Novel Methods of Displaying Source History: A Preliminary User Study

Zachary Weinberg

May 16, 2002

Abstract

This paper discusses the limitations of existing tools for displaying the history of source code. We then propose an improved tool, the "flip viewer," to deal with one of the limitations. Results from user testing are presented and directions for future exploration discussed.

```
1.103  (neil    03-Jan-02): /* Handle shifting A left by B
1.103  (neil    03-Jan-02):    bits. UNSIGNEDP is non-zero
1.103  (neil    03-Jan-02):    if A is unsigned.  */
1.21   (ghazi   25-Feb-99): static HOST_WIDEST_INT
1.1    (law     11-Aug-97): left_shift (pfile, a, unsignedp, b)
1.1    (law     11-Aug-97):      cpp_reader *pfile;
1.21   (ghazi   25-Feb-99):      HOST_WIDEST_INT a;
1.37   (zack    07-Mar-00):      unsigned int unsignedp;
1.21   (ghazi   25-Feb-99):      unsigned HOST_WIDEST_INT b;
```

Figure 1: Sample `cvs annotate` output

Introduction

Large software applications typically have their source code stored in a "version control system" such as CVS [1], which maintains a record of all the changes which have ever been made to the application. This facilitates maintenance, by allowing the developers to examine the complete history of the application. The capabilities of version control systems vary widely, but all can extract any version of a file, calculate the differences between any two versions, and resolve conflicting changes made by two people simultaneously (this last typically with user assistance). Most also have some capability for "branching," where two or more editions of a file evolve in parallel. This allows, for instance, bug fixes to be applied to a stable version, without dragging in any other, riskier changes.

Developers commonly need to know the answers to questions such as "When was this chunk of code introduced?" or "What were all of the changes made to fix this particular bug?" The version control system can answer these questions, but existing interfaces to display the answers are often quite limited. This paper will explore one way to improve the interface, and suggest several others.

Existing practice

Existing tools for examining source history can be divided into three broad classes: *history browsers* that look at the time evolution of a single file, *cross-referencers* that make connections between files at a single point in time (tracing call graphs, locating all the uses of particular identifiers, etc), and *merge tools* that assist in combining two branches of a file. Examples of each are the `annotate` operation built into CVS, the `cscope` program [2], and the "Ediff" mode of GNU Emacs [3], respectively.

Cross-referencers and merge tools tend to be much more advanced than history browsers. We believe this is because cross referencing and merging are nearly impossible without sophisticated mechanical assistance. An inadequate cross-referencer leaves one repeatedly issuing `grep` commands to hunt for text, which is tedious and error-prone. Merging is worse; a poor merge tool will increase its user's error rate compared to doing the task by hand. However, it's possible to deal with an inadequate history browser with work process—for instance, a requirement to mention all the files changed by a particular patch in the check-in comments can compensate for the version control system not recording that list. Therefore, attention has been focussed on improving cross-reference and merge tools, not history browsers.

Accordingly, in this paper we will discuss possible improvements to history browsers. Figure 1 shows typical output from the `cvs annotate` command. I would like to draw your attention to three problems. First, this listing tells you that a change happened on the third line in revision 1.21, but it does not tell you what the change was. To find out, you must switch to a different display (`cvs diff`). Second, any deleted lines are completely invisible. Third, the only way to move through time is to terminate the current listing and request a new one, which makes it very easy to lose your place.

One could also reasonably object to the amount of visual space occupied by the annotations for each line. These are wider than they need to be, often visually distracting, and include information that you may not need. There is no way

to adjust what appears there.

The flip viewer

The flip viewer is a novel graphical history browser which aims to solve all the above problems with `cvs annotate`. The basic concept is shown in figure 2 (next page). One version of the file is displayed in full; on either side are columns of "change markers" which show where changes have occurred. Clicking on any column causes that revision to be displayed. This layout is vaguely derived from lifestreams [4], although it does not have the sophisticated search features or the all-consuming metaphor of that system.

The age of any given line is indicated by the distance to its nearest change marker. This is reinforced by drawing each line with a gray background, which gets darker the longer ago the last change was. Since this is an interactive display, it need not show the detailed per-line annotations that CVS does. The user can select a revision to see its number, date, and author.

A key innovation which may not be obvious from the figure is that any given line of text always appears on the same horizontal row. Blank lines are inserted wherever necessary to maintain this invariant. This means you can move forward or backward in time without losing your place.

The complete application has three more panes. One, above, shows a diagram of the version graph of the file being examined. The other two, below, show the comment associated with the current version, and a list of the tags attached to it. Figure 3 (at the end of this paper) has a complete screen shot.

User testing

The prototype was offered to two large groups of experienced software developers, along with a sample file to browse, a short description of the program's interface, and a questionnaire. Users were asked to experiment with the prototype and then answer six relatively open-ended questions:

1. How comprehensible did you find the overall display?
2. How easy was it to understand the history view in the middle pane?
3. Do you think this tool or a similar one would be useful to you? Why or why not?
4. What would you like to have added to the information presented?
5. What would you like to have removed?
6. Do you have any other suggestions for improvements? Please be specific.

Seven people responded to the survey; of these, three were unable to get the prototype to work, due to compatibility issues (it was written in a rapid-prototyping language which is not terribly portable). I anticipated this and provided a screen shot (the same one as in figure 3) which one of the three commented on.

Overall, reactions were positive. The users compared the prototype favorably to existing tools that fill similar niches, such as ViewCVS. Most everyone felt it would be useful, either as is or with improvements. However, everyone had a laundry list of improvements they wanted to see.

All of the users commenting felt that there was inadequate visual distinction between the text and the columns of change markers. One person suggested using icons or a different font for the markers, the others wanted vertical lines between the two. Different background colors were also suggested.

"`C`" seemed initially to be a natural mnemonic for "line changed," but several users were confused by it. "`+`" and "`-`" were not so confusing. One user suggested "`!`" for "line changed," which is not mnemonic at all, but happens to be the symbol used for changed lines by the Unix command `diff -c`. This is not a bad suggestion; if it can't be mnemonic, it might as well be consistent with existing practice.

More interestingly, two people objected to the way the text shifts horizontally when you select a different revision. This was a deliberate feature of the original design; it was intended to evoke the idea of flipping back and forth through a series of pages, one per version (hence "flip viewer"). However, it is distracting—in one user's words, "it's very hard to see what changed about a given line, because visually, *all* the characters change." Indentation changes are especially difficult to see. Both these people suggested displaying all the change columns to the left of the text and highlighting the selected version's column.

One user pointed out that the change markers "sort of float in space next to each line," and that it's easy to lose your place going from one side of the text to the other. To some extent this is mitigated by the thin white lines separating lines of text—but this doesn't help with text on a white background, which is the text that just changed and therefore the text the user is most interested in. Also, two other users find the thin white lines visually distracting and want them to go away. (They are an artifact of the rapid-prototyping system's GUI toolkit, not something put in intentionally.) Putting all the change columns together would solve this problem too.

The prototype did not have much in the way of features for navigating the text. For instance, the Page Up and Page Down keys did not work. Several people observed that it was not always easy to find the changes made in the current

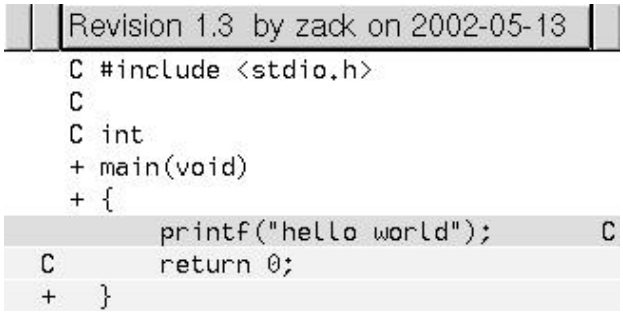

Figure 2: Flip view, basic concept

revision. They made different suggestions for how to improve this: in addition to fixing Page Up and Page Down, keystrokes to warp to the next or previous change were suggested, and more visually distinct highlighting.

The left and right arrow keys moved the currently selected revision backward or forward along the current branch, which was indicated in bold on the revision graph (see figure 3). Two users reported that they therefore expected the up and down arrow keys to navigate between branches. That was not implemented; you had to click on a revision in a different branch to activate it. This is an object lesson in the importance of implementing features completely or not at all. If two of the arrow keys do something useful and the other two don't, it's worse than if none of them do anything, unless the data being presented is truly one-dimensional.

The buttons at the top of each column of change markers are unlabelled because they do not have enough room to include the version number. It would be nice if some notation could be found to make them helpful; as it stands they simply take up screen real estate. One user suggested having the version number appear in a "tool tip" when the mouse pointer is placed over a button.

Immediate plans

Based on these results, we have some definite plans for immediate improvements to the prototype. After these are completed, we intend to carry out another round of testing. We hope to mitigate the portability problems with the first prototype—if necessary, by rewriting it in a more broadly usable language. At the same time, navigational features can easily be fleshed out, along the lines suggested by users.

Limitations of the GUI toolkit used for the prototype prevented the implementation of the interface as it was originally designed. The most significant difference was with the gray background coloring. As originally conceived, the bars would change color along their horizontal extent so that they reflected the relative age of each revision. The toolkit could not handle having more than one background color per row, so the backgrounds only showed the age of the current revision. None of the users in the test had a problem understanding the significance of the gray bars; however, we suspect that the original design would convey more information faster. We believe we know how to work around this limitation; if not, there are several other toolkits that could be used. This would also allow us to add vertical rules distinguishing the change columns from the text, which was badly wanted.

One concern with the proposal to display all the change columns to the left of the text, is that it might be hard to see what changed in the currently selected revision; your eye would have to jump back and forth across a field of irrelevant details. Carefully arranged highlighting might mitigate this. We plan to implement the proposal as it stands and try it out. If it doesn't work, another possibility is to keep the text in the middle of the display and have change columns jump from side to side, instead of having the text move within the columns.

An active project accumulates changes rapidly. On average, there have been ten changes per day made to the free compiler "GCC" since the beginning of 2002. These are distributed over roughly two thousand component files, but any given file is still likely to have a hundred or so different versions recorded in the version control system's database. It is therefore necessary for browsing tools to scale well. Unfortunately, the present prototype is based on an algorithm with cubic time complexity. This prohibits testing on files with lots of history. However, we are confident that a faster algorithm can be found.

Future directions

The current flip viewer can display changes only at one level of granularity: complete lines. When only one or two words changed in a long line, a facility for narrowing down the change is useful. One way to do this, as implemented by Ediff and other such tools, is to highlight the changed words with a different background color. This should be straightforward to add to the flip viewer.

For large files, it might be useful to have a fish-eye lens [5] which could be applied to the view. This requires the viewer to be aware of the syntax of the file it is displaying, which is nontrivial. There are too many different kinds of structured text out there for the viewer to know them all; an extensible mechanism, such as Emacs' syntax tables,[6] is necessary. On the up side, that mechanism would also allow the viewer to do "syntax highlighting," which can improve the readability of source code quite a bit.

Broader still, we would like to be able to display changes to entire directories, with file-level granularity. The simplest way to do that is give each unique file its own line in the flip view. We think this will work as well or better than clever visualization schemes like Robertson's cone

trees [7]—which may not be more efficient than the conventional "holophrastic" views.[8] People are already used to looking at directory trees displayed with one line per file, in programs such as Windows Explorer. Most version control systems do not record the list of files changed simultaneously in each checkin, which is an unfortunate limitation, but not a problem in this application; it is easy enough to correlate the revisions' timestamps.

Per-line annotations, similar to the ones `cvs annotate` generates, might be handy under some conditions. These could appear either between the text and the change markers, or out of the way on the right hand side (if separated change markers prove successful). The user should be able to control what information the annotations present.

The flip viewer can display only one version of a file at a time. It would be useful to have a mode in which it showed two versions side by side with the differences highlighted. Comparisons of two versions with a common ancestor ("three way diff") could also be handy. This operation is the core of a merge utility, which raises the question of whether it would be useful to have a combined browser and merger. We suspect that this is not as useful as it might seem. Needing to perform merges is rare. When it does happen, you generally know exactly which three versions are relevant to the merge operation. Additional information may only be a distraction.

Having a history browser around during a merge *could* help with the relatively common problem where a chunk of code has been modified on one branch and relocated on another. This leaves one wanting to apply the modification in the new position, but one may not know where that is. However, to solve this problem it suffices to be able to pop up a history browser in another window; integration is not a requirement. Also, a good merge utility has all, or nearly all, the capabilities of a full-fledged text editor; these would be less useful, perhaps even counterproductive, in a history browser.

More interesting would be to combine the flip viewer with a cross reference utility. A cross reference that was aware of the complete version history of all the files it indexes could answer questions such as "When was the last use of this procedure removed?" With current tools these can be next to impossible to answer. Terveen and Hill's "auditorium visualization"[9] is one possible way to display the cross reference information. It was invented for web site visualization, but should map nicely to call graphs and other source-code indexes.

Unfortunately, a tool like this would have to be closely integrated with the version control system itself, or else users would have to maintain local copies of the complete database. The latter is impractical, due to the size of the typical version control repository (gigabytes) and frequency of updates.

Acknowledgments

I would like to thank the people who took the time to respond to the survey: Nathaniel Smith, Dan Jacobowitz, Greg Stein, Anthony Green, Richard Guenther, Tom Tromey, and Jeffrey M. Vinconur. Also, Shweta Narayan and Michael Ellsworth provided informal comments on two different versions of the prototype.

Seth Schoen invented the algorithm used to calculate where to insert blank lines so that text always stays on the same row.

References

[1] Brian Berliner. CVS II: Parallelizing software development. In *Proceedings of the USENIX Winter 1990 Technical Conference*, pages 341–352. USENIX Association, 1990.

[2] Joseph Steffen. Interactive examination of a C program with cscope. In *Proceedings of the USENIX Winter 1985 Technical Conference*, pages 170–175. USENIX Association, 1985.

[3] Michael Kifer. *Ediff: a visual interface for comparing and merging programs*. Free Software Foundation, 1998.

[4] Eric Freeman and David Gelernter. Lifestreams: A storage model for personal data. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25(1):80–87, 1996.

[5] George Furnas. Generalized fisheye views. In *Proceedings of CHI 86*, pages 16–23. Association for Computing Machinery, April 1986.

[6] Bill Lewis, Dan laLiberte, and Richard Stallman. *GNU Emacs Lisp Reference Manual*. Free Software Foundation, 1998.

[7] George Robertson, Jack Mackinlay, and Stuart Card. Cone trees: Animated 3D visualizations of hierarchical information. In *Proceedings of CHI 91*, pages 189–194. Association for Computing Machinery, April 1998.

[8] Andy Cockburn and Bruce McKenzie. An evaluation of cone trees. In *People and Computers XV (Proceedings of the 2000 British Computer Society Converence on HCI)*. Springer-Verlag, 2000.

[9] Loren Terveen and Will Hill. Finding and visualizing inter-site clan graphs. In *Proceedings of CHI 98*, pages 448–455. Association for Computing Machinery, April 1998.

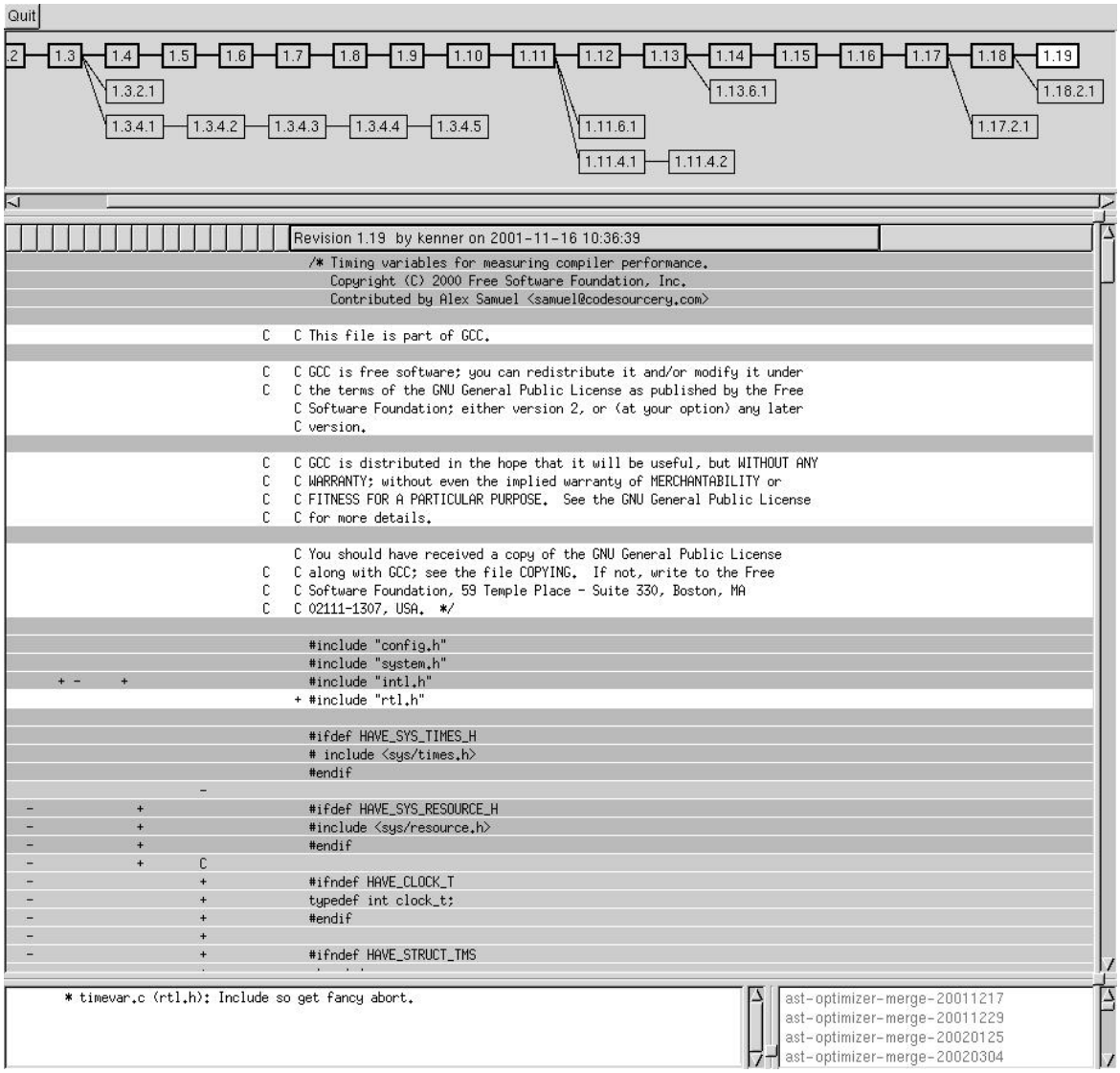

Figure 3: Complete screen shot of the flip viewer