Concurrent Versions System

Simon Yuill

2008

Originally published in Matthew Fuller (ed.), *Software Studies: A Lexicon*,
Cambridge, Massachusetts, London, England: The MIT Press, 2008.

> The highest perfection of software is found in the union of order and anarchy.
> — Pierre-Joseph Proudhon[1]

Concurrent Versions System (CVS) is a tool for managing collaborative software development. It enables groups of coders working on the same set of source files to coordinate and integrate the changes they make to the code, and acts as a repository to store all those changes. If, for example, two different programmers alter the same section of code, CVS can compare both versions and show that there is a difference between them (known as a "conflict" in CVS) that needs resolved or "merged." Another feature of the system is to keep an historical record of the project's development over time, enabling people to retrieve earlier versions. It also supports the possibility of the code "branching," meaning that alternative versions of the same code can be split off from the main project and maintained in parallel without causing conflicts. If someone wants to experiment with re-writing a certain section of a project, they can do so in a new branch while everyone else continues to use the main branch unaffected by the experiment.

The repository is a set of files in a directory structure that is maintained by the CVS server. Programmers submit updates and new files to the repository through a CVS client. This enables them to work remotely, with the CVS server acting as a central coordination point. Each entry in the CVS repository is represented by an individual file that maintains a record of both its content and changes. Other information relating to the project's development within the repository is stored as metadata. These enable logs of who has done what to be retrieved from the repository.

CVS was originally developed as a set of UNIX shell scripts by Dick Grune in 1984 as part of the Amsterdam Compiler Kit (ACK), a cross-platform C compiler developed at the Free University in Amsterdam. It was made public in 1986 and converted into C by Brian Berliner, from whose code the current version of CVS derives.[2] Other tools providing similar functionality, such as BitKeeper, also exist, and the new Subversion system is emerging as a possible replacement for CVS; however, CVS is currently the most widely used code management system.[3] In many ways CVS has been essential to the success of FLOSS (Free/Libre Open Source Software), as it facilitates the collaboration of the widely dispersed individuals who contribute to such projects. This facilitation, however, is restricted solely to the archiving of the code and its changes. Other aspects of development, such as communication between developers, are managed through tools such as mailing lists and IRC (Internet Relay Chat, or other online chat systems). Savane, used by the GNU project's Savannah repository, and Trac, are examples of larger

[1] Proudhon's original statement was: "The highest perfection of society is found in the union of order and anarchy." Pierre-Joseph Proudhon, *What Is Property? An Inquiry into the Principle of Right and of Government*, p. 286, translated from the French by Benjamin R. Tucker. The translation in the Dover edition has a slightly different phrasing from that used here.

[2] For historical documents on the development of CVS see: Wikipedia, Concurrent Versions System, available at http://en.wikipedia.org/wiki/Concurrent_Versions_System (accessed March 31, 2006), and Dick Grune, "Concurrent Versions System CVS." http://www.cs.vu.nl/ dick/CVS.html (accessed March 31, 2006).

[3] The main websites for the different tools are http://www.cvshome.org, http://www.bitkeeper.com, http://subversion.tigris.org.

toolsets that have been developed to pull these different components together.[4] Because CVS focuses on cohering code implementations, it is arguably not well suited to facilitating discussion of more abstract, conceptual aspects of a particular project. While mailing lists and IRC are often the forums for such discussions, they do not, by the very temporality of their nature, allow for such discussions to be built into identifiable documents. Similarly, comments in source code, while also facilitating this, can become too diffuse to gather such ideas together. The Wiki emerged as a response to this need, adapting the version control of CVS into a simpler web-based system in which the more conceptual modeling of projects could be developed and archived, exemplified in the very first Wiki, the Portland Pattern Repository, which gathered different programming "design patterns" together.[5] CVS, nevertheless, remains a central plenum within which the material origination of software is performed.[6]

Code creation is an inherently social act. It involves processes of collaboration, consensus, and conflict resolution, and embodies social processes such as normalization and differentiation. Software development tools such as CVS implicitly formalize such processes and, in doing so, potentially provide means of tracking them. As a result of this, forms of sociological analysis have developed based around "archaeological" studies of CVS repositories.[7] These studies revolve around questions of how FLOSS development actually works, especially given that it runs counter to many conventional models of product creation and production management. There is, for example, a lack of clearly delineated team structures in FLOSS projects; people can choose what they work on rather than being assigned jobs, there are frequently no project roadmaps or contractual deadlines, and you have a mixture of professional and amateur contributors, some working from within a paid capacity (such as in commercially or institutionally supported projects), others in their spare time.

Rather than following predefined managerial models, the practices and tools of FLOSS development facilitate emergent organizational structures. These can vary from one project to another, and may reflect aspects of the social situatedness of a given project, such as whether it is driven by institutional research, commercial development, or people with shared interests but no official affiliation. One recurrent form is described as an "onion structure,"[8] in which a reasonably stable core team of developers who are the main contributors and maintainers for a project is surrounded by layers of more occasional contributors and users. In some projects this may give a highly centralized shape to the overall social structure of the project, but in others there may be several such "core nodes" with an organizational form that is characterized by multiple interacting clusters. This latter formation is particularly evident in large-scale projects with many subareas, such as the KDE or GNOME desktop systems, or those that are largely driven by shared interest rather than institutional or commercial bodies.[9] Other studies describe a kind of guild structure, in which newcomers to a project have to serve a kind of apprenticeship and prove their capabilities before becoming accepted within the core development group.[10] Shadow networks

[4]For information on Savane see https://gna.org/projects/savane (accessed April 11, 2007). Savannah is the GNU project's main repository for free software development: http://savannah.gnu.org (accessed April 11, 2007). Trac is documented at http://trac.edgewall.org (accessed April 11, 2007).

[5]The Portland Pattern Repository is available at http://c2.com/ppr (accessed April 11, 2007). Design Patterns are high-level descriptions of how particular structures of code can be built, or problems in the design of software systems addressed. The notion of design patterns was derived from the architect Christopher Alexander's work on pattern languages in building design, see Christopher Alexander, et al., *A Pattern Language: Towns, Buildings, Construction*, New York: Oxford University Press, 1977.

[6]A plenum can be a "fully attended meeting" or a "space filled with material." In relation to CVS it carries both meanings: the gathering point of developers, and the space in which code is most evident in its material form.

[7]Examples of such studies include Kevin Crowston and James Howison, "The Social Structure of Open Source Software development teams," available at http://crowston.syr.edu/papers/icis2003sna.pdf, 2003; Stefan Koch and Georg Schneider, "Results from Software Engineering Research into Open Source Development Projects Using Public Data" in H. R. Hansen and W. H. Janko, eds., *Dikussionspapiere zum Tatigkeitsfeld Informationsverarbeitung unde Informationswirtschaft*; Gregory Madey, Vincent Freeh, and Renee Tynan, "Modeling the Free / Open Source Software Community: A Quantitative Investigation," in Stefan Koch, ed., Free / Open Source Software Development, pp. 203–220; Christopher R. Myers, "Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs," *Physical Revue E*; and Rishab Aiyer Ghosh, "Clustering and Dependencies in Free / Open Source Software Development: Methodology and Tools," *First Monday*, 8(4), April 2003.

[8]Crowston and Howison, "The Social Structure" p. 3.

[9]KDE and GNOME are two of the desktop environments available for the GNU/Linux operating system. For an analysis of the relationship between institutional interests and Open Source project development see Gilberto Camara, "Open Source Software Production: Fact and Fiction," in *MUTE*, volume 1, issue 27, Winter / Spring 2004, pp. 74–79.

[10]For a discussion of such issues see Biella Coleman, "The Politics of Survival and Prestige: Hacker Identity and the Global Production of an Operating System," available at http://healthhacker.org/biella/masterslongversion.html, Masters Thesis, University of Chicago, 1999; Biella Colemann, "High-Tech Guilds in the Era of Global Capital," available at

might also influence the social structures of a project, such as secondary affiliations constructed through ideological, institutional or corporate links.[11] These kinds of studies provide an understanding of agency and governance within FLOSS, and clarify how software development operates as a form of discursive formation.

As Foucault describes it, a discursive formation arises through the relations "established between institutions, economic and social processes, behavioural patterns, systems of norms, techniques, types of classification, modes of characterization"[12] that are not inherent within an object of discourse or practice itself, such as a piece of software, but is that which "enables it to appear, to juxtapose itself with other objects, to situate itself in relation to them."[13] All software is inherently discursive, it exists not as a set of discrete, stable artifacts, but rather as interrelated components, entering into various combinations with one another. This is evident both in the user experience of software and in how software is constructed. At a user level we can see this in the way in which a web browser, for example, will interact with various web server systems and the content tools they support, which may in turn feed into other pieces of software or computer-mediated processes. If I buy an air ticket online this will connect with other processes such as the management of my bank account and that of the airline company, and then in making my journey, the check-in process and management of the airport and air flight itself will utilize various software systems, all of which construct and articulate different relations "between institutions, economic and social processes, behavioural patterns . . ." etc. Similarly, no piece of software is a singular entity. The simple act of writing a piece of code involves the use of multiple software tools, such as text editors and compilers, but also issues such as which specific language the code is written in, whether it uses external code libraries and, if so, which choice of libraries, and what design patterns are followed in its construction. These derive not solely from pragmatic issues of functionality but also factors such as institutional alignment, the distribution and use of the final software, whether it operates by itself or as part of a larger system, and whether or not the source code will be made available for others to develop into and upon. Numerous decisions underlie the development of a software project: which language to develop in — whether to use Python, C or Microsoft's .Net, for example; what external code libraries to use (e.g., Apple's QuickTime library or the open source Simple Direct Media library); what kind of license to use — to distribute the code under an open source license that prohibits any commercial use, or one that allows the code to be used but not altered by others; and issues such as what file formats the software will support and what protocols it uses to interact with other software—will these be based on open standards such as SVG and HTTP, or on closed systems? The outcomes of such decisions are all influenced by the wider relations in which the production of the software is situated. The ways in which tools such as CVS are used will carry a residue of these factors, and the CVS repository can become a territory in which these issues and debates are inscribed. CVS is not simply a tool to manage the production of code therefore, but as "the space in which code emerges and is continuously transformed" (to paraphrase Foucault), also an embodiment and instrument of its discursive nature.

http://www.healthhacker.org/biella/aaapaper.html, undated; Warren Sack, "Aesthetics of Information Visualization."

[11] Madey, Freeh and Tynan, "Modeling the Free / Open Source Software Community," pp. 13–14.

[12] Michel Foucault, *The Archaeology of Knowledge*, Translated by Alan Sheridan Smith. London: Routledge, 1989, p. 45.

[13] Foucault, *ibid.*