

Summary Review #4
Sonja Ellefson
Dorrit Gordon (reviewer)
CMPS290G

Title: Managing Multi-Version Programs with an Editor

Author: Vincent Kruskal

Summary:

Vincent Kruskal wrote this article to describe ways in which one can keep track of the various versions of the same program. Discussed in this paper were the ways in which the programs are represented and the editors capable of showing these representations. One of the challenges in displaying multiple versions of a program is the person's ability to understand what is being presented. The article covers how one editor, P-EDIT, attempts to overcome this challenge.

Support for Modern Development Paradigms:

This paper seems to describe a simpler time when difficult configuration management problems were mostly related to sequential and concurrent versioning. With the advent of object-oriented programming it seems these problems have become only more complex. Previously the behavior of a functional unit depended only on what version of the code was loaded and what conditional compilation flags were relevant. Now many other factors (dimensions) may be interesting to the developer (editor user). For a particular class method the developer may be interested in not only all versions of that method (and any meta-language conditionals), but also in versions of each superclass that implements the method and possibly in versions from sibling classes. The problems noted by the author of distinguishing content changes from format changes remain just as difficult. Without a method for separating form from content it will be very difficult to even consider features like notification of commonality. While such a feature sounds very useful it's difficult to imagine a source code editor smart enough to know that two versions of the same code will have the same effect.

The paper's topics also reminded me of issues working with other forms of polymorphism, which could benefit from editor assistance. For example, an editor that could demonstrate issues with parametric polymorphism could be quite valuable. Emphasizing (or highlighting) the usage constraints of parameterized type(s) could help developers to see the limitations of their polymorphic functions (what types are legal).

Identifying Differences and Propagating Change:

In the common case where an author edits a single version of a document, Kruskal's technique sounds quite effective. It allows a versioning system to employ the more versatile Boolean expression technique for version storage, without burdening the user with additional complexity. It is less clear that Kruskal's technique is of use when applying changes to multiple versions of a document. He describes a system in which elements of the document are highlighted to indicate whether they are shared or unique. If more than two versions are being simultaneously edited, it is not clear how effective this highlighting technique would be. Is it a requirement that the multi-version program reside in a single file? Is it capable of going across

branches? The author also talks about the “difference” between two versions, but I don’t see how he measured the difference. By line? By segment? What tools does he use to calculate the difference? My guess is that it is by line and the tool is similar to diff. But I do like the notion of hiding underlying version control mechanisms and exposing only the simple and necessary user interface to end-users. SCM tools should help users instead of confusing them. The ease with which Kruskal's technique allows changes to be applied to multiple versions of a document may increase the risk of applying changes to inappropriate versions. If it requires special effort to apply changes to additional versions (as through a merge operation), a user is likely to consider more carefully which versions should receive the change.

Usability:

The idea of displaying multiple versions of the same program is useful to developers and can save time and effort because of the ease of viewing prior versions. However, very few, if not none of the common text editors used by developers are capable of displaying multiple versions of the same program. Even though this article discusses editors that were built for this purpose, developers are not using them. Why? I believe that developers are comfortable with familiar editors such as emacs, vi, or an IDE. In order for the multi-version display to be widely accepted, these ideas need to be brought into the commonly used editors. This paper contributes by documenting what an editor would need to do in order to display multiple versions. One can implement these ideas into more familiar editors.

When defining the editor, he depicts the users point of view in various paragraphs, which is helpful. "Here the user may specify that no matter which of these files he is editing, the corresponding versions of them should be available." He also tells the reader what the editor is doing for the user, behind the scenes: "(T)he editor remembers which version it was previously displaying in that environment." He runs through describing various dimensions that extend the editor's functionality like "TIME", "SYSTEM", "PRECISION", and "MASK" giving understandable descriptions of their results. It would have been good to see some screenshots and example scenarios. For example, it wasn't clear what the user sees when editing a file that is different for SYSTEM=unix and SYSTEM=win32. At one point the paper says that users always see one particular version with highlighting to indicate changes with other versions. Does this mean what the user sees (one version) and what he is editing (multiple versions) are different?

The part that seemed odd is the treatment of time. Time seems to be a special dimension - it keeps flowing one way. Also, newer things are almost always derived from older things. The system presented treats time like any other attribute. Unless the editor does something special with time, you could go back in time and edit some early code without affecting the current code at all. This seems very weird and undesirable.

Conclusion:

The article had a definite bent towards an IBM systems' point of view, which I found rather interesting to contrast compared to the previous articles. Such sentences as "since variable length records with more than one terminal blanks are rare in the VM/SP environment...", remind us he is talking about IBM files systems. Sometimes the article appears dated when talking about functions like "UNDO". He makes "UNDO" out to be not as common, whereas we now

find in most editors. Despite parts of the article which appear dated, the article overall is fairly easy to follow with his points understood.