Unveiling distributed organizing in open source development: Practices of using, aligning, and wedging

Thomas Østerlie and Knut H. Rolland

Department of Computer and Information Science
Norwegian University of Science and Technology
Trondheim
Norway
thomas.osterlie@idi.ntnu.no, knut.helge.ronas.rolland@idi.ntnu.no

INTRODUCTION

Grounded in the undisputable success of open source software such as the Linux operating system, the Apache web server, and the Mozilla web browser, the phenomena of open source software and open source processes has recently been given considerable attention in various literatures (e.g. Feller and Fitzgerald, 2002; Fielding, 1999; Ljungberg, 2000; Moody, 2001; Raymond, 1999). One of the most characteristic aspects of open source projects is that the actual development of software is usually undertaken in large communities of globally distributed developers. For example, the Apache project concerned with developing a HTTP (web) server included a geographically distributed community of volunteers located in the U.S., Britain, Canada, Germany, and Italy (Fielding, 1999). Despite a highly distributed and heterogeneous environment and in deep contrast to usual recommendations given for systems development, open source projects can be characterized by the following (Mockus, Fielding, and Herbsleb, 2002: p. 310):

- Work is not assigned; people undertake the work they choose to undertake.
- There is no explicit system-level design, or even detailed design
- There is no project plan, schedule, or list of deliverables.

However, whereas these features of the open source process are widely known and the distributed nature of open source software development is frequently recognized as one of the main characteristics of open source (e.g. Feller and Fitzgerald, 2002; Ljungberg, 2000), sound empirical accounts and 'thick interpretive' case studies (e.g. Walsham, 1993) of how this distributed organizing is practiced and maintained over time in specific projects are still hard to

come by. Henceforth, given the puzzling evidence above, how do open source developers actually collaborate and coordinate their activities? And how do they keep track of changes, versions, and each other's activities?

A potentially valuable source of research that can shed light on the distributed organizing in open source projects are a body of literature concerned with understanding (commercial) software development in geographically distributed teams (Orlikowski, 2002; Sarker and Sahay, 2003) and studies of use of integrated information systems as an infrastructure for (globally) distributed work arrangements (Rolland and Monteiro, 2002; Star and Ruhleder, 1996). For example, Orlikowski (2002) describes in a case study of distributed software development how software developers' competence to do global product development is manifested in a specific repertoire of practices directly concerned with the distributed nature of organizing. Likewise, Rolland and Monteiro (2002) illustrate how the situated practices use of an integrated information system for distributing ship surveys are shaped according to the distributed nature of doing ship surveying. There are however, we argue, important differences between distributed organizing in business organizations as illustrated in the studies above and open source projects. First, the political and economical context of open source projects is very different from international corporations. Participants in open source projects are volunteers, and there are no formal managers, employers, and employees. Thus, while there are no traditional hierarchy and employer – employee relationships in open source projects, there are still other forms of control mechanisms. Second, although business organizations are doing distributed development, participants on a common project typically establish trust by interacting face-to-face and building and sustaining social networks across the organization (Orlikowski, 2002.). In contrast, developers on open source projects very seldom meet face-to-face. Henceforth, given their highly distributed nature (cf. Lane, 1998), it is important to unveil how open source projects establish trust – or alternatively suffers from lack of trust. Third, open source projects lack many of the typical arrangements to build a shared identity in globally distributed firms – e.g. common training of employees, workshops and social events (Orlikowski, 2002). On the other hand, many developers involved in open source projects share a common knowledge and skills of various software tools – e.g. CVS, e-mail, and newsgroups (e.g. alt.os.linux).

It is therefore important that we unveil the repertoire of practices that are involved in the distributed nature of open source projects. In this paper we aim at taking a closer look at how distributed organizing was achieved and practiced in the development of the Apache web server. We report from an interpretive case study of the Apache project, and focus on how a distributed way of organizing the development of the web server was achieved through a heterogeneous collection of different repertoires of practices and software tools (e.g. diff, patch, SMTP mail). Based on the empirical evidence, we argue that the practices of *using*, *aligning*, and *wedging* are the *sine qua non* for the global distribution of work in the Apache project. A second aspect of the Apache project worth attention is that occasionally, the distributed way of organizing seems to be substituted with a more centralized approach akin to that of Linux, and then after a period the distributed way of organizing re-appears.

CASE DESCRIPTION – THE APACHE PROJECT

The Apache project has its beginnings in a loosely organized group of developers and web masters sharing code patches for the unsupported NCSA web server. Version 1.3 of the NCSA web server was released in 1994, but later the same year a group of its core developers started their own company, Netscape. With their transition to Netscape, the NCSA web server was more

or less orphaned by the original developers left at NCSA. While buggy, the NCSA web server was at the time the most widely used web server on the Internet. With its origins from the University of Illinois, the NCSA web server shipped with its source code. Since the NCSA team no longer actively fixed bugs in the software, a group of users/developers took it upon themselves to fix bugs on their own, sharing patches among themselves. The patch or patch file is the difference listing produced by the Unix program `diff`. `diff` compares the contents of two files. One file is the base which changes to the second file are found. The output of `diff` can be stored in a file. This file is called a patch, so called because its contents can be used to update a copy of the base file with the Unix program `patch`. A patch is relative to the contents of the base file. Herein lays the problem facing the Apache developers. Once a patch is applied, the base file is disrupted. The first problem arises when several patches changes the same code blocks, making the patches mutually exclusive. The second problem is to handle patch inter-dependencies, when one patch relies on one or more other patches, which in turn might be mutually exclusive.

By 1995 dependencies among the plethora of patches available for the NCSA web server was becoming unwieldy. Drawing a baseline of the source code was required to resolve the situation. A mailing list was set up to coordinate the effort of drawing baselines, and create an alternative distribution of the web server incorporating not only the most critical bug fixes but also additional functionality. The new web server was called the Apache web server, a self-deprecating pun (read: "a patchy web server"). The group of developers soon started referring to their efforts as the Apache project.

All patches would be submitted and registered in a patch repository, which was simply a public FTP directory with full read and write access. The Apache members would download and apply patches from the repository to their own servers in order to test the patches. The servers used in testing were not dedicated test servers, but rather the participants own production servers. Not only were these servers serving a substantial number of connections, but they also provided critical business services.

In the continuous effort of drawing new baselines and applying patches to the source code, the Apache team agreed upon a decision-making system with the patch as its focal point. At irregular intervals, seldom more than a few weeks apart during the initial stages of the project, a participant would announce a vote. The vote announcement contains a list of patches that have been found to be working. Voting is given a deadline, upon which the votes are tallied. All Apache members are free to participate in the voting. Each patch in the vote announcement is given a numerical value between -1 and +1; +1 meaning the patch has been tested and found to be working and -1 being a veto. 0 is rarely submitted, but is intended to indicate the voter not knowing sufficiently about the patch in question to form an opinion about whether or not to integrate the patch in the code base. Each patch is given a vote, and an accumulated +3 is needed for a patch to make it into the next release. The veto vote, -1, though rarely used, means that the voter blocks the integration of the patch in question. This patch cannot be integrated into the next release no matter how many accumulated votes it would receive otherwise. Once voting is over, the approved patches are integrated with the code base and a new version of the web server is released.

The Apache project is initiated in March 1995, and by May the project has already arrived at release 0.6. Since mid-April the project has been showing strain. In late April one developer writes: "I'm getting more and more frustrated trying to wedge stuff into a large program that was not well architected … I am not proud of the patches I need to do, but there is no clean easy way of integrating the junk." Another developer observes somewhat later the same month: "Some time soon we'll need to take a look at redesigning Apache completely." The whole effort starts

collapsing during May 1992, as increasingly fewer patches are being submitted to the patch repository. By June no new patches are submitted to the patch repository, and a number of technical issues are arising with the 0.7 release. The project is practically pulled forward by a single obstinate developer, Rob Hartill, intent of resolving the pending issues with release 0.7. Development is grinding to a halt.

The project is almost dead when Robert Thau announces a garage project of his: a modular rewrite of the Web server in late June 1995. At first Thau's project is nothing but a curiosity, but quickly it is gaining attention in late July and early August. Thau's modular rewrite is a binary drop-in replacement of the existing Web server. Once done, it is supposed to provide the same services as the NCSA based Apache server. Participation picks up, but not towards improving the existing web server. Instead people start submitting patches to Thau's modular rewrite. The lack of interest in the original web server is so obvious, that Hartill provokes a confrontation resulting in the Apache project dropping its original web server altogether in favour of Thau's modular rewrite. Along with the original web server, the project also abandons the patch and vote system in favour of a centralized decision-making model with Thau at the helm. The argument for a centralized model is that it yields the control and speed required for developing the server from scratch.

The team works on the modular rewrite throughout August, porting the original web server's functionality over to this new platform. By the end of August, Apache 0.8 is released. With the release, the project returns to its original patch and vote system again.

DISCUSSION

The distributed nature of the Apache project involves a repertoire of practices. In this section we look closer of the practices of using, aligning and wedging, and show how the project's distributed way of organizing the web server development was achieved through these practices.

The *practice of using* may be understood as a variant of Orlikowski's (2002) practice of learning by doing. Through use, the Apache developers acquire knowledge of what bugs to fix and what new features should be added to the web server. Using the server as development platform the Apache developers acquire knowledge of how to change the source code to fix bugs and add new features. When it comes to knowing how to rewrite the software to better suit the kind of functionality the Apache developers want of the server as development platform, participation is a critical component. It is through both participating in the Apache developer community and through using the software that the Apache members develop an understanding of what must be improved for the server to better server their needs as a development platform. This is in line with Orlikowski's (ibid.) argument that the knowing constituted in the practice of participation means knowing how to innovate. The practice of using in the Apache project can be understood as a mix of both learning by doing and participation. As Orlikowski also points out, practices are "overlapping and intersecting through the specific activities engaged in by individuals" (ibid. p.267).

Due to the project's distributed nature, few Apache developers have ever met face to face. Face to face meetings builds and maintains strong social networks that generate trust (Orlikowski 2002). In the Apache project the trust of social networks are replaced by the practice of using. A majority of the project's participants in 1995 are either webmasters or work with ISPs themselves. They run Apache in business critical systems. Use becomes a quality control mechanism. The patch and vote system relies not only on the submission of new patches, but also upon people patching up their servers to test, and possible fix and/or improve, the existing

patches. The servers used in testing are real production servers, used to deliver business critical services. These are web servers receiving a lot of hits, servers whose operations are crucial. When patches are integrated in a new release, the participants are certain that the new code is already thoroughly tested. Nobody would vote for a patch that would break their servers.

Orlikowski (2002) argues that developing software systems successfully in distributed teams demand effective and ongoing aligning of product, projects and people across time and space. The *practice of aligning effort* can be observed within the Apache project as well. While the Apache project has no central authority to coordinate the developers and no project plan to be followed, there is still an effective and ongoing aligning of the development effort. Knowledge of who is working on what and how is central to the practice of aligning (ibid. p.268). The Apache developers have a de facto standardized tool set consisting of the Unix tools `diff` and `patch` and the GNU C compiler. In Orlikowski's case (ibid.) members of the organization "are able to make sense of (and modify if necessary) who is working on what … where, when and how" (ibid. p.268) by using common project management models and methodologies, and relying on standard contracts and metrics to annually assign developers to work on projects. The patch file, produced and used by respectively `diff` and `patch` from their standardized tool set, plays much of the same role within the Apache project.

The patch repository is used in sharing patch files, so the developers can keep up-to-date on who (the developer submitting the patch file) is doing what (the contents of the patch file) and how (by looking at the patched code). Frequent releases are used to keep the shared code base synchronized, which in turn to keeps patch dependencies at a minimum. The patch and vote decision-making system is used to determine what patches go in and which have yet to reach the required maturity. Where Orlikowski's practice of aligning effort is kept at the organizational level, the patch and vote system and frequent releases are examples of the technical side of aligning the development effort. By keeping the shared code base as closely synchronized as possible, the integration of patch files is eased. Voting is announced and votes are cast on the project's mailing list. The mailing list is together with the patch repository the main channel for aligning efforts within the project. Robert Thau's modular rewrite is an example of how the mailing list is used in aligning the development effort. Thau's announcements about the rewrite were pivotal for the entire Apache project's change of direction. Thau e-mailed status updates to the mailing list during July of 1995. The e-mails included download instructions for other developers to install and test his rewrite. With this information, the project's scope and direction was renegotiated resulting in the adoption of Thau's new architecture in favour of the original NCSA based code.

The practice of aligning is an ongoing effort, and the mailing list and patch repository's importance to the practice is apparent when developers fail to use them. An example of such a failure is the case of virtual hosting. Virtual hosting is a recurring theme on the mailing list from March 1995, but always in the form of being a nice feature for Apache to support. Then in mid-April the same year a developer announces he has started working on virtual hosting, but wants feedback from the rest of the team on two possible ways to implement the feature. Shortly after another developer says he has already implemented the feature, but a small problem with logging has kept him from submitting the patch to the repository. The second developer's failure to use both mailing list and patch repository in keeping others up to date on his work, leads to a situation were the efforts at implementing virtual hosting is unaligned. To align the effort, the second developer submits the unfinished patch to the repository. Alignment is restored, as the first developer drops his own effort into virtual hosting and instead downloads the unfinished patch to help out with that effort.

This episode shows another constituent of the Apache project's practice of aligning effort: the forming of ad hoc alliances. The Apache developers' primary mode of working can be labelled cooperative individual work, where each developer works individually, sharing his work products with others in the form of patch files. Where stricter control and coordination of specific activities are required, ad hoc alliances are formed where several developers work closely towards a common goal. The forming of ad hoc alliances is a way of aligning developers' efforts by coordinating specific activities and allocating resources to the activities. This allows for flexible movement of focus within the project. When the entire Apache group relinquishes the patch and vote system during initial stages of developing the modular rewrite, this may be seen as a pan-project ad hoc alliance motivated by the same needs for stricter alignment of efforts as the two man alliance formed in developing virtual hosting.

The *practice of wedging* is used to describe the activity of fitting functionality onto a piece of source code that wasn't initially designed to facilitate that kind of change. The metaphor of wedging is borrowed from carpentry, and used in contrast to fitting something meant to be there in place. Wedging is forcing a change through brute force, while fitting is a question of making a change that was intended all along. Wedging can be understood as a form of bricolage (Hannemyr, 1998). Wedging is the continuous adaptation of software artefacts through own use. It is a bottom-up activity, in contrast to having explicit system-level design, or even detailed design. Wedging can also be understood as using what is at hand to create new functionality. Instead of starting from scratch, the NCSA based source code is used. Although ugly, new functionality is wedged onto the existing code. The same form of minimalism is to found in the way the Apache project is organized. Simple tools are used effectively, without the burden of an all-encompassing groupware system.

In the borderland between wedging and use lies the dynamic that is important for innovation in the Apache project. By cooperating on wedging features onto the existing code, the Apache developers build a shared understanding of the existing source code's problems and shortcomings aligning their understanding of what is required of the code to support future functionality. Like the Apache project's practice of using and Orlikowski's (2002) practice of supporting participation, wedging as a practice fosters knowledge of how to innovate.

CONCLUDING REMARKS

The Apache project has been highly successful. At the time of writing Apache has a 63% share of the web server market (http://news.netcraft.com/archives/web_server_survey.html). This isn't due to the fact that open source software development is a silver bullet for software development, as some seem to believe. Apache's success is founded in a repertoire of practices that foster sharing, innovation and organizational flexibility. While the practices remain the same throughout the development effort, the way of organizing work actually shifts and changes with the tasks at hand. This shows that it isn't trivial to coin a single development approach as *the* open source development process. Even within the Apache project there is no such thing as a single process. Instead of searching for the one right way, the true open source development process, it would possibly be more fruitful to look at open source projects as rich data sources for research into software development in practice and distributed work.

REFERENCES

Feller, J. and Fitzgerald, B. 2002. *Understanding Open Source Software Development*. London: Addison-Wesley

Fielding, R.T. 1999. Shared leadership in the Apache project. *Communications of the ACM, Vol. 42, No. 4, pp. 38-39.*

Ljungberg, J. 2000. Open Source Movements as a model for organizing. *European Journal of Information Systems, Vol. 9, pp. 208-216*

Hannemyr, G. 1998. The Art and Craft of Hacking. *Scandinavian Journal of Information Systems, Vol. 10, pp. 1-2*.

Moody, G. 2001. *Rebel Code – Linux and the Open Source revolution*. London: Penguin Books

Mockus, A., Fielding, R.T., and Herbsleb, J.D. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology, Vol. 11, No. 3, pp. 309-346.*

Orlikowski, W.U. 2002. Knowing in practice: Enacting a Collective Capability in Distributed Organizing. *Organization Science, Vol. 13, No. 3, May-June 2002, pp. 249-273.*

Raymond, E.S. 1999. *The Cathedral & the Bazaar – Musings on Linux and Open Source by an accidental revolutionary.* Beijing: O'Reilly.

Rolland, K.H. and Monteiro, E. 2002. Balancing the Local and the Global in Infrastructural Information Systems. *The Information Society Journal, Vol. 18, No. 2., pp. 87-100.*

Sarker, S. and Sahay, S. (2003) *Understanding virtual team development: An interpretive study*, Journal of the Association of Information Systems, 4, 1-38.

Star, S.L. and Ruhleder, K. 1996. Steps Toward an Ecology of Infrastructure: Design and Access for Large Information Spaces. *Information Systems Research, Vol. 7, No. 1, pp. 111-134.*

Walsham, G. 1993. *Interpreting Information Systems in Organizations*. Chichester: Wiley.