Andrea Hemetsberger; Christian Reinhardt

The Invisible Coat-tail. Successful Collaboration in Open-Source Communities

2010

https://doi.org/10.25969/mediarep/19823

Veröffentlichungsversion / published version
Sammelbandbeitrag / collection article

Empfohlene Zitierung / Suggested Citation:
Hemetsberger, Andrea; Reinhardt, Christian: The Invisible Coat-tail. Successful Collaboration in Open-Source Communities. In: Theo Hug, Ronald Maier (Hg.): *Medien – Wissen – Bildung. Explorationen visualisierter und kollaborativer Wissensräume.* Innsbruck: Innsbruck University Press 2010, S. 165–176. DOI: https://doi.org/10.25969/mediarep/19823.

The Invisible Coat-tail: Successful Collaboration in Open-Source Communities

Andrea Hemetsberger & Christian Reinhardt

Abstract

Online collaboration is often organized without strong predetermined rules or central authority, which is why coordination and ways of organizing cooperation become crucial elements of collaboration. This article investigates how online projects can overcome problems of dispersed work, solve inherent contradictions, and utilize tensions in the activity system to develop collaborative artifacts and practices. To this end, we introduce cultural-historical activity theory as our theoretical framework. We introduce the notion of 'coat-tailing' – a term used to denote the parallel pursuit of individual and collective objectives – as a successful mechanism for online coordination and cooperation in co-configurative (Engeström 2004) online projects. Empirical evidence is based on a detailed observation of a successful open-source project – the K Desktop Environment (KDE). Our findings show that work tools and rules are designed to match individual expectations and to enable individual activity, where the collective activity is achieved by 'piggybacking' on the fulfilment of the individual task. In other words, coat-tailing systems enable 'doing just one thing together'.

Introduction

The Internet has opened up new space for boundary-spanning collaboration and has generated invaluable technological improvements for the coordination of online projects. In such online work groups, work is often organized without strong predetermined rules or central authority, which is why coordination and ways of organizing cooperation become crucial elements of collaboration. This article introduces the notion of coat-tailing as a successful mechanism for online coordination and cooperation in co-configurative (Engeström 2004) online projects that integrate users and other external actors. The central argument is that online cooperation is not only a matter of task coordination but rather a question of overcoming tensions that derive from the alignment of strategic activity and individual action within a highly dispersed group.

We develop the argument based on a case study of an open-source (F/OSS) project – the K Desktop Environment (KDE) – that employs a complex activity system for coordination and cooperation. This article is intended to shed light on how co-configurative projects can overcome problems of dispersed work, solve inherent contradictions, and utilize tensions in the activity system to develop collaborative artifacts and practices. Based on activity theory and F/OSS-related research, we analyze the complex activity system of the KDE project in-depth, and discuss the ways in which our findings contribute to activity theory and online collaboration.

F/OSS communities as activity systems

Activity theory defines activity as the central unit of analysis (Vygotsky 1978). Cultural-historical activity theory argues that social practice should be understood as tool-mediated activity (Leontiev 1978; 1981; Cole 1996). This idea of *mediation* via tools is central to activity theory (Kaptelinin 1996). At the primary level, tools include physical tools which mediate people's thoughts and behavior. Conversely, people's thoughts also shape technological artifacts and their usage. More recent conceptualizations of activity theory have moved to a collective, artifact-mediated and object-oriented definition of the activity system (Engeström 1997; 1999; Nardi 1996; Cole et al. 1997). Engeström introduced the *community* as the collective which is interested in an object, *rules* which mediate the relationship between a community and the subject of an activity, and *division of labor* as the way the community is related to the object of the activity.

Activity theory suggests three interrelated levels of interaction – coordination, cooperation, and co-construction (Engeström 1997; Bardram 1998). Coordination ensures that *what* people are doing independent of each other results in the achievement of the common task (Engeström 1987). Cooperation concerns *how* coordination is achieved, and involves the social interaction of group members when doing things together. Co-construction corresponds to the re-elaboration or re-construction of work practices and demands reflective communication on a meta-level (Engeström 1987). At this level, work itself is the subject of contemplation (Barthelmess and Anderson 2002). By distinguishing these levels, activity theory draws particular attention to these boundaries and how they are managed in work groups. In order to manage these boundaries, artifacts need to be established that serve as *anchors* between different levels.

Activity theory is an approach that explicitly focuses on the dialectical aspects of the activity system. Research is directed towards contradictions within the system, and discourse as an important catalyst for change and the co-construction of meaning (Wells 2002; Engeström & Blackler 2005). Through collective reflection, cooperative processes become visible and give rise to awareness for the need for improvement. By focusing on contradictions in existing activity systems, activity theory is particularly well-suited to identifying how complex and highly dispersed activity systems cope with these contradictions and how they use them for improvement.

In an attempt to address new forms of collaboration, Engeström (2004) integrated the concept of co-configuration work into activity theory. Co-configuration is a *participatory* model that is not confined to collaboration between professionals, and integrates users as active subjects. F/OSS projects are prototypical examples of co-configurative work, including professionals, experts, and users. Collaboration among people with such varying expertise necessitates a dynamic, dialogic relationship between multiple actors; it is a relationship characterized by collaborative and discursive construction of tasks (Engeström 2004).

In co-configuration work, participants are required to recognize and engage with different goals of action and different expertise distributed across group members. Work in F/OSS

projects is voluntary; task assignment and decisions cannot be enforced (Demil & Lecoq 2006). Hence, conflicting goals of different actors could impede the pursuit of the activity and the achievement of the object. As a consequence, it is easy to lose sight of the overall objective in such complex collaborative online projects (Blackler et al. 2000). F/OSS researchers have reported concordantly that openness of source code and open communication in mailing lists is considered key to successful coordination. Parallel and overlapping activities are possible without losing sight of the current version of their work (Yamauchi et al. 2000; Kuk 2006). Tasks are of modular character and are self-selected by contributors according to their expertise and interests (Lanzara & Morner 2003; Lee & Cole 2003). Although openness and modularity decreases complexity for the individual, it increases the complexity of the overall activity. Most researchers agree that parallel software development, peer review, and user involvement combined with 'openness' are the most important ingredients of F/OSS collaboration (Feller & Fitzgerald 2001). What has been left open is how F/OSS projects cope with the tensions that the pursuit of individual goals and collective activity entails. This study aims to explore this question in order to gain important insights for the design of co-configurative work in highly dispersed groups.

Methodology

This case study is part of a bigger F/OSS research program. Research proceeded in several phases. First, a case was selected that provided deep insight into the collaboration process. Secondly, data had to be gathered over a sufficient period of time in order to achieve theoretical saturation (Goulding 2002). Thirdly, and in parallel with data generation, the research team was constantly involved in writing memos, coding, discussing emergent categories and interpretations, and analyzing data. Prior to the selection of an appropriate F/OSS community for investigation, the researchers defined three requirements. In accordance with Crowston et al. (2004), we used (1) the time of existence, (2) the number of members, and (3) the rate of innovation and diffusion, as indicators. After an extensive exploration of other open-source projects and careful consideration, we ultimately selected the K Desktop Environment (KDE) project for our research purpose. KDE is a desktop environment for UNIX workstations, similar to those found under MacOS or Microsoft Windows. KDE is one of the largest F/OSS projects in the world. More than 1000 developers have contributed over four million lines of code.

We applied a grounded theory approach (Goulding 2002; Charmaz 2006) adding non-participatory elements from Kozinet's (2002) netnography. The research team closely monitored the project community for a four-month period in order to gain a deep understanding of what they were doing. We observed the community regularly, included external open-source affiliates in our discussions and attended F/OSS conferences in order to observe their culture and grasp their technical jargon.

The primary data source were the core developers' mailing list, which is used for discussion about the development of the main developer tools and strategic activities, and the developer list, which is used for general discussions regarding KDE source code development. For the purpose of studying co-construction, we additionally sent short email questionnaires to four

core members, including two developers, who initiated the change, and two, who are responsible for the version control system. Their answers informed our data and interpretation with motives for the initiation of this major change, with technical details, timing issues, and explanation of rules.

Findings

New member integration

The integration of expert users is at the core of co-configurative work, and has obvious advantages for F/OSS communities. First, user feedback is integrated into innovative software solutions. Secondly, it increases opportunities by enlarging the workforce. However, for user integration to be successful, co-configurative workgroups have to cope with some inherent contradictions. Users must be made aware of the opportunities to contribute. Yet, users have different expertise with regard to usage, and programming. Hence, co-configurative work needs educational facilities starting at different levels. While communities of practice apply an apprenticeship approach where mentor and apprentice work closely together, F/OSS project members have to rely on remote training and interactive help from more experienced programmers. However, attracting new members should not interfere with the primary task of programming. The goal of user integration, therefore, is to integrate aspirant members with different expertise, but to avoid distraction from pursuing core activities. Furthermore, F/OSS projects have to solve the tension between volunteer work and task prioritization. The KDE community tries to solve these contradictions through ***designing the cultural entrée***, ***providing learning opportunities*** at different levels, and integrating users in ***continuous improvement***.

Aspirant members' first contact with the KDE community is usually established through the project's online contents. A conglomeration of detailed descriptions and rules allows the newcomer to become familiar with the community, its culture, and its activities. KDE is an open, but also a political-ideological culture in its basic tenets. As the objective and typical working style of F/OSS projects is firmly based on its cultural values, cultural discourse is particularly present in communication with newcomers. Here the community seeks to avoid distraction. The technologies used are simple: publicly observable sites, FAQs (frequently asked questions), and discourse. Aspirant developers also benefit from discursive events, stored chronologically in mailing list archives; they are encouraged to *observe* common practice and discourse before they become a member. Presenting discourse in a sequential order enables internalization of the cultural norms of the group and their way of thinking. Open communication enables newcomers to *re-experience* community history – which brings aspirant members a step closer to contributing.

The goal is to provide help for different levels of expertise and to offer different opportunities to participate. KDE supports the first steps of integration by providing tutorials. In contrast to documentation, which only provides an abstract description of how things work, tutorials are much more oriented towards the activity of using work tools and coding itself. Once newcomers are familiar with the community's technology and programming style, they engage in

small, simple tasks according to their level of expertise. F/OSS communities must work with the constant contradiction of having a voluntary workforce while at the same time getting things done that have to be done. Therefore, the community must find ways in which to encourage participation in activities which contribute to the common objective. Task prioritization is partly fostered by discourse which conveys the message that contributors should bear the community's overall objective in mind. Initial contributions are further encouraged by propagating a pragmatic approach of *doing*. It fosters, of course, learning by doing, but also prevents newcomers from doing 'invisible' work. This particular meritocratic approach is enabled by several work tools, including bug tracking technology. The specific way F/OSS projects handle bug fixing is reputed to be one of the main advantages of their development model (Kuk 2006). The primary advantage of the bug tracking system is that the detection of bugs and bug fixing are split among individuals with different expertise. This makes the system exceptionally well-suited to fostering self-selection of high priority tasks. The bug tracking system is the work tool that enables advanced users to report a bug or a missing feature in a way that enables developers to fix it. User voting for the most hated defect or the most desired feature further reflects the severity of the problem, which helps with the prioritization of tasks. New members benefit from receiving comments on rejections, much like in academic reviewing. This results in the continual improvement not only of code, but also of developers' skills.

Developers who have reached an acceptable level of competence are invited to ask for write access to the source code repository so they can integrate patches to the source code. Being granted write access entails strong social symbolism. It engenders pride on the side of the ambitious developer as it marks the transition from a newcomer to an integrated member of the KDE project. Hence, it is an important cultural means to foster excellence.

Collective development

In their manifesto the KDE community commits itself to excellence. The outcome of their activity should be no less than the best desktop environment. ***Concerted development*** is one way the community tries to achieve this excellence. Peer reviewing and feedback contributes to this. KDE reuses peer reviewed code, because it prevents the community from reinventing the wheel. However, new ideas develop out of variation and ***continuous experimentation*** (Lanzara & Morner 2005). As a consequence, KDE set up technology and rules which enable both reuse and redundancy. This produces an enormous amount of code as well as discourse for ***collective innovation***, which calls for the reduction of complexity. In the following, we describe how the community solves these contradictions.

Task complexity is especially problematic when a huge number of developers are involved (Brooks 1995). Modularization breaks the whole source code down into manageable parts. Consequently, complexity for the individual is low as members can work on modularized tasks which fit their expertise. Furthermore, specific modules can be used for several applications. The rule of reuse adds to this by forcing contributors to actively search for modules that are peer-reviewed and can be incorporated in a further piece of work. Modularity and reuse combined, thus contribute to both an increase in variety and a reduction in complexity. As this

dynamism is at the core of KDE's collaborative work, corresponding rules are sometimes fervently defended, and violations provoke unfriendly replies.

In F/OSS projects, several developers work on the same application, sometimes even on the same module. Concerted action through common artifacts (Bødker 1997; Bardram 1998) is therefore crucial. The KDE project uses the version control system SVN (Subversion). Continuous coordination systems (van der Hoek 2004) avoid change collisions in the code base, therefore they are of utmost importance for the coordination and development of KDE's codebase. Version control systems help people see at a glance what others are doing and hence improve visibility (van der Hoek 2004). In order to make changes in source code visible, SVN provides a very simple tool with the 'diff' function. The use of different colors enables developers to quickly scan the changes made in the repository. As with almost every other technology used in the community, its complexity-reducing functionality lies in its simplicity. Such work tools not only coordinate work and encourage reuse, they also foster productive redundancy.

Mailing lists and IRC channels provide platforms to solve problems and collectively think about action. Developers reflect on their work and evaluate simple ideas for implementation. KDE is exploiting the fact that mailing lists open the opportunity for self-selection, whereby help is provided by those developers who think they are most likely to have a good answer. The communicative interactions that emerge can involve a variety of different contributors with different expertise. On the Internet it is not important to know *who-knows-what* (Faraj & Sproull 2000); it is sufficient to know *where* to post or to search for. As the addressee is the entire group of participants who are subscribed to a mailing list, helping behavior in mailing lists is extensive.

However, the fact that anybody can talk is both an advantage and a disadvantage. Written communication encourages reflection before someone hits the send button. Even so, core developers also admit that 'social interaction on the mailing lists could be drastically improved'. Still, due to the fact that mailing lists are powerful 'help desks', they have never been changed in their basic functionality. However, the constant influx of new members and an increase in code and subprojects has led to several refinements of the system. Mailing lists split from time to time into new, topic-specific mailing lists in order to keep the amount of discourse on the lists within digestible limits. Furthermore, mailing list digests reduce the overwhelming number of mails received.

Developers can also search mailing list archives. Archived discourse with experts constitutes an invaluable source for productive inquiry when developers look for solutions to their coding problems. Developers primarily use source code, and error messages in particular, to express their problems. Another common way to express technical problems is to present solutions which have already been tried out, in order to make other developers understand the problem. Code is the common language of the audience and is in itself an invaluable mental artifact, because it reflects the mental models of its creators.

Together with the code repository, mailing lists build the project's platform for technical help, but also for reflection, and idea generation. Collective reflection shapes goals and produces collective ideas that go beyond individual thinking. These collectively produced ideas are the source of innovation. Yet, collective reflection on new ideas demands shared understanding of the problem or issue involved, which is difficult to achieve online. Developers use various mechanisms to overcome the drawbacks of physical distance, for instance recapitulating an idea, or elaborating the idea in more depth. They further support ideas, present different perspectives of the problem, point out flaws, insist on their views, disagree, and defend their own ideas in a constant process of idea generation and reviewing.

However, for joint conceptualizations, KDE developers use other mental artifacts such as programming language (e.g.: plain code; "what if, if then" arguments), analogies, and future usage scenarios for collective reflection. One simple variant is to use analogies. Developers 'de-contextualize' their idea from the technical background and put their thoughts in a comprehensible, everyday context. Contrary to 'tech-talk', the language here changes drastically in order to achieve a common understanding of a problem. Another common method of supporting the presentation of an idea is to describe future outcomes or usage scenarios, and collectively develop ideas in a way that describes a 'virtual reality', that is, plays with the future realization of the respective idea.

The use of language as a mental artifact enables developers to collectively create innovative ideas. Discussants collectively work to circumscribe their ideas, thus trying to achieve shared understanding and setting goals for future action. These processes, of course, do not run smoothly. Analysis has shown that during discourse, ideas are constantly contested. In a way, ideas are peer-reviewed. Idea reviewing differs from the peer review of source code because of its emphasis on generating rather than selecting ideas. Using the reuse rule here would be detrimental for idea generation. Instead, for innovative and co-constructive tasks, the community has put a focus on *doing,* which provides the ultimate feedback – either it works, or it doesn't!

Co-constructing the activity system

Work tools are changed or developed when there is tension, be it a technological or a social problem, or when new opportunities arise (Bardram 1998). Discussions on the object of work are extremely rare; however, the growth of the KDE code- and developers base has led to tensions that derive from the usage of work tools, such as the version control system. KDE originally decided to work with the 'Concurrent Versions System' (CVS). Their switch from CVS to SVN was one of the most far-reaching changes in recent years. Restrictions in reorganizing and renaming parts of the source code without losing a file's history, for example, hindered developers from experimenting on more innovative software concepts in parallel. This contradiction triggered discussions among core developers. Decisions with such far-reaching consequences are first discussed face-to-face.

A one-year discussion was held on the core developers' mailing list in order to retrieve opinions from everyone concerned and achieve a shared understanding of the problems involved.

The core developers list is the only list where participation is restricted to developers, but it is publicly accessible. This reduces the perils of an unmanageable amount of discourse and at the same time enables all interested members to gain a shared understanding of the ongoing issues. Analysis of related threads showed that the decision-making process follows a particular path. In a first step, all developers are invited to contribute to the discussion. Extensive information on SVN and other version control systems was collected, evaluated and discussed. Developers considered not only the impact of the new system on their future work but also the consequences of the switch itself.

The rules of decision making are consensus and 'he who does the work decides'. When mutual consent among KDE developers had become apparent, one of the core developers took over the lead and wrote a task list. This was followed by a call for active commitment. Task lists and commitments signal a switch towards action. After decision-making, further discussion is unwanted. The community sets clear rules: democracy and meritocracy, not in a political sense, but as a rule as to how to do things. While democracy manifests itself through open discussions, meritocracy is the way in which those who commit themselves to work are empowered. Although seemingly contradictory, both entail important functions. As with regard to the strategic decision, open discussions increase the knowledge-base and increase acceptance. After finalizing the decision, further discussions distract from implementation, thus meritocracy takes over.

Discussion

It has been argued that the world of work is going through some major transformations as global, decentralized, participatory, creative online organizations are taking shape (Engeström 2006). Our study was intended to enrich the increasing body of research in the field of online collaboration and co-configurative work. It contributes to the existing literature in two important ways. First, we address and extend the recent assertion in co-configurative work research that the overall objective and the goals of activities of the many and diverse actors have to be 'anchored' in order to link coordination and cooperation in complex activity systems. Second, our research contributes to our knowledge of how to design online activity systems that are able to cope with the contradictions inherent in co-configurative work.

A central insight derived from our findings is that online co-configurative work needs what we may call *coat-tailing systems* which tie everyday actions to the overall activity of the group. Due to the fluid boundaries of co-configurative groups, and their dispersed character, coat-tailing is of critical importance. As various actors are constantly contributing in parallel, activity systems have to be designed, which weave the work of many into the web of activities. Coat-tailing systems enable '*doing just one thing together*'. Coat-tailing extends the concept of anchoring as described by Engeström (2005). Contrary to work environments where links between different levels of activities are established with particular single artifacts (Engeström 2006), in dispersed groups such boundary objects are prone to avoidance (Sapsed & Salter 2004). Dispersed groups, rather, develop what Kellog et al. (2006) have called a 'trading zone' by engaging in the practices of displaying, representation, and assembly of work through the

use of Internet technology. Work tools and rules are designed to match individual expectations and to *enable* individual activity, where the collective activity is achieved by 'piggybacking' on the fulfillment of the individual task. Coat-tailing is a subtle effect as individuals are not necessarily aware of it.

Coat-tailing systems are also restrictive and exclusive. First, as technology's mediating capacity is strongly dependent on users' understanding of the properties and functionalities of a technology (Orlikowski 2000), coat-tailing systems need appropriate technology for different users. Second, restrictions are also strongly conveyed in openly displayed discourse, content, rules, and licensing. Openness thus contributes to access restrictions. Contradictory as it may sound, this is exactly the characteristic of coat-tailing systems: to embrace contradictions in order to solve them. This is why coat-tailing systems are not a stand-alone solution, nor should they be thought of as merely technological artifacts. Technology and culture are closely interwoven to support the parallel pursuit of the long-term, collective activity and short-term, individual tasks. In order to resolve contradictions between individual action and the collective activity, combinations of artifacts are applied which flexibly adapt to different situations. Our findings support Engeström's (2006) proposition that co-configurative work tends to form integrated 'toolkits'. As a coherent system they foster mental processes, and enable action. Coat-tailing systems are geared towards making people do *and* think. While browsing through open content, newcomers, for instance, may discover interesting tasks, and engage in productive inquiry.

Our findings add yet another component to online collaboration which is important for collaborative success in dispersed group work. Coat-tailing systems foster and allow for the pursuit of individual goals *and* the strategic objective. However, contrary to Engeström (2006), who presents a hierarchical view of either upwards or downwards 'anchoring', we found that online co-configurative systems are designed so as to enable both, and at the same time. Coat-tailing systems enable parallel processing and attainment of strategic objectives and operative goals. User integration, for instance, takes place simultaneously on the strategic level of cultural integration and on the operative level of immediate task involvement. Furthermore, coat-tailing systems are geared to cope with tensions that arise from the pursuit of both. The tensions *as well as* the solutions to the strategy-task conflict are inherent in coat-tailing systems. Yet, both are important; tensions are important triggers of change, while solutions enable coordinated work. Technological artifacts become vehicles or *enablers* of human agency (Kuutti 1996; Orlikowski 2000). It is not technologically sophisticated work designs but the ability of groups to co-constructively react to tensions in activity systems which contributes to sustainable online collaboration.

Co-configurative work rests on informal relationships, liberal sharing of information, meritocracy, and openness to external members. Literature has portrayed these particularities as paradoxical as these organizations seem to act in a contradictory fashion (Kreiner & Schultz 1993). Yet, co-configurative activity systems are actually only contradictory when viewed from a *coordinative* perspective on the level of individual actions. Viewed from a collective, *cooperative* perspective, the paradoxical nature of contradictory human actions vanishes. By embracing

contradictions, and weaving them into a coat-tailing system of technological, cultural, and mental artifacts, online collaboration can transgress the limitations of the dispersed group work.

References

Bardram, J.E. (1998) *Collaboration, coordination, and computer support: an activity theoretical approach to the design of computer supported cooperative work*. Ph.D. Thesis, Aarhus.

Barthelmess, P. & Anderson, K.M. (2002) A view of software development environments based on activity theory. *Computer Supported Cooperative Work*, 5(3), pp.13–37.

Blackler, F., Norman C. & McDonald, S. (2000) Organization processes in complex activity networks. *Organization* 7(2), pp.277–300.

Bødker, S. (1997) Computers in mediated human activity. *Mind, Culture, and Activity* 4(3), pp.149–158.

Brooks, F.P. (1995) *The mythical man-month: essays on software engineering*. Reading, MA, Addison-Wesley.

Chaiklin, S. & Lave, J. eds. (1993) *Understanding practice – Perspectives on activity and context*. Cambridge, Cambridge University Press.

Charmaz, K. (2006) *Constructing grounded theory – A practical guide through qualitative analysis*. London, Sage.

Cole, M., Engeström Y. & Vasquez, O. eds. (1997) Mind, culture, and activity: seminal papers from the laboratory of comparative human cognition. Cambridge, Cambridge University Press.

Cole, M. (1999) Cultural psychology: some general principles and a concrete example. In: Engeström, Y., Miettinen, R. & Punamäki, R.L. eds. *Perspectives on Activity Theory*. Cambridge, Cambridge University Press, pp.87–106.

Crowston, K., Annabi, H., Howison, J. & Masango, Ch. (2004) *Towards a portfolio of FLOSS project success measures*. ICSE Open Source Workshop.

Demil, B. & Lecocq, X. (2006) Neither market nor hierarchy nor network: the emergence of bazaar governance. *Organization Studies* 27(10), pp.1447–1466.

Engeström, Y. (1987) *Learning by expanding: an activity theoretical approach to development work research*. Helsinki, Orienta Konsultit.

Engeström, Y., Brown, K., Carol, Ch. & Gregory, J. (1997) Coordination, cooperation, and communication in the courts. In: Cole, M., Engeström, Y. & Vasquez O. (eds) *Mind, Culture, and Activity: Seminal Papers from the Laboratory of Comparative Human Cognition*. Cambridge, Cambridge University Press, pp.369–385.

Engeström, Y. (1999) Innovative learning in work teams: analyzing cycles of knowledge creation in practice. In: Engeström, Y., Miettinen, R. & Punamäki, R.L. eds. *Perspectives on Activity Theory*. Cambridge, Cambridge University Press, pp.377–404.

Engeström, Y. (2004) *New forms of learning in co-configuration work*. Presented to the LES Department of Information Systems ICTs in the contemporary world: work management and culture seminar, January 2004.

Engeström, Y. (2005) Activity theory and expansive design. In: Bagnara, S. & Crampton Smith, G. eds. *Theories and Practice in Interaction Design*. Lawrence Erlbaum Associates, pp.3–24.

Engeström, Y. & Blackler, F. (2005) On the life of the object. *Organization* 12(3), pp.307–330.

Engeström, Y. (2006) From well-bounded ethnographies to intervening in mycorrhizae activities. *Organization Studies* 27(12), pp.1783–1793.

Faraj, S.A. & Sproull, L.S. (2000) Coordinating expertise in software development teams. *Management Science*, 46(12), pp.1554–1568.

Feller, J. & Fitzgerald, B. (2001) *Understanding open source software development*. London, Addison-Wesley.

Goulding, Ch. (2002) *Grounded theory – A practical guide for management, business and market researchers*. London, Sage.

Kaptelinin, V. (1996) Computer-mediated activity: functional organs in social and developmental contexts. In: Nardi, B.A. ed. *Context and Consciousness: Activity Theory and Human Computer Interaction*. Cambridge, MIT Press, pp.23–34.

Kreiner, K. & Schultz, M. (1993) Informal collaboration in R & D. The formation of networks across organizations. *Organization Studies*, 14(2), pp.189–209.

Kuk, G. (2006) Strategic interaction and knowledge sharing in the KDE developer mailing list. *Management Science* 52(7), pp.1031–1042.

Kuutti, K. (1996) Activity theory as a potential framework for human-computer interaction research. In: Nardi, B.A. ed. *Context and Consciousness: Activity Theory and Human Computer Interaction*. Cambridge: MIT Press, pp.17–44.

Lanzara, G.F. & Morner, M. (2003) *The knowledge ecology of open-source software projects*. Paper presented at the 19th EGOS Colloquium (July), Copenhagen.

Lanzara, G.F. & Morner, M. (2005) Artifacts rule! How organizing happens in open source software projects. In: Czarniawska, B. & Hernes, T. eds. *Actor Network Theory and Organizing*. Copenhagen, Copenhagen Business School Press, pp.67–90.

Lee, G.K. & Cole, R.E. (2003) The Linux kernel development: an evolutionary model of knowledge creation. *Organization Science*, 14(6), pp.633–649.

Leontiev, A.N. (1978) *Activity, consciousness, and personality*. NJ, Prentice-Hall.

Leontiev, A.N. (1981) The problem of activity in psychology. In: Wertsch, J. ed. *The concept of activity in Soviet psychology*. Armonk, NY, Sharpe.

Nardi, B.A. ed. (1996) *Context and consciousness: activity theory and human computer interaction*. Cambridge, MIT Press.

Orlikowski, W.J. (2000) Using technology and constituting structures: a practice lens for studying technology in organizations. *Organization Science*, 11(4), pp.404–428.

Sapsed, J. & Salter, A. (2004) Postcards from the edge: local communities, global programs and boundary objects. *Organization Studies*, 25(9), pp.1515–1534.

van der Hoek, A., Redmiles, D., Dourish, P., Sarma, A., Filho, R.S. & de Souza, C. (2004) Continuous coordination: a new paradigm for collaborative software engineering tools. In: *Proceedings of the Workshop on WoDISEE*.

Vygotsky, L.S. (1978) *Mind in society*. Cambridge, MA, Harvard University Press.

Wells, G. (2002) The role of dialogue in activity theory. *Mind, Culture, and Activity*, 9(1), pp.43–66.

Yamauchi, Y., Yokozawa, M. Shinohara, T. & Ishida, T. (2000) Collaboration with lean media: how open-source software succeeds, *Proceedings of the 2000 ACM conference on Computer Supported Cooperative Work*, pp.329–338.