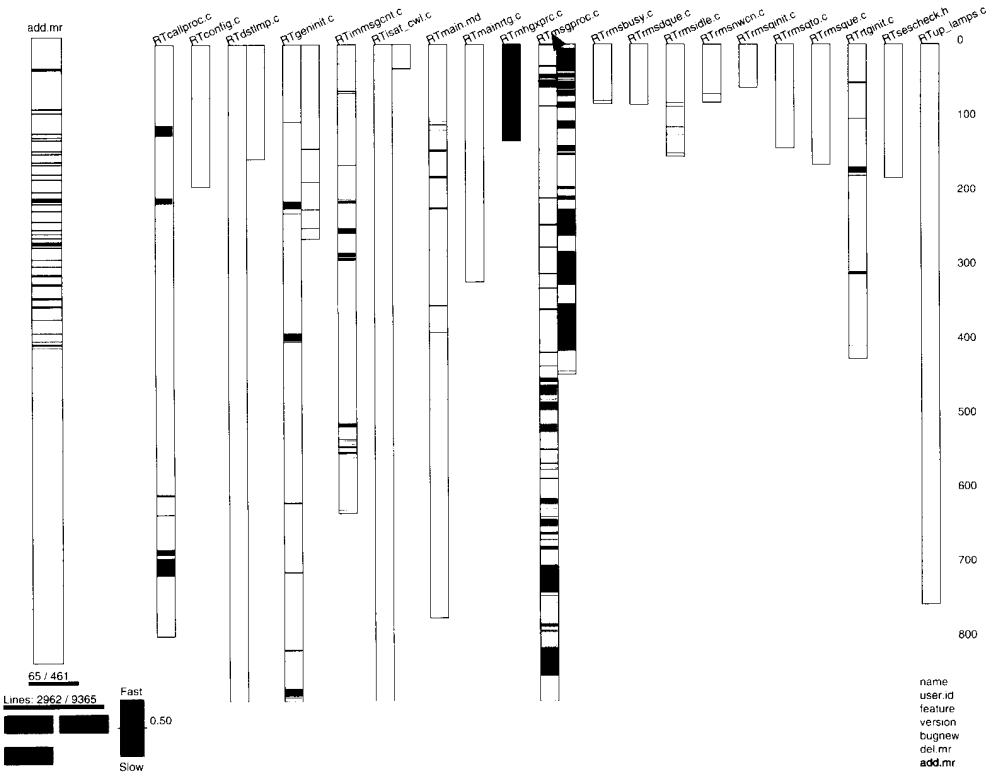

Fig. 4. A File With Many Changes. File `RTmsgproc.c` has been changed 65 times out of 461 total for this directory. The changes have occurred consistently over the last several years.

system for each line we obtain the adding MR, whether the change fixed a bug or added new functionality, the version, the feature number, the user id of the developer adding each line, and the developer's name. The directory has 20 source code files including a configuration management (.md) and an include (.h) file that were created by 461 MR's and 168 developers. In total there are 9 365 lines of code. Fig. 3 shows the code with indentation turned off, and the color of each line tied to the MR that created it. Turning off indentation makes it easier to see the age of the lines. The oldest code dates to 1984 and is shown in dark blue and the newest code from early in 1991 is shown in red. The age patterns of the files are striking. Much of the code in files `RTmsgpoc.c`, `RTcallproc.c`, `RTgeninit.c`, `RTmainrtg.c`, and `RTginit.c` is blue indicating that it dates to 1984. These files are interesting because along with the blue code they display many other colors indicating that they have been changed many times, including recently. There is a set of small light blue files (`RTrmsbusy.c`, `RTrmsdque.c`, `RTrmsidle.c`, `RTrmsnwcn.c`, `RTrmsqinit.c`, `RTrmsqto.c`, and `RTrmsque.c`) dating to 1985, and a set of green files (`RTconfig.c`, `RTdstlmp.c`, `RTisat_cwl.c`, `RTmhgxprc.c`, and `RTup_lamps.c`) dating to 1987. The light blue and green files have been stable since few changes are shown.

In this directory there have been 461 changes. To find the files that have changed the most we turn off all lines and sequentially touch each of the file names. Fig. 4 shows that the file `RTmsgproc.c` has been changed 65 times, and the pattern of the changes on the MR color scale shows that changes have been occurring consistently since this file was created. We investigated the frequent changes and found that it is the main control flow for the directory. A common enhancement strategy has been to add a function call to this file and put the code for the new function definition in another file. Fig. 5 shows such an enhancement where a new function was added to fix a bug and a corresponding function call was inserted in `RTmsgproc.c`.

By activating files created by common MR's, there appear to be three different sets of files in this directory. Figs. 6, 7, and 8 show the groupings. Fig. 6 shows the set of green files that were added in 1987. These files were created for a major enhancement.[2] Fig. 7 shows the set of light blue files that have

[2] A 5ESS expert on this section of the code subsequently told us that these files were added to provide IDSN capability.

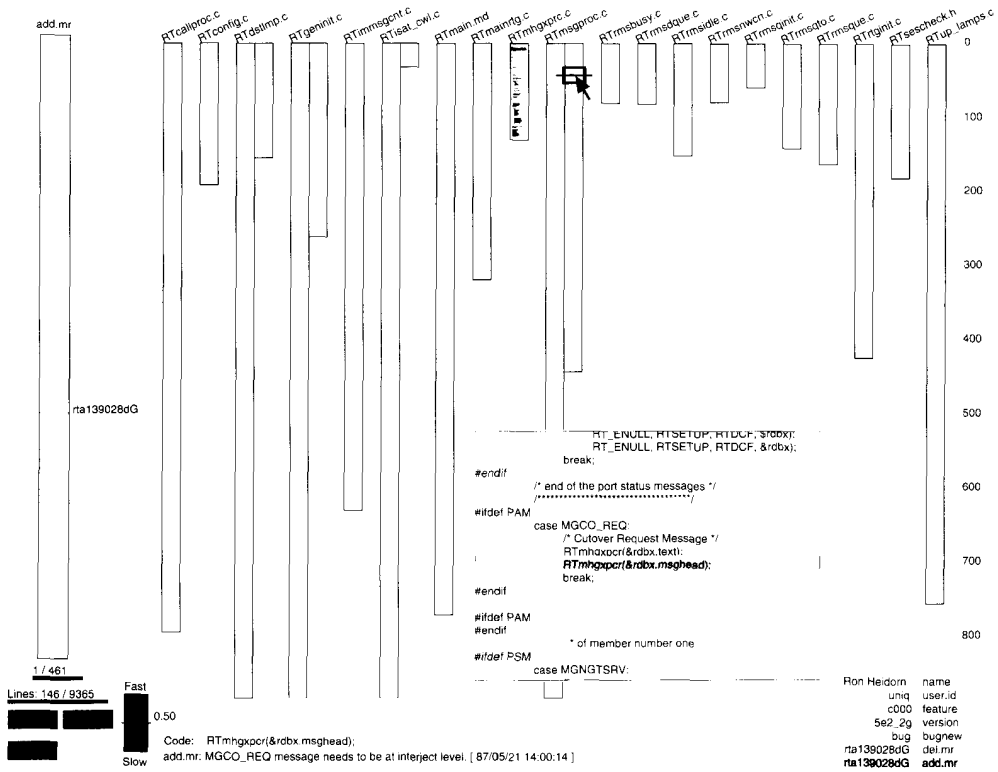

Fig. 5. Modular change. A bug is fixed by inserting a function call in `RTmsgproc.c`, shown in the *Code Reading* window, and putting the code for the function in a new file. The *Code Reading* window may be independently positioned.

been stable since 1985.[3] The heavily changed set of files in Fig. 8 are the control flow files in the program. In this directory there is a clear historical work pattern. New functionality is added by creating new functions and then inserting function calls in the main files. Through time there have been several major enhancements that created sets of files.

There are two types of MR's, new feature MR's and bug fixing MR's. Fig. 9 shows the display with the bug MR's in red. Certain files such as the green and light blue files have had few bug fixes. These files were created at one time with a small number of MR's. Other files have multiple bug fixes.

In Fig. 10 the line color is tied to the user id of the programmer writing each line. Files `RTconfig.c`, `RTdstlmp.c`, and `RTup_lamps.c` are written by one individual with a few changes by other people later to fix bugs. Some of the other files with lots of colors have been changed by many different developers. There is a relation between the locations of the bug fixes in Fig. 9 and the number of developers touching the files in Fig. 10. Files touched by many developers have more bug fixes.

[3]The expert was unaware that these files even existed.

What have we learned in this quick Seesoft session? We know which files are changed most often, the age of the code, and when each file was last changed. We also know that the files in this directory may be clustered into three groups, each group created by a different set of MR's. If it became necessary to divide this directory, files in each cluster could be kept together. We also know where code has been changed recently and that recent MR's have created fewer lines of code than earlier MR's. We also found that certain files have been changed continuously and that the bug fixing MR's are concentrated in these files. These files might be candidates to be restructured or rewritten to reduce maintenance costs.

IV. Visualization Techniques

Our approach to visualizing software is to think of source code files and lines as entities in an ordered database. In the preceding examples we display statistics associated with entities obtained from the version control system. For each entity we have a representation, columns, and rows, chosen so that we can view a large volume of data on a single screen. This allows us to gain insight into the overall structure of the database. Database queries are entered and answered visually.

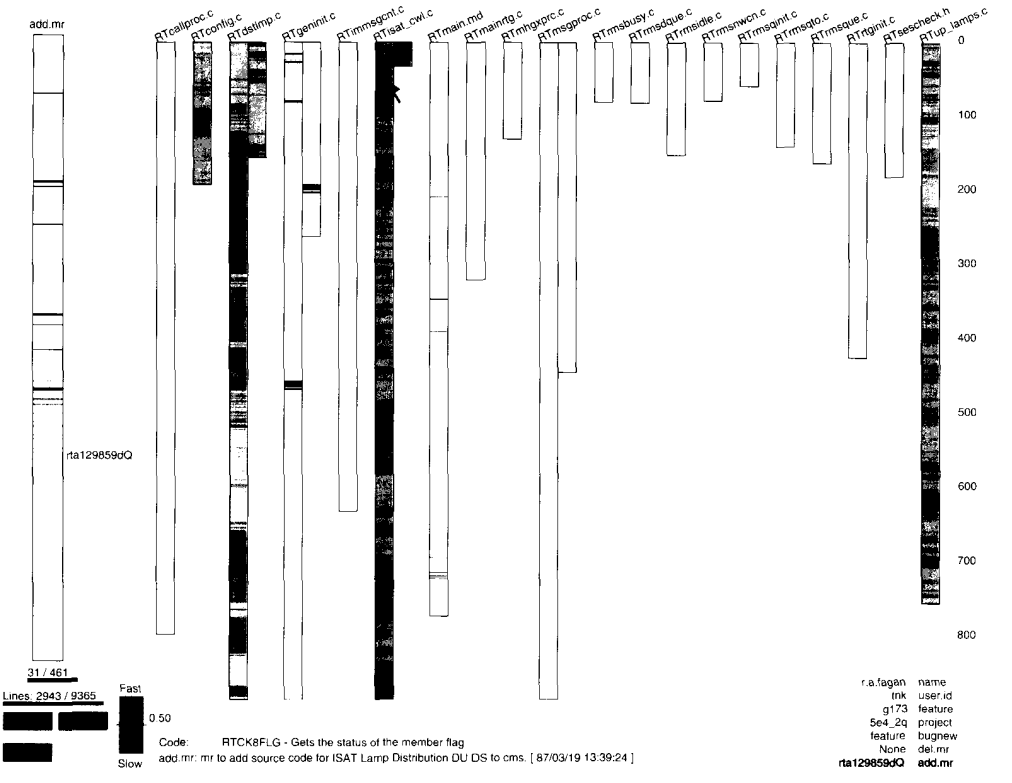

Fig. 6. Groupings of files—a significant new feature. A new feature was put into the code in 1987. This code is primarily in four files and has been stable.

As the user moves the cursor over mouse sensitive portions of the screen he or she is performing a series of database queries. Seesoft then activates the lines of code that resolve the database queries. This approach allows the user to probe the database by moving the mouse around the screen. When he or she discovers an interesting pattern, we provide a mechanism to view the database directly, the reading window in our case, in addition to the reduced representation.

We are currently in the process of obtaining some additional ordered databases. Our approach applies to databases where there is interest in understanding the overall structure and querying the database based on particular attributes. For example, one possible application would be to display a text corpus such as the Bible. Each book could be represented as a column and each verse as a row. A subject index or the age of each verse could be used to color the rows. Another application we are working on is to represent directories as columns and files as rows. This would allow us to visualize even more code on a single display.

The Seesoft user interface employs high interaction graphics and direct manipulation techniques. As the mouse is moved over the display screen entities are automatically activated and deactivated. Since there are no "point and click" delays and no waiting for screen refreshes, a different style of interaction is possible. Our style of interaction makes it easy for the user to experiment with different activations and to probe the display interactively. For example, with Seesoft it is possible to view each one of several hundred MR's by running the mouse over the color scale. Any unusual MR's will be visually obvious. This would be infeasible if the user were required to "click" on every MR.

V. Seesoft Applications

We envision Seesoft being used in several application areas including

- code discovery,
- new developer training,
- project management,
- quality assurance and system testing,
- software analysis and archeological studies,
- code coverage analysis, and
- code execution optimization.

The code discovery problem is faced by a programmer attempting to change an unfamiliar portion of the source code. Programmers, given requests for additional functionality, must

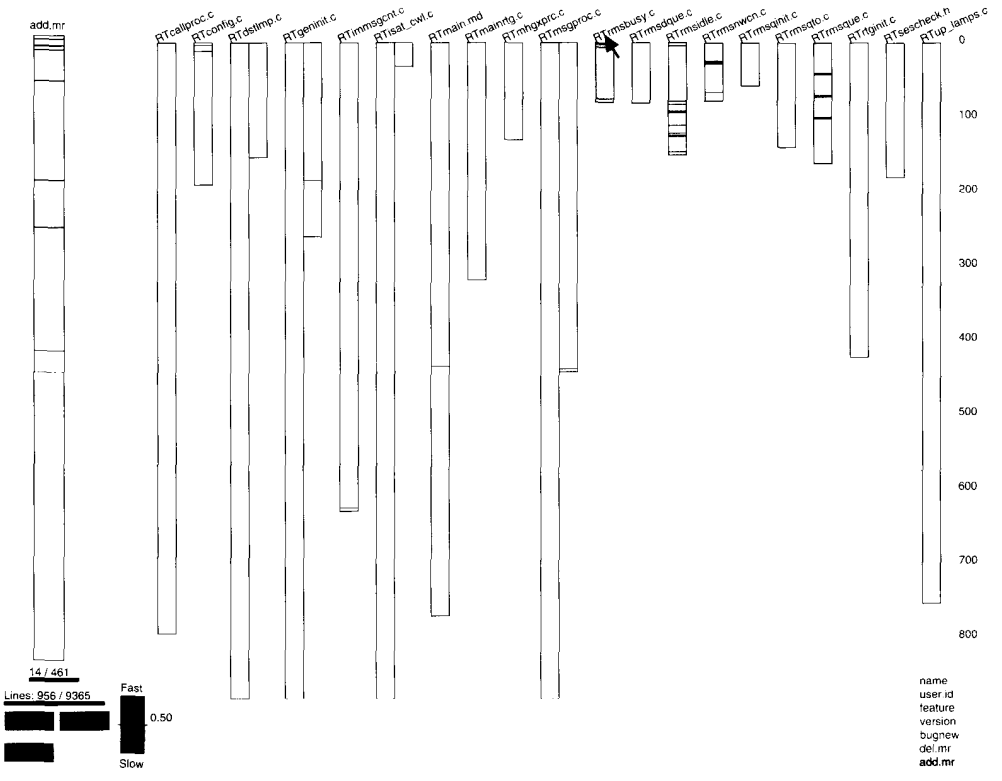

Fig. 7. Groupings of files—stable utility functions. By activating files created by common MR's, we find that there are three different sets of files in this directory. This set is a stable set of utility functions.

study the current code to determine which files contain the existing functionality and which lines to change within these files. This task is often difficult and time consuming. In fact it may take several weeks of detailed study to change a few lines with no unwanted side effects. On large, old projects a significant fraction of a programmer's time is devoted to code discovery. Using Seesoft a programmer can easily determine which lines were created to deliver an existing functionality, the programmers who created those lines, why they were created, and the purpose of nearby lines.

New programmer training is a problem faced by all large software projects. Multiyear projects with large development staffs have considerable staff turnover. Seesoft can ease the programmer training problem by providing new trainees with a global view of the source code. Since Seesoft displays tens of thousands of lines of code simultaneously, new programmers can form a mental picture of the code. Using Seesoft it is easy to answer questions such as:

- How are the files in my program organized?
- When were they last touched?
- Where is the code for this feature?
- What code was written by the person I am replacing?

A class of new programmers might be given Seesoft and access to a code expert. They would use Seesoft to view the code and could immediately ask the expert to explain the interesting things that they discovered.

Project managers monitor a development in order to ensure that the project is on schedule. Using Seesoft a project manager can visually track all source code changes done during the last week or month and can verify that recent changes are consistent with the schedule. In addition, he or she can identify potential trouble spots by the presence of a high level of churn or excessively complex code. A manager can check recent changes in order to trap quick fixes that are likely to cause long term difficulties. He or she can also use Seesoft to identify code that needs to be restructured or rewritten.

Quality assurance inspectors can use Seesoft to determine if new code meets coding specifications and is in the proper files. System testers can determine which regression tests to run by identifying the system functionality embodied in the files that are changed by recent MR's.

Analysts may use Seesoft to understand the effect of various software development environments and processes. For example, an analyst could compare code developed using C

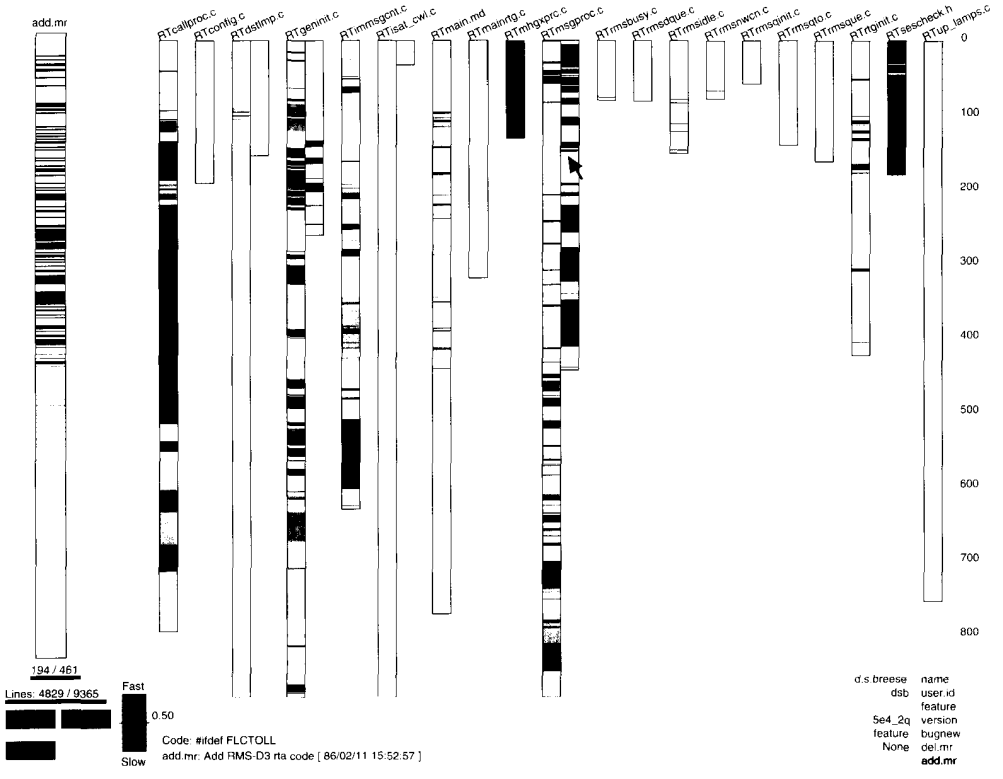

Fig. 8. Groupings of files—main process control flow. Activating files `RTmsgpoc.c`, `RTcallproc.c`, and `RTgeninit.c` creates this figure. These files control the flow in this directory. They have been changed many times, usually with small changes, as developers add hooks for new functionality. Many of the MR's are for bug fixes.

and C++ in order to determine if bug fixes in the C++ code were more localized. Analysis of multiple projects over time could be used to estimate the effects of project age and size on programmer efficiency.

Developers optimizing the performance of a program can use Seesoft to display execution frequency data for each line, identify hot spots, and then use data from static analyzers and version control systems to evaluate potential improvements.

VI. Seesoft Field Experiences

This section describes the experiences of some actual Seesoft users. The reaction of programmers and managers at all levels to Seesoft demonstrations has been enthusiastic. Many state that they wish that Seesoft had been available for some recent work. Another common scenario is that a department picks up responsibility for some unfamiliar code and would like to use Seesoft to help gain familiarity with the software. Since our visualization techniques are recent, our experiences to date with actual applications are limited, but preliminary results are promising.

The results in Figs. 9 and 10 suggest that files changed by many different developers have more bugs than files written by one or two developers. These and other results have led some projects to assign ownership of large sections of code to individuals who will have responsibility for all changes. To make the code assignments one developer used Seesoft to look at the change history of all of the code in her project. She divided the code into related areas using a clustering technique based upon the change history, and used Seesoft to review the results. She then used Seesoft to assess the activity level in each cluster, and assigned developers sets of clusters intended to balance the load.

Another Seesoft project involved object-oriented programming. A particular manager was interested in applying object-oriented programming to developing switching software. She used Seesoft as an archeological tool to determine which subsystems would benefit most. She found that in certain subsystems the embedded code rarely changed—new features involved adding new files. For these subsystems she concluded that the object-oriented approach would have limited value. For other subsystems, however, implementing new features involved making extensive changes to existing files, suggesting that using an object-oriented approach might be useful.

There have been several cases where developers were

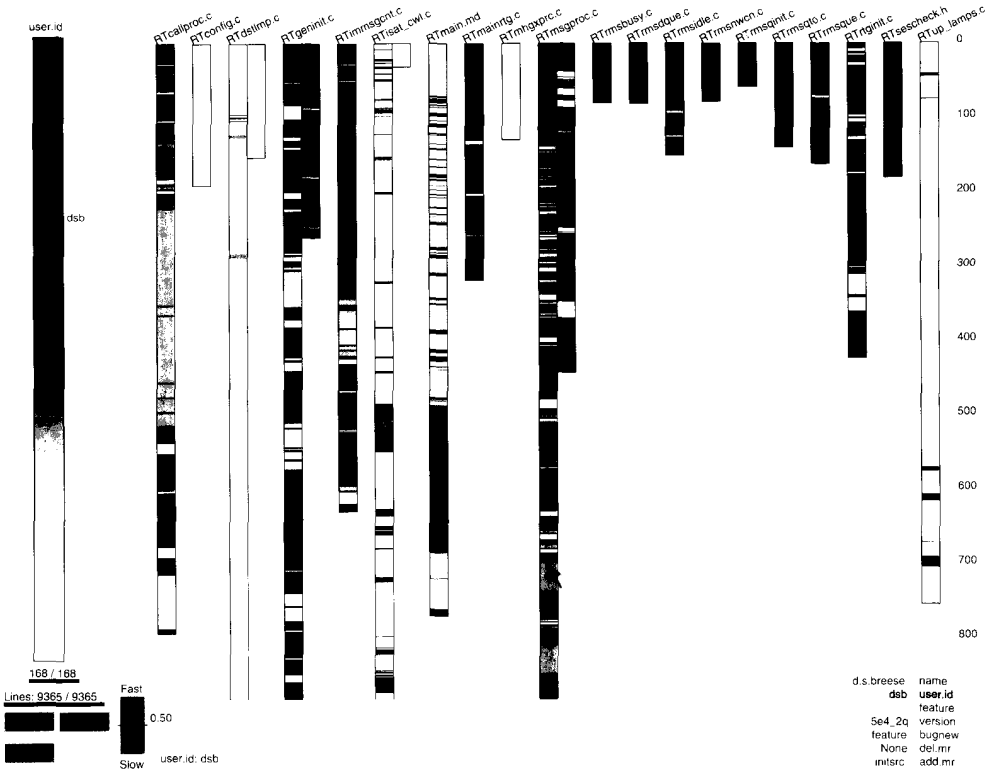

Fig. 9. Locations of bug fixes. MR's for fixing bugs are shown in red. Bug fixes are concentrated in a few of the files.

interested in looking at their own code. One particular expert looked at a recent port and discovered that he had made an error. He had copied a few files from another application and had intended to add a few lines of his own code to each file in order to complete the port. When he colored the copied code in blue, and his own in red, it was immediately obvious that he had failed to add code to one of the files.

VII. Implementation

Seesoft currently runs on Silicon Graphics Iris workstations, although we plan to port it to the X Window System [18]. Seesoft is written in C++ [19] and uses the Silicon Graphics GL graphics library. In total it is about 2 000 lines of C++ code. To deliver real-time user interaction we require the graphics capability to rapidly manipulate the displays, particularly the color map. Seesoft draws each statistic value and associated code lines in its own color. Activating and deactivating is done by manipulating the color map. The colors for the activated statistic values are turned on and for the deactivated values are turned off. Color map manipulation is fast on Iris workstations because it is done in hardware.

Our source code history data came from ECMS. We use the S language for data management and preliminary analysis [20]. To read this data we developed a series of shell scripts to strip unnecessary information and reformat the data for S input. S provides a computational environment, static graphics, and data management that support interactive manipulations. We link Seesoft into the S executive, perform all data manipulation in S, and then launch Seesoft from S.

Silicon Graphics Iris workstations come with 19-in color monitors. Using the column and row representation we find that we can easily understand 20 000 lines of code and can understand 50 000 if we are close to the monitor. In each column we can display about 900 lines and can comfortably fit 25 columns on a single monitor. With more than 50 000 lines displayed the columns become very narrow.

VIII. Discussion and Conclusion

This paper describes a new technique for visualizing line oriented statistics associated with source code and a software tool, Seesoft, embodying the technique. There are four key ideas: reduced representation, coloring by statistic, direct manipulation, and capability to read actual code. The reduced representation is achieved by displaying files as columns and lines of code as thin rows within the columns. The color of each row is determined by a statistic associated

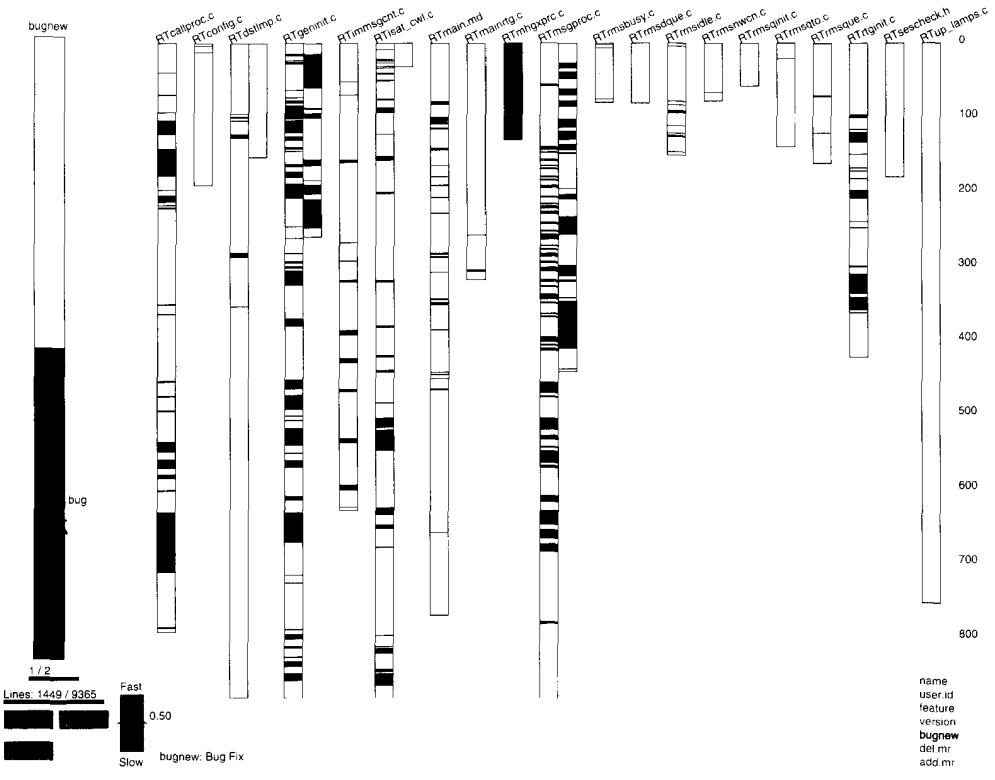

Fig. 10. The color shows the users making changes in this directory. Note that the files with lots of bug fixes are the same files that have been changed by many different developers.

with the line of code that it represents. In our examples we obtained the statistics from a version control system. The visual impression is that of a miniaturized copy of the code with color depicting the spatial distribution of a statistic. Then, using direct manipulation and high interaction graphics, a user manipulates the display to find interesting patterns. To display the actual code text the user opens up reading windows and positions magnifying boxes over the reduced representation.

Besides analyzing source code statistics, our technique has application to any ordered database. Examples include transaction databases, indexed text such as legal writings, a text corpus such as the Bible, and software documentation.

As with any method there are limitations. Our visualization technique provides a qualitative view of the distribution of a statistic in code. As with all graphical methods, the technique is useful for discovering patterns. After the patterns are discovered, hypotheses may be tested by means of standard statistical methods. Currently, Seesoft is unable to display more than 50 000 lines of code simultaneously; however, we are working on other techniques using different abstractions that scale beyond this limit. A key idea in Seesoft is its interactive use of direct manipulation techniques and use of color. It is difficult to describe these in a static monochrome medium such as this paper.

We developed Seesoft in conjunction with the 5ESS ®Telecommunications Switch Project, which includes millions of lines of code developed over 10 years. Because the initial results are so promising we are creating an 85-gigabyte optical disk based archive of the complete code history of 5ESS, including version control data, project management data, and cscope symbol databases derived from monthly snapshots of the code.

ACKNOWLEDGMENT

The authors would like to acknowledge helpful conversations with R. A. Becker, R. Drechsler, G. Nelson, and A. R. Wilks.

REFERENCES

[1] W. F. Tichy, "RCS—A system for version control," *Software—Practice and Experience*, vol. 15, pp. 637–654, 1985.
[2] M. J. Rochkind, "The source code control system," *IEEE Trans. Software Engineering*, vol. SE-1, pp. 364–370, 1975.
[3] B. R. Rowland and R. J. Welsch, "Software development system," *Bell Syst. Tech. J.*, vol. 62, part 2, pp. 275–289, 1983.
[4] P. A. Tuscany, "Software development environment for large switching projects," in *Proc. Int. Switching Symp.*, pp. 199–214, 1987.

[5] S. Cichinski and G. S. Fowler, "Product administration through SABLE and NMAKE," *AT&T Tech. J.*, vol. 67, pp. 59–70, 1988.
[6] Y. F. Chen, "The C program database and Its applications," in *Proc. Summer USENIX Conf.*, 1989.
[7] J. L. Steffen, "Interactive examination of a C program with Cscope," in *USENIX Dallas 1985 Winter Conf. Proc.*, USENIX Association, pp. 170–175, 1985.
[8] P. J. Weinberger, "Cheap dynamic instruction counting," *AT&T Bell Laboratories Tech. J.*, vol. 63, pp. 1815–26, 1984.
[9] R. Baecker and A. Marcus, *Human Factors and Typography for More Readable Programs.* Reading, MA: Addison-Wesley, 1990.
[10] A. A. Pal and M. B. Thompson, "An advanced interface to a switching software version management system," in *Proc. 7th Int. Conf. Software Engineering for Telecommunications Switching Systems*, pp. 110–113, 1989.
[11] G. M. Nielson, B. Shriver, and L. J. Rosenblum, Eds., *Visualization in Scientific Computing.* Los Alamitos, CA: IEEE Computer Society Press, 1990.
[12] E. R. Tufte, *The Visual display of Quantitative Information.* Cheshire, CT: Graphics Press, 1983.
[13] A. W. Donoho, D. L. Donoho, and M. Gasko, *MACSPIN: A Tool for Dynamic Display of Multivariate Data.* Monterey, CA: Wadsworth & Brooks/Cole, 1986.
[14] R. A. Becker and W. S. Cleveland, "Brushing scatter plots," *Technometrics*, vol. 29, pp. 127–142, 1987.
[15] R. A. Becker, S. G. Eick, and A. R. Wilks, "Basics of network visualization," *IEEE Computer Graphics and Applications*, vol. 11, pp. 12–14, 1991.
[16] B. Shneiderman, "Direct manipulation: A step beyond programming languages," *IEEE Computer*, vol. 16, pp. 57–68, 1983.
[17] R. A. Becker, W. S. Cleveland, and G. Weil, "The use of brushing and rotation for data analysis," pp. 247–275 in *Dynamic Graphics for Statistics*, William S. Cleveland and McGill, Eds. Wadsworth, 1988.
[18] V. Quercia and T. O'Reilly, "X window system user's guide," O'Reilly & Associates, Inc., Sebastopol, CA, 1988.
[19] B. Stroustrup, *The C++ Programming Language.* Reading MA: Addison-Wesley, 1987.
[20] R. A. Becker, J. M. Chambers, and A. R. Wilks, The New S Language. Pacific Grove, CA: Wadsworth & Brooks/Cole, 1988.

Stephen G. Eick (M'87) received the B.A. degree from Kalamazoo College in 1980, the M.A. degree in mathematics from the University of Wisconsin, Madison, in 1981, and the Ph.D. degree in statistics from the University of Minnesota in 1985.

He joined AT&T Bell Laboratories in 1985, where he has been involved in statistical graphics and network research. He has developed techniques to visulaize networks, network performance models, and network simulations. As a member of the Software Production Research Department, he is applying visulaization techniques to understand software.

Dr. Eick is a member of the ACM.

Joe Steffen received the B.S. degree in electrical engineering from Purdue University and the M.S. degree in computer science from the Illinois Institute of Technology.

He has done considerable work on software tools, including writing cscope and ctrace, which are part of UNIX System V, and adding run-time checking of array subscripts and pointer bounds to the portable C compiler. His current research interests are configuration mangement for large software systems and software tools.

Mr. Steffen is a member of the ACM.

Eric E. Sumner, Jr. received the A.B. and Ph.D. degrees in engineering science from Harvard University in 1980 and 1984, respectively.

He joined AT&T Bell Laboratories in 1984 and is currently Head of the Software Production Research Department, which was formed in 1990 at Indian Hill, the Illinois complex that houses many large software developments, including the 5ESS® switch. Prior to assuming his current position, he played backgammon professionally, developed models of composite materials, built choice models for AT&T network equipment, and led the development of a tool for computer-aided engineering of underwater surveillance systems.