2006

The distributed open source software development model: observations on communication, coordination and control

Bjorn Lundell
University of Skovde, bjorn.lundell@his.se

Brian Lings
University of Skovde, brian.lings@his.se

Par J. Agerfalk
Uppsala University, par.agerfalk@ul.ie

Brian Fitzgerald
University of Limerick, bf@ul.ie

THE DISTRIBUTED OPEN SOURCE SOFTWARE DEVELOPMENT MODEL: OBSERVATIONS ON COMMUNICATION, COORDINATION AND CONTROL[1]

Björn Lundell and Brian Lings, University of Skövde, P.O. Box 408, SE-541 28 Skövde, Sweden, {bjorn.lundell | brian.lings}@his.se

Pär J Ågerfalk and Brian Fitzgerald, University of Limerick, Limerick, Ireland, {par.agerfalk | bf}@ul.ie

Abstract

There are many reasons why an organisation should consider adopting distributed development of software systems and applications, including access to a larger labour pool and a broader skills base, cost advantages, and round the clock working. However, distributed development presents many challenges stemming from the complexity of maintaining good communication, coordination and control when teams are dispersed in time (e.g. across time zones) and space, as well as socio-culturally. The open source software (OSS) development model is distributed by nature, and many OSS developments are considered success stories. The question therefore arises of whether traditional distributed development models can be improved by transfer of successful practice from OSS development models. In this paper we compare OSS with traditional distributed development models using a framework-based analysis of the extant literature. From our analysis we find that the advantages of temporal and geographical distance dominate in OSS, rather than their associated problems. Further, socio-cultural distance is lowered by active developer selection. However, there is a challenge to satisfying project goals when personal goals dominate.

Keywords: Industrial Open Source, Open Source Development, Development Models, Global Software Development, Distributed Development.

[1] This research has been financially supported by the European Commission via FP6 Co-ordinated Action Project 004337 in priority IST-2002-2.3.2.3 'Calibre' (http://www.calibre.ie), and also by the Science Foundation Ireland Principal Investigator projects B4-STEP and Lero.

1 INTRODUCTION

Distributed development of software systems and applications (DD) is an issue of increasing significance for organisations today, all the more so given the current trend towards outsourcing and globalisation. However, as well as involvement in conventional DD many companies are getting involved in open source software (OSS) development projects, which have their own development models of DD.

The core challenges of DD seem to lie in the complexity of maintaining good communication, coordination and control when teams are dispersed in time (e.g. across time zones) and space, as well as socio-culturally. Since the OSS development model is distributed by nature, and many OSS developments are considered success stories, the question arises naturally as to whether lessons can be transferred between OSS and distributed development. However, proven methods for successful DD have not yet been formulated, and there is a need for a better understanding and transfer of lessons in relation to all distributed software development. As put by Crowston et al. (2005): "Understanding the work practices of teams of independent knowledge workers working in a distributed environment is important to improve the effectiveness of distributed teams and of the traditional and non-traditional organizations within which they exist." (p. 7)

In line with this sentiment, proponents of the Tigris.org development platform (www.tigris.org) claim that there is much to learn for traditional[2] DD projects by taking a closer look at OSS projects. As stated on the Tigris.org website, the "software engineering field can learn much from the way that successful open source projects gather requirements, make design decisions, achieve quality, and support users."

To facilitate such learning, this paper compares OSS development models with traditional DD models by means of a framework-based analysis of the extant literature on case studies in OSS development. From our analysis we derive the underlying characterisations of OSS development models in the literature. These characterisations can help to inform anyone involved in any kind of DD. As the analysis is based on a previously established framework for DD, our analysis will allow broad comparisons to be made.

2 BACKGROUND

For the purpose of this research, we take the position of Ågerfalk et al. (2005) in defining DD. A development team is distributed if its team members are not co-located, but geographically spread out. Here, *development* is interpreted broadly as *any software development lifecycle activity*. This thus extends beyond *pure* development activities and includes, for example, deployment and maintenance.

For a number of years the international workshop on Global Software Development (GSD) has highlighted the impact of distribution on *communication*, *coordination* and *control* within DD lifecycle activities (see, for example, Damian et al., 2003). This view is consistent with the position taken by a number of authors who have focused on one or more of these three fundamental processes (e.g. Carmel and Agarwal, 2001; Evaristo et al., 2004; McChesney and Gallagher, 2004; Nurmi et al., 2005; Sutanto et al., 2004). In particular, the communication, coordination and control activities are affected over a number of *dimensions*, which have been well elaborated in the literature (e.g. Battin et al., 2001; Espinosa and Carmel, 2003; Ghosh et al., 2004; Nicholson and Sahay, 2001; Sutanto et al.,

[2] We use the phrase "traditional distributed development models" when referring to approaches and models for conducting (non-OSS) distributed development in commercial contexts.

2004). These relate to temporal, geographic and socio-cultural distance. These processes and dimensions have been incorporated into a framework of issues in DD (Ågerfalk et al., 2005).

We will use this framework to present the results of our own study on Open Source development, and so introduce it briefly here (see table 1 below). Basically, *communication* refers to exchange of information. For communication to be successful, exchanged information should be complete and unambiguous so that sender and receiver can reach a common understanding (Carmel and Agarwal, 2001). The communication process concerns the transfer of knowledge and information between actors, and the tools used to facilitate such interaction. *Coordination* is "the act of integrating each task with each organisational unit, so the unit contributes to the overall objective." (Carmel and Agarwal, 2001, 23) The coordination process concerns how this interaction makes actors interdependent on each other. *Control* is "the process of adhering to goals, policies, standards, or quality levels." (Carmel and Agarwal, 2001, p. 23) The control process concerns the management and reporting mechanisms put in place to make sure a development activity is progressing. *Temporal distance* is a directional measure of the dislocation in time experienced by two actors wishing to interact. Temporal distance can be caused by time zone difference or time shifting work patterns. In general, low temporal distance improves opportunities for timely synchronous communication but may reduce management options. *Geographical distance* is a directional measure of the effort required for one actor to visit another at the latter's home site. Geographical distance is best measured in ease of relocating rather than in kilometres. In general, low geographical distance offers greater scope for periods of co-located, inter-team working. *Socio-cultural distance* is a directional measure of an actor's understanding of another actor's values and normative practices. As a consequence, it is possible for actor A to be socio-culturally closer to actor B than B is to A. It is a complex dimension, involving organisational culture, national culture and language, politics, and individual motivations and work ethics. In general, low socio-cultural distance improves communication and lowers risk.

A development context is considered distributed if it exhibits significant distance in the geographical dimension. We would consider a development team comprising members in two different offices in different cities within the same country to be distributed, even if they exhibit low temporal and socio-cultural distance. The key feature is that the cost (not necessarily monetary) to bring dispersed team members together is a significant inhibiter to spontaneous face-to-face meetings. When a DD project exhibits high distance in all dimensions, it is commonly referred to as a GSD project.

The complete framework, presented as table 1, forms a matrix in which each cell represents the impact of one dimension on one process. The table has been populated with an overview of the DD issues relating each process to each dimension (from Ågerfalk et al., 2005). This is the basis for our later comparisons.

3 RESEARCH METHOD

In this paper, we use the framework of table 1 as a basis for analysing the extant literature on case studies in OSS development. Our goal is to compare OSS development models with traditional DD models.

Based on an analysis of the published literature, we first review documented case studies in OSS which focus on process. Our initial goal is to characterise the contexts in which OSS development takes place. Following this, we consider the broader literature on reported OSS development experience. We look specifically at how reported experience relates to the framework of Ågerfalk et al. (2005), thereby systematically considering threats to communication, coordination and control in OSS development caused by Temporal Distance, Geographical Distance, and Socio-Cultural Distance. This results in populating the framework with the characteristics of, and work practices used in OSS development.

We then use the two populated frameworks to make observations on how OSS work practices relate to the issues in DD. For the literature analysis, systematic searches of the literature were made using

keyword and author searches, and searches of tables of contents of Journals, and Conference and Workshop proceedings. Bibliographic databases were used to assist in forwards and backwards referencing. Papers were included if they had a core focus on DD in OSS, or were considered highly relevant for understanding core issues raised in the literature. An extensive note file was also compiled, with quoted sections from papers which contained their major import. This allowed faster filtering in the later stages of analysis, but context was always checked against the full text. We illustrate the research process in figure 1, which is annotated to show the structure of the paper.

Process	Dimension		
	Temporal Distance	Geographical Distance	Socio-Cultural Distance
Communication	Reduced opportunities for synchronous communication, introducing delayed feedback. Improved record of communications.	Potential for closer proximity to market, and utilisation of remote skilled workforces. Increased cost and logistics of holding face to face meetings.	Potential for stimulating innovation and sharing best practice, but also for misunderstandings.
Coordination	With appropriate division of work, coordination needs can be minimised. Coordination costs typically increase with distance.	Increase in size and skills of labour pool can offer more flexible coordination planning. Reduced informal contact can lead to reduced trust and a lack of critical task awareness.	Potential for learning and access to richer skill set. Inconsistency in work practices can impinge on effective coordination, as can reduced cooperation through misunderstandings.
Control	Time zone effectiveness can be utilised for gaining efficient 24x7 working. Management of project artefacts may be subject to delays.	Difficult to convey vision and strategy. Communication channels often leave an audit trail, but can be threatened at key times.	Perceived threat from training low-cost 'rivals'. Different perceptions of authority/hierarchy can undermine morale. Managers must adapt to local regulations.

Table 1: An Overview of the Framework for Analysing DD (after Ågerfalk et al., 2005)

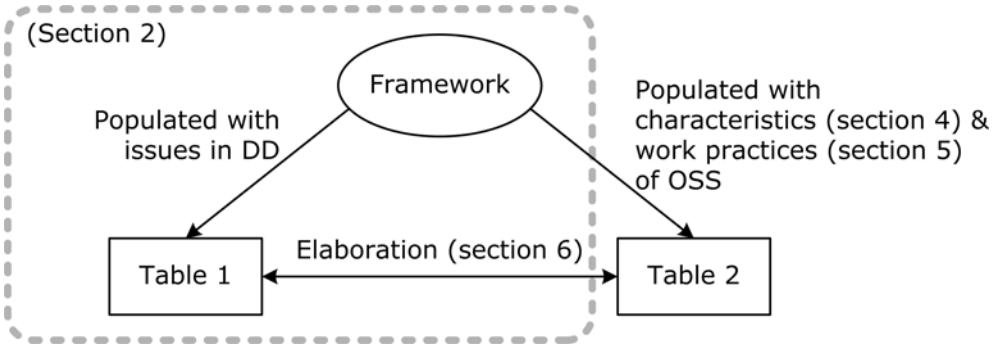

Figure 1: An Overview of the Research Process (with reference to paper structure)

4 VARIETY IN OSS DISTRIBUTED DEVELOPMENT CONTEXTS

OSS development is often characterised as occurring in a very different context from traditional development as managed in commercial companies. This is because the motivational factors can be very different (Bergquist and Ljungberg, 2001; Feller and Fitzgerald, 2002; Haruvy et al., 2003). However, it should be noted that the majority of contributors to OSS projects are now employed in

commercial companies (Ghosh et al., 2002). In fact, the relationship between companies and OSS projects is a complex and dynamic one, which must be considered in any attempt to reflect on OSS development – whether or not the DD aspects are of primary importance. Dahlander and Magnusson (2005) characterise three types of relationship: parasitic (in which the commercial interest is indifferent to its effect on OSS) – of great concern to the OSS community[3] as over-exploitation can threaten the "OSS ecosystem"; symbiotic (in which each gains advantage); and commensalistic (referring to a commercial interest not harming the OSS project). Apart from benefiting by improving an OSS product on which a company relies, a symbiotic relationship may result from less obvious benefits. For example, Lussier (2004) details an instance of process enhancement in a company brought about through the experience of its developers in an OSS development project.

To expand on this theme, we consider three case studies chosen because of their direct concern with issues related to communication, coordination and control in specific OSS projects. Although other case study papers have been published (see, for example, Dinh-Trong and Bieman (2004), Franke and von Hippel (2003), Koch and Schneider (2002), Mockus et al. (2002), Moon and Sproull (2000)) only the three studies described directly address the above issues.

Interestingly, the three studies address three very different contexts of OSS development, giving three different perspectives on contrasting models for OSS development with traditional DD. The first contrasts OSS development activities in a project with limited commercial input, with those of a commercial software development organisation (Persson et al., 2005). The second considers the phenomenon of employees of commercial companies paid to support OSS projects (German, 2003, 2003b). The third reports on the experience of an intra-company OSS project (Gurbani et al., 2005).

Process discovery in an OSS project has been identified as a challenging problem (Jensen and Scacchi, 2005b), but is as least made possible by the existence of detailed project information provided over the Internet. The ArgoUML project (argouml.tigris.org/) aims to develop a UML modelling tool with a good interface. Persson et al. (2005) analysed the process dimension of ArgoUML by tracking specific issues raised in developer mailing lists, module development mailing lists, Issuezilla and the project home page. They noted a lack of progress on certain stated goals, concluding that control and structure are weaker in ArgoUML than in a commercial context, for which "predictable time scales" is an important project goal. There is an industry view that you pay people to meet this goal, as it requires developers to sacrifice the freedom to work on what they find most interesting, but with the potential drawback of reducing enthusiasm. Roles within the ArgoUML project change unpredictably, changes being made by the project leader in response to core developer inactivity. Roles may also change in a commercial development environment, but they are more strictly under management control and largely fixed. This again has the potential for roles being misaligned with the interests of individuals, and so may lead to under-use of latent skills and enthusiasms. ArgoUML channels of feedback are open, allowing all interested contributors to monitor change. This is in contrast to commercial channels of feedback, which are usually more focussed, reflecting development structure. The latter has the advantage of shielding developers from "irrelevant" issues, related to other components; and the disadvantage of preventing potential contributions from outside a fixed team.

The GNOME project (www.gnome.org) aims to develop a free desktop for Unix. German (2003, 2003b) relates a case study of paid employees of companies contributing to the GNOME project. These developers are generally paid to contribute because their employer sees benefit in progressing the project at a steady pace. Hence, they pay for work which is less attractive to volunteers, such as project design, coordination, documentation and bug fixing. Such developers may even become dominant in terms of lines of code committed to the project CVS. They may also be active beyond code contribution, for example adding accessibility features. Volunteers still contribute as bug hunters, contributors and in specialised contributions such as internationalisation. However, straightforward

[3] A recurring concern raised by practitioners during the EU FP6 Calibre series of industrial conferences/workshops.

commercial involvement with OSS can have its drawbacks. Conflicts can arise when non-commercial members have reservations about a company's involvement, and one company's strong involvement can discourage the involvement of other companies (Jensen and Scacchi, 2005).

Within a commercial company, an open source project has been initiated to develop a server to support a standard telecommunications protocol. Gurbani et al. (2005) consider the case of the SIP server, initially developed by one developer within Lucent technologies, Inc. After initially maintaining the system in the light of user feedback, a decision was taken to release the source code within the company. This led to code suggestions, and then to contributions from many internal experts on optimisation issues, API design and compliance ideas. Ports were also made to other operating systems. A typical OSS cycle had been initiated, with interested user/developers contributing their updates so that they would survive future releases.

Such diversity of contexts for OSS suggests that OSS is not inherently a voluntary, distributed and Internet-based activity, although most successful OSS development projects benefit from these characteristics. Further, participation in OSS projects varies widely: only 5% of open source projects are developed by more than 5 developers (Zhao and Elbaum, 2003, p. 68), whereas the Debian project (www.debian.org) has around 1000 active developers (although the term 'developer' in Debian has a different connotation as the work is largely integration of software packages written elsewhere).

5 ON OSS DISTRIBUTED DEVELOPMENT MODELS

It is a widespread view that anyone who wants can contribute to an OSS project, and several well-known OSS projects have large numbers of developers. However, as noted above, it has also been observed that many projects have very few contributing developers. In fact, it has even been claimed that "most OSS products are developed and maintained by a tiny number of developers" (Krishnamurthy, 2002), and that the "community model is a poor fit for the actual production of the software." (Krishnamurthy, 2002) Irrespective of the size of an OSS project, the temporal, geographical and socio-cultural separation of developers has direct consequences for communication, coordination and control in a project.

Communication in OSS projects is largely asynchronous, with developers favouring the use of email (Sarma, 2005). According to Erenkrantz and Taylor (2003), this is because it allows participants to contribute on an equal basis in discussions, and such forms of communication lend themselves well to long-term archiving. However, there have been concerns raised in, for example, the Apache project that off-list discussions take place which are not always relayed back to the larger community (Ågerfalk, 2005). Similarly, Erenkrantz and Taylor (2003) note that the use of synchronous methods amongst distributed developers implies that "some participants may not be able to contribute to a discussion." (p. 23) Although face-to-face meetings in OSS projects are rare, there are some exceptions. For example, the GNOME OSS project has a yearly conference to get developers together (German, 2003). Similarly, the Zope community (www.zope.org) and the PyPy project (pypy.org) are known for their "sprints" where developers come together to work intensively on specific problems.

In OSS development projects developers are typically also users: a good argument for benefiting from DD. For example, in a case study analysis of the FreeBSD OSS project, it was noted that the "developers of FreeBSD were clearly users" (Dinh-Trong and Bieman, 2004). Some go further, for example claiming that "all open source developers are users, but not all users are developers" (Gacek and Arief, 2004, p. 35, 36), and even that it is a "necessity that developers must be users" (Mockus and Herbsleb, 2002, p. 3).

Gurbani et al. (2005) advance a slightly different argument, stressing the benefits of having "a significant pool of users who were interested and capable developers" which seems to be "a precondition for a successful open source project." (p. 28) The reason why developers must be users relates to the role of requirements in an OSS project, and the developers' assumed domain expertise "since there is generally no requirements gathering, and the developers are assumed to be domain

experts." (Mockus and Herbsleb, 2002, p. 1) In general, communications within an OSS project reflect the notion that developers are users, something which may have consequences for what can be done in this development model. As noted by Mockus and Herbsleb (2002), "only what this group of user-developers wants will actually get built." (p. 1) Of course, this is changing as paid employees play an increasingly central role in many OSS projects. It is also the case that many companies adopt OSS only as an embedded component in systems which are not used exclusively by developers (Ågerfalk et al. 2005b). In these cases, the communication processes become more complicated and more formalised.

As a final point on communication, there are many accounts in the literature of the benefits of openness, and in particular releasing code early, utilising the massively parallel peer-review process which ensues (e.g. Cubranic, 1999, p. 5). One advantage, noted by Lussier (2004), relates to the Wine project for implementing a Windows API on Unix. Through the mailing list, developers "quickly learned what their common errors were, and how to avoid them." (p. 70)

Coordination needs in OSS projects are minimised. Modularisation is generally seen as an important strategy for minimizing dependencies in software engineering, and even more so in DD. This has been strongly recognised by OSS researchers, "creating software with modules that can be worked on independently." (Crowston et al., 2005, p. 5) For example, GNOME (German, 2003) is "divided into several dozen modules, ranging from libraries (such as GUI, CORBA, XML, etc) to core applications (such as email client, graphical editor, word processor, spreadsheet, etc)" (p. 39), minimising the amount of communication between developers.

Mockus and Herbsleb (2002) raise the concern that "modularity may suffer" (p. 2) if decisions on new features in OSS projects become driven by market demands. Such concerns have been raised in the related area of traditional development tools, where Lundell and Lings (2004) have identified increased market pressures as a possible cause for vendors adopting "feature-driven" development, instead of longer-term visions for their products emphasising reliability and quality.

It has been argued that (free) tools based on standards should be adopted to support coordination in OSS development projects. However, support seems to be mainly for "low-level coordination" (Fielding and Kaiser, 1997, p. 89). Modern tools have not been widely adopted in the OSS community for supporting the software process. For example, it has been claimed that "CVS, now a de-facto standard version control system of medium-to-large open-source projects, is 10 years behind equivalent commercial offerings" (Cubranic, 1999, p. 4) We note that in a survey of 11 successful OSS projects (Erenkrantz and Taylor, 2003), CVS is used by all OSS projects (Apache HTTP Server, GNOME, GCC, Tomcat, KDE, Linux Kernel, Mozilla, NetBeans, Perl, Python, XFree86), although it is acknowledged that since the publication of the survey, Linux "had adopted BitKeeper as its source control system" (Erenkrantz and Taylor, 2003).

Control in OSS projects might be considered anathema. In DD projects in general, and in OSS projects in particular, it is important to acknowledge the potential tension generated by the need of participants to fulfil their goals. In the context of OSS, only when "private goals are being met will participation continue" (Erenkrantz and Taylor, 2003, p. 24). Essentially, when "a difference of vision occurs between developers and organizations, projects can be forked to maintain the integrity of the private goals." (Erenkrantz and Taylor, 2003, p. 24)

Different authority models have been adopted in the OSS community, and Linux has been recognized as an example of a "central authority organizational model" (Erenkrantz and Taylor, 2003, p. 25) By way of contrast, the GNOME OSS project has a board (GNOME Foundation), which meets regularly, usually over teleconferencing. Minutes are taken from each of these meetings, which are then published in the main GNOME mailing list (German, 2003, p. 42). Gurbani et al. (2005) use the term *benevolent dictator* when describing the final arbiter on what goes into the code while "preserving the architecture" (p. 28) of an OSS product. In the case study reported, which is an OSS project within a company, the benevolent dictator needs to consider the commercial goal of the company also, and not act according to 'blind' idealism for controlling the overall vision and architecture in the OSS project.

Several studies have stressed the importance of a strong leader for successful OSS projects, to maintain a unified architectural vision. Leadership in OSS projects is often based on reputation and respect. For example, Linus Thorvalds is often characterised as a strong leader of the Linux project – Moon and Sproull (2000) refer to him as a "great man", and others have commented on his devotion and strong role in spending a "considerable number of hours rewriting code submissions by others." (Krishnamurthy, 2002),

As noted above, although in most OSS projects participants are self-selected, it is becoming more common that developers are paid for their contributions. In fact, participation on a voluntary basis can be a way to obtain a paid job. For example, there are paid employees working in the GNOME project who were initially volunteers. In fact, German (2003) notes that: "Most of the paid developers in GNOME were at some point volunteers. Their commitment to the project got them a job to continue to do what they did before as a hobby." (p. 42) However, their change in status clearly has implications for control, which effectively moves into the company employing them.

In summary, when looking at communication, coordination and control issues in OSS it becomes clear that a number of characteristics emerge. Firstly, the advantages of temporal and geographical distance dominate, rather than their associated problems. Basically, routine development is asynchronous, Internet-based and assisted by having timely feedback from a large user base. There is no concept of having to bring distinct teams together for certain phases of a project, but some projects do utilise "sprint" meetings and developer conferences to increase cohesion and boost progress. Secondly, socio-cultural distance is low by virtue of active developer selection: a developer joins a project through choice, and because of a perceived alignment between a project and the developer's own aims. Thirdly, personal rather than project goals dominate, so control in an OSS project must be through maintaining a shared vision in which contributors' goals are enhanced through participation.

In table 2 we summarise the characteristics of OSS DD emerging from the case studies as they relate to the processes and dimensions identified in table 1.

Process	Dimension		
	Temporal Distance	Geographical Distance	Socio-Cultural Distance
Communication	Practically all routine communication asynchronous, through the Internet.	Typically, developers acting as the market. Internet used creatively for communication channels.	Responsive communities of motivated, self-selected contributors.
Coordination	Preponderance of modular, plug-in style architectures, reducing the need for coordination.	Dynamic and flexible labour pools. High critical task awareness throughout the community.	Common environments based on free, lightweight tools and lightweight process infrastructures.
Control	Typically 24x7 working. Control primarily through the commit process.	Mechanisms in place for identifying and addressing the issue of non-active key members.	Shared project goals and no project forking. Protection of OSS values. High level of activity on mundane tasks.

Table 2: Characterising active OSS DD activities within the framework of table 1

6 DISCUSSION AND CONCLUSIONS

In this final section, we elaborate on the characteristics of the OSS model of development, looking at it from the perspective of the issues raised in table 1. As in section 5, we structure the discussion according to the processes identified for DD.

6.1 On Communication in OSS projects

Routine communication in OSS projects is classically and primarily asynchronous and Internet-based. Projects are typically hosted using a collaborative development environment (CDE) such as SourceForge or Tigris.org. Such environments offer version control, issue tracking, mail lists, news and other project services. This asynchronous mode typically accounts for the majority of development activity. The use of a CDE brings with it the advantage of a total record of communications which utilise the CDE. Of course, as in traditional DD it is known that personal and other forms of non-CDE channels of communication are also used within OSS development; such communication naturally goes off the map.

Developers also sometimes use synchronous communication channels. For example, face-to-face meetings are sometimes used for deep technical discussions and code development. This can happen through several mechanisms, including “sprint” meetings, when developers committed to an OSS project get together for some days of intensive work on the project. It may also occur when a group of OSS developers is naturally co-located through their employment situation.

There have been a number of successful OSS projects at the infrastructure level, such as Linux, Apache and MySQL. What these have in common is a mutual understanding stemming from the fact that all developers are users. This inherently keeps the project close to the market, and motivates a keen interest in validation. For some OSS projects, in the application domain, this relationship may differ. It is unclear to what extent consensus may be achieved when a project is removed from the infrastructure level.

OSS communities are typically composed of highly motivated and self-selected contributors. At its best this results in a highly responsive community, responding quickly to contributions, whether code, enquiries or feedback. The public channels of feedback encourage a view of openness and responsiveness. However, problems can occur. A view can build of projects becoming elitist, discouraging new participation by novices in the project, or undervaluing activities not held in as high esteem as coding.

6.2 On Coordination in OSS projects

Many successful OSS projects have been designed with a highly modular, plug-in style of architecture. This reduces the need for coordination by reducing the number of task dependencies. It also reduces the barrier for potential contributors to become involved in the overall project. This increases the likelihood of maintaining a dynamic and flexible pool of contributors.

It is a feature of a number of well-documented successful OSS projects that critical task awareness is high, largely resulting from the fact that the developers are all themselves users. This phenomenon relies on project goals being shared by all project participants.

Much of the success of OSS stems from the global adoption of Internet technology, enabling a large pool of potential participants to interact and coordinate their activities using lightweight tools and common environments. There is a low entry cost to projects, both in monetary terms and in terms of learning the technology used for coordination by a community. It is therefore possible, and common practice, for contributors to participate in several projects.

6.3 On Control in OSS projects

With global participation, many OSS projects can take advantage of 24x7 working. A new release will undergo significant testing by the distributed and large user base in large, successful projects. Feedback can thereby be rapid, no matter how an individual contributor decides to time-shift.

Control of project code bases varies. One successful strategy has been to maintain it primarily through the commit process, which is typically reserved for a small number of core developers. However, commit then becomes a potential bottleneck, and mechanisms must be put in place for identifying and addressing the issue of a core developer becoming non-active. This is usually a trigger for the project leadership to reassign roles – perhaps implicitly recognising the level of commitment of a new contributor. It is important to any project to maintain a shared vision in order to keep motivation high and protect against project forking. It is arguably easier to maintain a common understanding of project goals and requirements when a project concerns the infrastructure level, which may explain why the most cited success stories are at that level.

OSS values have evolved over a long period of time, being first articulated by the Free Software Foundation. Over time, commercial interest in OSS has risen significantly to the point at which major players in the IT business, such as HP (Debian linux distribution), IBM (the development platform Eclipse) and SUN Microsystems (GNOME unix desktop), have invested significant resources in OSS projects. Further, many companies have offered packaged OSS products for commercial gain. Such commercial involvement, if not handled sensitively, can undermine the OSS values in a project and de-motivate volunteers, threatening the vitality of the project. In order for a company to build a business model around OSS projects it is critical for it to build a deep understanding of OSS values and to act in a way which protects the dynamics of the larger community.

For a project leader, one of the major concerns is to ensure activity on mundane or low-prestige tasks. One of the reasons why companies are welcomed in an OSS project is that they may ensure that these tasks are carried out, perhaps through some kind of non-salary reward system.

6.4 Conclusions

In this paper we have compared OSS with traditional distributed development models using a framework-based analysis of the extant OSS literature. We have found that the advantages of temporal and geographical distance dominate in OSS, rather than their associated problems. This is achieved by making routine development asynchronous, with timely feedback from a large user base. Further, socio-cultural distance is lowered by active developer selection. Every developer has a personal stake in the success of the project. The low entry cost to projects, both in monetary terms and in terms of learning the technology used for coordination by a community, also makes it possible for contributors to participate in several projects. This could significantly increase flexibility in prioritising development effort between projects.

However, there is a challenge for OSS in relation to satisfying project goals when personal goals dominate. It is unclear to what extent goals can be unified when a project is removed from the infrastructure level, to a level where consensus is more difficult to achieve. How much this factor may dominate in OSS development has yet to be established.

More deep case studies are needed to enable cross-fertilisation of ideas throughout traditional distributed and OSS development. Such studies would allow further development of the characterisation of OSS presented here, with a view to informing best practice throughout distributed development. However, although the framework of Ågerfalk et al. (2005) has proved useful in this characterisation, we believe that it can be further improved. In particular, we see advantage in extending the framework with two further dimensions: successful strategies associated with each cell in the framework; and technologies offering leverage in dealing with each issues related to each cell. By a process of continuous development of such a framework, best practice in distributed development will be made more accessible to both industry and the OSS community.

References

Ågerfalk, P.J. (2005). Studying Persistent Conversations in an Open Source Context: A Conceptual Framework, In Proceedings of Promote IT 2005, (Eds, Bubenko jr., J et al.) Studentlitteratur, Lund.

Ågerfalk, P.J., Deverell, A., Fitzgerald, B. and Morgan, L. (2005b). Assessing the Role of Open Source Software in the European Secondary Software Sector: A Voice from Industry, In Proceedings of the 1st International Conference on Open Source Systems (Scotto, M. and Succi, G. Eds.), p. 82-87, Genoa, Italy.

Ågerfalk, P.J, Fitzgerald, B., Holmström, H., Lings, B., Lundell, B. and Ó Conchúir, E. (2005). A Framework for considering Opportunities and Threats in Distributed Software Development. In Proceedings of the International Workshop on Distributed Software Engineering, p. 47-61, Austrian Computer Society.

Battin, R.D., Crocker, R., Kreidler, J. and Subramanian, K. (2001). Leveraging resources in global software development. IEEE Software, 18 (2), 70-77.

Bergquist, M. and Ljungberg, J. (2001). The Power of Gifts: Organizing Social Relationships in Open Source Communities, Information Systems Journal, 11, 305–320.

Carmel, E. and Agarwal, R. (2001). Tactical approaches for alleviating distance in global software development. IEEE Software, 18 (2), 22-29.

Crowston, K., Annabi, H., Howison, J. and Masango, C. (2005). Effective work practices for FLOSS development: A model and propositions. In Proceedings of the 38th Hawaii International Conference on System Sciences – 2005, p. 1-9, IEEE Computer Society.

Cubranic, D. (1999). Open-Source Software Development. In 2nd Workshop on Software Engineering over the Internet, 7p.

Dahlander, L. and Magnusson, M.G. (2005) Relationships between open source software companies and communities: Observations from Nordic firms. Research Policy, 34, 481-493.

Damian, D., Lanubile, F. and Oppenheimer, H.L. (2003). Addressing the Challenges of Software Industry Globalization: The Workshop on Global Software Development, In Proceedings 25th International Conference on Software Engineering, p. 793-794, IEEE Computer Society.

Dinh-Trong, T. and Bieman, J.M. (2004) Open Source Software Development: A Case Study of FreeBSD. In Proceedings of the 10th International Symposium on Software Metrics, IEEE Computer Society.

Erenkrantz, J.R. and Taylor, R.N. (2003). Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community. In The 1st Workshop on Open Source in an Industrial Context, p. 21-30, <wwwbib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0319.pdf>

Espinosa, A. and Carmel, E. (2003). The Impact of Time Separation on Coordination in Global Software Teams: a Conceptual Foundation. Software Process Improvement and Practice, 8, 249-266.

Evaristo, J.R., Scudder, R., Desouza, K.C. and Sato, O. (2004). A dimensional analysis of geographically distributed project teams: a case study. Journal of Engineering and Technology Management, 21 (3), 175-189.

Feller, J. and Fitzgerald, B. (2004). Understanding Open Source Software Development, Addison-Wesley, London.

Fielding, R.T and Kaiser, G. (1997). The Apache HTTP server project. IEEE Internet Computing, 1 (4), 88-90.

Franke, N. and von Hippel, E. (2003) Satisfying heterogeneous user needs via innovation toolkits: the case of Apache security software. Research Policy, 37 (2), 1199-1215.

Gacek, C. and Arief, B. (2004). The Many Meanings of Open Source. IEEE Software, 21 (1), 34-40.

German, D.M. (2003). GNOME, a case of open source global software development. In International Workshop on Global Software Development, p. 39-43, <gsd2003.cs.uvic.ca/gsd2003proceedings.pdf>

German, D. M. (2003b). The GNOME Project: a Case Study of Open Source, Global Software Development. Software Process Improvement and Practice, 8 (4), 201-215.

Ghosh, T., Yates, J.A. and Orlikowski, W.J. (2004). Using Communication Norms for Coordination: Evidence from a Distributed Team, In 2004 – Twenty-Fifth International Conference on Information Systems, p. 115-127, Association for Information Systems.

Ghosh, R., Glott, R., Krieger, B. and Robles, G. (2002). FLOSS Final Report – Part 4: Survey of Developers, University of Maastricht, The Netherlands <www.infonomics.nl/FLOSS/report>

Gurbani, V.K., Garvert, A. and Herbsleb, J.D. (2005). A Case Study of Open Source Tools and Practices in a Commercial Setting. In Proceedings of the 5th Workshop on Open Source Software Engineering, p. 24-29, ACM.

Haruvy, E., Prasad, A. and Sethi, S.P. (2003). Harvesting Altruism in Open Source Software Development. Journal of Optimization Theory and Applications, 118 (2), 381-416.

Jensen, C. and Scacchi, W. (2005). Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community. In Proceedings of the 38th Hawaii International Conference on System Sciences – 2005, 10p, IEEE Computer Society.

Jensen, C. and Scacchi, W. (2005b). Experience in Discovering, Modeling, and Reenacting Open Source Software Development Processes, In Unifying the Software Process Spectrum: Proc. Software Process Workshop (Mingshu, L. et al. Eds.), Beijing, China, May, Springer, 2005 *(to appear).*

Koch, S. and Schenider, G. (2002). Effort, co-operation and co-ordination in an open source software project: GNOME. Information Systems Journal, 12 (1), 27-42.

Krishnamurthy, S. (2002). Cave or Community? An Empirical Examination of 100 Mature Open Source Projects. First Monday, 7 (6), <www.firstmonday.org/issues/issue7_6/krishnamurthy/>

Lundell, B. and Lings, B. (2004). Changing perceptions of CASE-technology. Journal of Systems and Software, 72 (2), 271-280.

Lussier, G. (2004). New Tricks: How Open Source Changed the Way My Team Works. IEEE Software, 21 (1), 68-72.

McChesney, I.R. and Gallagher, S. (2004). Communication and co-ordination practices in software engineering projects. Information and Software Technology, 46 (7), 473-489.

Mockus, A. and Herbsleb, J.D. (2002). Why Not Improve Coordination in Distributed Software Development by Stealing Good Ideas from Open Source?. In Meeting Challenges and Surviving Success: The 2nd Workshop on Open Source Software Engineering, p. 19-25, <opensource.ucc.ie/icse2002/MockusGerbsleb.pdf>

Mockus, A., Fielding, R.T. and Herbsleb, J.D. (2002). Two Case Studies of Open Source Software Development: Apache and Mozilla. Transactions on Software Engineering Methodology, 11 (3), 309-346.

Moon, J.Y. and Sproull, L. (2000). Essence of Distributed Work: The Case of the Linux Kernel. First Monday, 5 (11), <www.firstmonday.org/issues/issue5_11/moon/>

Nicholson, B. and Sahay, S. (2001). Some political and cultural issues in the globalisation of software development: case experience from Britain and India, Information and Organization. 11 (1), 25-43.

Nurmi, A., Hallikainen, P. and Rossi, M. (2005). Coordination of Outsourced Information System Development in Multiple Customer Environment – A Case Study of a Joint Information System Development Project. In Proceedings of the 38th Hawaii International Conference on System Sciences – 2005, p. 1-10, IEEE Computer Society.

Persson, A., Lings, B., Lundell, B., Mattsson, A. and Ärlig, U. (2005) Communication, coordination and control in distributed development: an OSS case study. In Proceedings of the 1st International Conference on Open Source Systems (Scotto, M. and Succi, G. Eds.), p. 88-92, Genoa, Italy.

Sarma, A. (2005). A Survey of Collaborative Tools in Software Development, ISR Technical Report # UCI-ISR-05-3, Institute for Software Research, University of California, Irvine <www.isr.uci-edu/tech-report.html>

Sutanto, J., Kankanhalli, A. and Tan, B.C.Y. (2004). Task Coordination in Global Virtual Teams. In 2004 – Twenty-Fifth International Conference on Information Systems, p. 807-819, Association for Information Systems.

Zhao, L. and Elbaum, S. (2003). Quality assurance under the open source development model. Journal of Systems and Software, 66 (1), 65-75.