Version Control Tools: A Collaborative Vehicle for Learning in F/OS

Maha Shaikh
Department of Information Systems
London School of Economics
m.i.shaikh@lse.ac.uk

Tony Cornford
Department of Information Systems
London School of Economics
t.cornford@lse.ac.uk

Abstract

In this paper we explore how version control software participates in learning within free/open source activities (F/OS). We see F/OS in terms of a product, and a community of people engaged in the process of its development, with version control software at the centre of all three activities as they learn and innovate. Learning is analysed through the perspective provided by Bateson's Levels of Learning, a relational model that stresses collaboration and conflict as drivers of learning and showing how conflict resolution may lead to higher and more profound or significant learning.

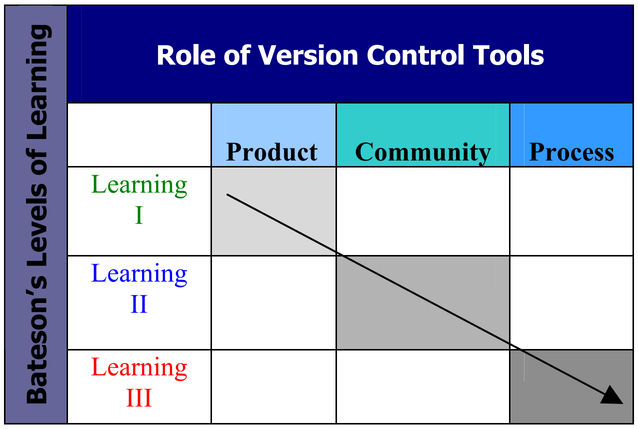
1. Introduction

It is usually taken for granted that version control software [VCS] is an indispensable part of free/open source [F/OS] development activity. Software patches are accumulated and exchanged via such tools, working systems are distributed, and developers manage some of their communication through them. However, VCS does more than just support development activity at the level of changing chunks of code. Here we ask what role VCS plays in learning in F/OS? F/OS can be seen as a combination of three elements: a *product* - software that is jointly worked on by developers and is available to a wider group of users; a *community* of developers centred around the product; and a *process* that they engage in

F/OS software *products* have certain characteristics that can be learned about or innovated, such as their architecture, criteria of quality, coherence and accessibility. Given Linus' Law, there is a particular need for transparency to allow review. *Community* refers to the norms of the developers, how they enter the community, their motivations for participation, the ideology that binds them together, and the levels of trust required to be self managing. Qualities that make a F/OS *process* appropriate include a high degree of responsiveness (release early, release often), to be inclusive, reliable and coherent (understood). Such distinctions are not absolute, but for the sake of clarity we use them here to identify various opportunities for learning that may be mediated to some degree through VCS.

The structure of the paper is as follows; in the next section we briefly introduce version control software. This is followed by an explanation of Bateson's learning ideas and a quick sketch of our case study and finally onto the analysis. The diagonal line in Table 1 indicates the path of analysis we take.

Table 1. Role for version tools in F/OS learning



Bateson's Levels of Learning	Role of Version Control Tools		
	Product	Community	Process
Learning I	▨		
Learning II		▨	
Learning III			▨

2. Version control software

Version control softwares are used by software developer communities in both proprietary and free/open source environments. Usually designated as a 'tool' they are conventionally defined as '*a mechanism for managing the multiple versions of the software objects that are created during the software development process* [4] and emphasise control, '*keeping track of the configuration items which are any documents created during a software development process, and which are found necessary to be placed under configuration control like requirements documents, data flow diagrams, design documents, source code, and test results* [7]. In contrast to a simple tool or control perspective, our view is to see VCS as an actor within a heterogeneous network of interests which it inscribes and translates.

Such software has long been a key part of open source activity with a history going back at least to the origins of UNIX [13]. F/OS communities, by their very nature, require them to manage software revision and release control within a multi-developer, multi-directory, multi-group environment [3, 6]. Tools such as RCS [Revision Control System] and SCCS [Source Code Control System] have long been in use [8], but Concurrent Version System [CVS], an OS product, is now recognized to be the most popular tool in F/OS communities [17].

3. Levels of learning

To understand how learning occurs and is manifested in F/OS we have adopted Bateson's concepts of levels of learning as our framework [2]. Argyris and Schon [1] base their organizational learning work on Bateson's ideas, adapting it to help understand collective or organizational learning. Bateson suggests that "*what can be studied is always a relationship or an infinite regress of relationships. Never a 'thing'*". This takes us away from learning as something that can be isolated but rather, learning occurs through relationships and collaborations [14]. VCS mediate such relationship building and it is central to this paper to see them as actors that are engaged within a learning process, promoting and reflecting learning. For Bateson learning is about some sort of '*change*', a *communication of ideas* and a *mastery of a new approach or solution* [11]. His view of learning is adaptive and ecological; 'learning is adaptation, and evolution is its highest form' [11], and emphasizes learning as a product of collaboration that is inherently social and relational. But any relational process involving collaboration also invites paradoxes and conflict. Such paradox and conflicts, and how they are understood or resolved, are proposed as the key to understanding how an individual or organization learns.

The version control software considered here has often been at the centre of conflict. For example, conflicts over patch submission may be resolved partly by VCS itself. It is, after all, named as a *control* device, and it does, through its process of managing software items, exercise control over development. The choice of VCS itself has also lead to explicit conflict, revealing the dynamic within F/OS communities and their conception of the process they engage in [13].

Bateson's presents learning as occurring at four different levels. At the lowest is Zero learning, which is hardly any learning at all. This involves simple reflex actions, or stimulus-response, which are needed to maintain a local status quo.

Learning I is the ability to learn to respond to diverse stimuli within the same or similar context – learning something. Learning II is a reflexive activity that allows activity to move across contexts and resolve paradox – learning about learning. Change of context and paradox resolution differentiates Learning I from Learning II; an ability to adapt responses and to consciously reflect on learning techniques and habits and improve them – often described as learning to learn. Reflection may be triggered by some crisis, breakdown or paradox that forces consideration of behaviour. This can lead to either reflection on learning habits in order to improve skills, or cause confusion and result in a state of limited reaction.

Learning III is understood as rarer and entails a 'profound reorganization of character' as for example in religious conversions. Learning III is a response to significant challenges to established theory or paradigms. It is an intriguing feature of open source communities that this form of learning lies at its core, and the Linux kernel community, for example, has over the last 10 years undergone a series of profound reorganizations when their learning paradigm has adapted.

4. Linux kernel case study

The F/OS case used here is the Linux kernel community. This community has a long history of concerns with VCS adoption and use [13]. They began with pre-patch releases and then part of the community adopted CVS. This VCS was not espoused by Linus Torvalds who, after some time, chose a closed source tool BitKeeper [BK]. This proved to be the catalyst of much distrust and dissent. It also lead to an attempt at a 'GPL'd clone' called BitBucket [9]. Torvalds backed BitKeeper and uses it to this day. His reasons for doing so, and the problems this caused, are discussed in Shaikh and Cornford [13].

The creator of BitKeeper, Larry McVoy, resented the BitBucket project and threatened to sue. At around this time he made 'amendments' to the BitKeeper license [BKL] making it impossible for any developer who had previously worked on 'competing' VCS to use BK. This threatened to become a serious dispute with the level of distrust rising.

Another issue that arose was the incompatibility between CVS and BK. Those developers who could not use BK, or did not wish too because of a conflict in ideology, were unable to access the main tree of the Linux kernel kept by Torvalds. McVoy took the opportunity to appease kernel developers by creating a gateway between BK and CVS. The situation seemed to get better only to end in further suspicion. BK's conversion of data into CVS did not add up completely to what was held by BK. What was available to the CVS users was not *all* the metadata and nothing McVoy said about the insignificance of the difference altered some kernel developers' resentment or mistrust.

5. Analysis

The brief history given in the section above will help the reader to understand the significance of VCS for F/OS communities. The case suggests immediately how such software mediates aspects of the community's governance structure, signifies access to code and metadata, and inscribes procedures for patch submission. Table 2, provides our framework for analysis of how VCS engage with activities of learning within this community and indicates areas of learning we identify within our study of the VCS debate within LKML. We only explore some of the framework here, taking a path to consider learning I in *product*, learning II in *community*, and learning level III in *process*.

5.1. Learning I in Product

VCS stores source code. In the case of BK, code is held as changesets, which are groups of deltas, accessible to developers to read and reuse. Both BK and CVS provide developers with an opportunity to read and compare code written by others and thus 'learn something'. A requirement for comparison may arise when more than one patch is sent to solve the same bug. In the case of BK the second developer is made aware of the duplication of effort and is then asked to determine which of the two patches should be accepted by looking at both patches side-by-side[5]. At a simple level, this comparison affords developers a window to observe *how* others would attack the same bug, but also allow the review of other code-use in other contexts. VCS also offers developers the chance to read metadata written by developers to explain their lines of code [12]. This, like the openness of source code, grants developers a chance to compare and learn.

5.2. Learning II in Community

A slightly different scale of comparison is offered by the ability to backtrack through deltas to see *how* change has occurred and what direction it took. This can support learning within the community through emulation and adapting of ideas. It is at the point of adaptation of ideas that this level of learning penetrates into the next, higher level of Learning II. It also moves from simple learning about code and coding to learning about aspects of participation within the community.

A community in F/OS clings to and seeks to sustain some ideology, has certain motivations, and relies on a culture built on trust and peer-review. Learning II indicates ability within such a community to adapt behaviour when the context changes and as a result of reflection. This level of learning is characterized by breakdowns, paradox situations or challenges that need to be addressed to regain some form of order.

As the brief history given above shows, VCS is seen to exercise control over the community and (in the case of BK) is structured to meet the needs of the governing body. One of Torvalds' main reasons for embracing BK was to control who had access to 'his' tree. He didn't want all developers to have access or the ability to load their patches without going through him first.

The community conflict over BK vs CVS, the gateway and the metadata, challenged many people to reflect on their current ways of working, the values they espoused, the norms of participation and development contribution. The context had changed and old behaviours or beliefs would not work. Such reflection demanded some new responses – for example, to GPL BK, to work through a gateway, or to engage in more debate and dissent.

5.3. Learning III in Process

In conventional software engineering terms VCS are all about process (control, tools), while in F/OS terms the process is itself a radical reorientation of software engineering. Here we want to analyze the role version control tools play in challenging fundamental assumptions and paradigms, and how learning at level III is manifested.

There is more than one instance of such a challenge and response in the Linux kernel case, for example in discussion of the purity of the GPL, ownership of codes, rules of participation and forking. Rare as this level of learning may be in individuals, it seems to be a recurrent theme in F/OS communities as they debate their existence and fundamental values. Here we focus on Torvalds decision to adopt BK [16] rather than CVS, remembering that in 1995 Torvalds had decided clearly that he would not use CVS for kernel development, though he was happy to use the tool for his work at Transmeta [15]. If this had been a move to a F/OS tools it still would have caused a stir, but the idea of using a closed source tool to create *the* kernel that stands for F/OS for many was an outrage. It divided developers between BK and CVS users and served as a catalyst for serious dissension and even a threat of a fork.

Using a closed source product to create their open source kernel threw the GPL into question and the sanctity of the code they produced. Faced with two equally distressing choices [or double-bind] of either forking the kernel or adopting a closed source tool gave rise to two responses. The community adapted in the form to create a GPL'ed clone of BK [BitBucket] so that they wouldn't have to use BK, and it also reinforced a sense of community and voice sufficient to persuade McVoy to create the BK→CVS gateway [10] which allowed CVS users indirect access to the source code and (some) metadata held in the BK repository.

6. Conclusion

Our contention is that F/OS activity is based on learning as a response to changing environmental factors and on the basis of experience. We suggest that such learning activity can be identified, for analytical purposes, within three main elements, the code itself, the community that surrounds it, and the process they engage in. Each element has to be learned about, each element poses problems and paradoxes that require some reflection on learning style or modes. At times, and more often perhaps than we thing, learning in F/OS is focused around a questioning of certain fundamental assumptions or core values, as in issues of license discussed here. In all this we see version control software playing a role within the heterogeneous network of interests. It is both a medium of learning (e.g. code visibility), but also a product of learning as it embodies or inscribes certain interests, while attenuating others. We see the community and the 'tool' mutually constructing one another as a learning activity

7. References

[1]C. Argyris and D. Schön, *Organization learning: A theory of action perspective*, Reading, Mass: Addison-Wesley, 1978.

[2]G. Bateson, *Steps to an ecology of mind*, New York: Chandler, 1972.

[3]B. Berliner, "CVS II: Parallelizing Software Development," in *Proceedings of Proceedings of the USENIX Winter 1990 Technical Conference*, Washington D.C., 1990.

[4]G. Clemm, "Replacing Version-Control with Job-Control," in *Proceedings of Proceedings of the 2nd International Workshop on Software configuration management*, Princeton, New Jersey, United States, 1989, pp. 162-169.

[5]V. Henson and J. Garzik, "BitKeeper for Kernel Developers," in *Proceedings of Ottawa Linux Symposium*, Ottawa, Ontario Canada, 2002, pp. 197-212.

[6]J.D. Herbsleb and R.E. Grinter, "Splitting the organization and integrating the code: Conway's law revisited.," in *Proceedings of 21st International Conference on Software Engineering (ICSE 99)*, Los Angeles, 1999.

[7]T. Kilpi, "New Challenges for Version Control and Configuration Management: a Framework and Evaluation," *IEEE Computer*, no. 1st Euromicro Working Conference on Software Maintenance and Reengineering (CSMR '97), 1997, pp. 33-41.

[8]H. Koike and H.-C. Chu, "Integrating Version Control and Module Management Using Three-Dimensional Visualization.," in *Proceedings of HCI*, San Francisco, California, USA, 1997, pp. 853-856.

[9]P. Machek, BitBucket: GPL-ed BitKeeper clone, University of Indiana, 2003. http://www.uwsg.indiana.edu/hypermail/linux/kernel/0302.3/0931.html

[10]L. McVoy, kernel.bkbits.net and BK->CVS gateway, University of Indiana, 2003. http://www.ussg.iu.edu/hypermail/linux/kernel/0305.1/0632.html

[11]W.J. Rothwell, "Bateson's Heterarchy of Learning," *Training and Development Journal*, vol. July, 1983, pp. 24-27.

[12]M. Shaikh and T. Cornford, "Version Control Software for Knowledge Sharing, Innovation and Learning in OS," in *Proceedings of Open Source Software Movements and Communities Workshop hosted by the International Conference on Communities and Technologies*, Amsterdam, The Netherlands, 2003, pp. 39-46.

[13]M. Shaikh and T. Cornford, "Version Management Tools: CVS to BK in the Linux Kernel," in *Proceedings of 25th International Conference on Software Engineering - Taking Stock of the Bazaar: The 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, 2003, pp. 127-132.

[14]S. Star and K. Ruhleder, "Steps toward an Ecology of Infrastructure: Design, Access for Large Information Space," *Information System Research*, vol. 7, no. 1, 1996, pp. 111-134.

[15]L. Torvalds, Re: CVS, Linus, and us, University of Indiana, 1995. http://www.uwsg.indiana.edu/hypermail/linux/kernel/9602/0800.html

[16]L. Torvalds, linux-2.5.4-pre1 - bitkeeper testing, University of Indiana, 2002. http://www.uwsg.indiana.edu/hypermail/linux/kernel/0202.0/0989.html

[17]A. van der Hoek, "Configuration Management and Open Source Projects," in *Proceedings of Proceedings of the 3rd International Workshop on Software Engineering over the Internet*, Limerick, Ireland, 2000.

Table 2. Framework to understand the role played by version control tools in F/OS learning

Bateson's Levels of Learning	Roles for Version Control Software		
Issues where learning may centre.	*Architecture* *Code quality* *Coding conventions* *Security*	*Membership* *Motivation* *Ideology* *Reputation and trust* *Norms*	*Responsive* *Inclusive* *Reliable* *Coherent*
	Product	**Community**	**Process**
Learning I *Learning 1 is an ability to understand and repeat behaviour in similar contexts.*	▪ Allow developers to compare and read the code ▪ Allows simple reading of metadata	▪ Informs the community about *how* to 'talk' to each other and communicate ▪ A vehicle for learning netiquette	▪ How to submit a patch or read code and deltas ▪ Defining software activities ▪ Defining software roles
Learning II *Learning 2 is the ability to adapt behaviour when the context is changed and is a result of reflection and paradox resolution.*	▪ Backtracking through deltas to see *how* change occurred ▪ Comparing patches to improve technique ▪ Insights into how 'elegant code' can be written	▪ Changing norms of communication. ▪ Use of peer review ▪ Conflict resolution activities.	▪ Inscribing the management of the software process ▪ Adding audit trails and accountability
Learning III *Level 3 is a more profound ability to challenge and change assumptions.*	▪ Visibility of code and architecture allow technical challenges ▪ User demands meet established practices	▪ Inscribing new community norms of governance and hierarchy ▪ Addressing questions of license which challenge openness and ownership of code	▪ Challenges to issues of openness and access to code and metadata ▪ Questioning participation ▪ Demanding new processes (gateway)