LUCI

"Edge effects: complex practices between open source and proprietary software development"

by

Tri Choontanom and M. Six Silberman

Technical Report LUCI-2010-001
http://luci.ics.uci.edu

The Laboratory for Ubiquitous Computing and Interaction

Department of Informatics

Donald Bren School of Information and Computer Sciences

University of California at Irvine

Edge effects: complex practices between open source and proprietary software development

Tri Choontanom
Department of Informatics
University of California, Irvine
tchoonta@uci.edu

M. Six Silberman
Department of Informatics
University of California, Irvine
msilberm@uci.edu

INTRODUCTION

In this section we list our research objectives, develop a basic rationale for the project, and begin to sketch our conceptual framework.

Research objectives

We aim to map software practices which fall outside the canonical categories of open source and proprietary software development and use. We extend Nardi and O'Day's metaphor *information ecologies* [1] to develop a notion of *edge effects* in software practice. In ecology, this phrase denotes the biological diversity that often flourishes at boundaries between zones as a result of the variety of niches that such boundary regions furnish.

In our work, we focus on edge effects between open source and proprietary software development practices and their attendant technical, social, legal, and economic resources, constraints, norms, and strategies. In particular, we identify a handful of particular software projects in which 'open' and 'closed' code coexist, and in each case examine: (1) **technical characteristics** including platform, software language and architecture, code availability; (2) **software use and development patterns** as distributed in space and time among stakeholders; (3) **roles** of, and **relations** between, users and developers; (4) **patterns of interaction** among and between users, developers, and other stakeholders; (5) **technologies and practices** which facilitate and mediate interaction between stakeholders; e.g., code versioning software, documentation, feedback forums, and IRC channels; (6) **legal issues** including software licenses (and their interoperability in large software projects), intellectual property, terms of service, and legal actions; and (7) **economic relations** between stakeholders.

These are not separate facets of analysis. They intertwine, but can serve as entry points into analyses and narratives that foreground different features of software practice.

Rationale

This research is of interest as a handful of accounts of underexamined sociotechnical practices which lie outside traditional categories of lay discourse and academic analysis. We aim to enrich understandings of the interplay between software use and development and the technical, social, political, economic, and cultural aspects of software practices beyond the too-simple categories of open source and proprietary; networked and institutional; distributed and centralized; expert and lay; and so on. We propose to develop a more complex account of software practice which foregrounds the ways these categories are often blurred in the 'real world'. It is intended as a contribution simultaneously to empirical research in software engineering (SE) and computer supported cooperative work (CSCW).

This work may be of particular interest to SE researchers focusing on the relation between software architecture and organizational strategies; on the development and empirical assessment of software processes; and on activities traditionally outside software process models, such as documentation, testing, deployment, and maintenance.

It may also be of interest to CSCW researchers studying software engineering *as* cooperative work, and is especially relevant to ongoing questions about the interplay *between* software development and use as a locus of cooperative work.

We address the relevance of the proposed study to SE and CSCW research in greater detail in subsequent sections. More broadly, we imagine that it can serve as a rich resource to (1) researchers and practitioners in fields drawing on CSCW research, such as collective intelligence, social informatics, information architecture, and virtual, networked, or hybrid organizations; (2) researchers in the emerging transdisciplinary field of game studies; (3) researchers in the sociology and politics of sociotechnical systems, including scholars in science and technology studies and critical informatics; (4) anthropologists of sociotechnical systems, especially those involving software; and, most broadly, (5) designers and users of and in sociotechnical systems, especially those involving software. We do not however address these (inter-)disciplinary conversations explicitly here.

Conceptual framework: technological ecologies

For Nardi and O'Day, an *information ecology* is "a system of people, practices, values, and technologies in a particular local environment." Analytically, "the spotlight is not on technology, but on human activities that are served by the technology" [1] (p. 49). Nardi and O'Day develop five analytical foci suggested by the ecological metaphor:

> An information ecology is a complex *system* of parts and relationships. It exhibits *diversity* and experiences

> continual evolution. Different parts of an ecology *coevolve*, changing together according to the relationships in the system. Several *keystone species* necessary to the survival of the ecology are present. Information ecologies have a sense of *locality*. [1] (pp. 50-51)

We elaborate on these foci in reverse order.

Locality is a troubled characteristic in analyzing distributed work[1] and we propose to treat it not as a fundamental analytical lens but only if and as it arises.

Keystone species is a useful concept, but the idea of a 'species' in a technological ecology can obscure more than it clarifies by covering over differences within ostensibly homogenous categories. When Nardi and O'Day discuss librarians as a keystone species, for example, it becomes more difficult to ask about the ways in which librarianship itself functions *internally* as an ecology, with different librarians playing different roles.

Coevolution within ecologies, *diversity*, and the composition of a discernible whole from interconnected parts signified by the term *system* are all integral to our analysis. Models other than 'information ecology' exist to analyze objects with these features, especially in the literature on the sociology of technology. We do not review these in detail here: the ecological metaphor affords a particular analytical perspective (including the 'edge effects' concept) which we value. We do however think that two models, in particular Edwards' notion of *discourse* [31] (pp. 30-41) and Latham and Sassen's construct *digital formation* [32], enrich Nardi and O'Day's ecological lens in useful ways. Because we are focused not only on information per se, we will expand this lens to develop an analytic of *technological* ecologies.

For Edwards, *discourse*, broader than just "the act of conversation," refers to all "*signifying* or *meaningful practices*: those social interactions—material, institutional, and linguistic—through which reality is interpreted and constructed for us and with which human knowledge is produced and reproduced." Analytically, it allows us to balance traditional questions in the history and sociology of science and technology about the social construction of technology with "their converse," what Edwards calls "*the technological construction of social worlds*." In Edwards' usage, *discourse* is "a broad term...for the heterogeneous *media* in which the processes of social construction operate" [31] (p. 34). By focusing attention on the ways in which, to simplify, 'technology' and 'society' constitute *each other*, rather than holding up as ideal a scenario in which technologies always "serve" human activities, Edwards' notion of discourse affords a richer understanding of what might be called the *dynamics* of technological ecologies.

Latham and Sassen develop a notion they name the *digital formation*. This "construct" builds on the term "social form" from sociology and "is meant to convey that digital formations have ontological status as social 'things' (with coherence and endurance), but not as fixed units whose attributes are pregiven to analysis" [32] (p. 9). Importantly, social logics have weight in digital formations (see e.g., [33]). This provides an analytical entry point for power relations in technological ecologies, a conspicuous absence noted by critics of Nardi and O'Day's approach (e.g., [34]) and which, following Shaikh and Cornford [27] (see below), cannot be ignored if we wish to understand the unfolding of software work.

This expanded metaphor, which we call *technological ecologies*, forms the basis of our approach.

BACKGROUND

In this section, we link our research objectives to broad ongoing discussions and open questions in software engineering and computer supported cooperative work.

Dittrich et al. [2], in their editorial in the 2009 special issue of the journal *Computer Supported Cooperative Work* ("JCSCW") on software engineering as cooperative work, observe that "software engineering research (SE) and research on computer supported cooperative work (CSCW) seem to have had a difficult—even non-existent—relationship at various times," on account primarily of differing methodological and epistemological commitments (pp. 393-394). As stories of large and costly failures in software projects accumulated in the early 1990s, an uneasiness with the traditional idealizations of software process in SE research led to collaboration between qualitative researchers in CSCW and SE researchers (p. 394). CSCW came both to produce "studies *for* software engineers" and to "address the design and development of software as a cooperative work practice itself" (p. 394)—i.e., to produce studies *of* software engineers. There has, they point out, been some consternation over this relation among CSCW researchers, citing in particular well-known papers from Dourish [3] and Crabtree et al. [4].

The reverse is also true: the empirical turn (originally not identical with the 'qualitative turn') in software engineering has been a locus of disciplinary angst for at least a decade. On one hand, significant activity has developed around empirical approaches in software engineering, including an annual ACM workshop[2]; a special issue (in 2007) on qualitative research in the software engineering journal *Information and Software Technology*; an annual IEEE/ACM SIGSOFT symposium[3]; and a journal, *Empirical Software Engineering*, inaugurated in 2006. On the other hand one observes that this activity has included a running conversation indexed

[1] Indeed geographical metaphors generally are troublesome. We acknowledge, for example, that our own metaphor, 'edge effects,' as applied, requires interpretation in a conceptual rather than physical space.

[2] CHASE, the International Workshop on Cooperative and Human Aspects of Software Engineering, colocated with the CSCW conference in 2006 but colocated with ICSE, the International Conference on Software Engineering, in 2008, 2009, and 2010.

[3] ESEM, the International Symposium on Empirical Software Engineering and Management, inaugurated in 2007.

by papers and keynotes with titles like "Is there a future for empirical software engineering?" [5] and "The limits of empirical studies of software engineering" [6]. These authors (and others with less dramatic titles; e.g., [7], [8]) have expressed concern over a lack of methodological consensus in the subfield and have sometimes attempted to constrain empirical, qualitative, or descriptive methods to a role subservient to formal, quantitative, prescriptive strategies.

It can perhaps be said fairly that SE research continues to search for scalable, portable, and generalizable technical solutions to organizational challenges (and to develop software tools and formalize organizational processes which embody these solutions; e.g., [9]) while CSCW research continues to highlight the ways in which new tools, techniques, and formal models present new organizational problems which must be 'worked around' or solved 'in practice' or 'on the ground' by situated actors with deep local knowledge and particular interpersonal connections (e.g., [10]).

Despite these long-running concerns, a variety of factors continue to conspire to drive uptake of CSCW-flavored approaches within SE research, and to inspire CSCW researchers to examine SE practices, problems, and contexts.

Perhaps most widely acknowledged among these is the growing importance of coordination and the growing complexity (and cost) of collaboration in increasingly distributed software development teams. Some of the challenges associated with what is now referred to as 'global software engineering' or 'global software development' ("GSD") were articulated in 2007 by Herbsleb [11], who argued that "the key phenomenon of GSD is coordination over distance" and that the scale of coordination in large GSD projects called for new research trajectories in software architecture, requirements engineering, software tools, and organizational structure. Perhaps appropriately, proceedings from the annual conference on global software engineering (ICGSE, inaugurated 2006) evince great methodological diversity, mixing case studies and other qualitative and empirical approaches with software tools, formal models, and other quantitative and even predictive efforts.

Another aspect of software engineering examined by both SE and CSCW researchers is software deployment, defined by one SE researcher as "a postproduction activity...performed for or by the customer of a piece of software" [12]. As in GSD, SE research has focused on deployment systems (e.g., ibid.), while CSCW has emphasized practices, highlighting ways in which tools can support rather than automate such phenomena as "human infrastructure" [13], configuration and "appropriation work" [14], and "articulation work" [15], [16], [17], and foregrounding occasions in which deployment of systems intended to automate or replace human labor simply changes or moves it (e.g., [18]).

A related subfield of software engineering called software evolution has emerged which examines, models, and develops tools for practitioners involved in the process of changing complex software programs after they have been developed or deployed. The subfield has spawned an annual workshop (IWPSE, the International Workshop on the Principles of Software Evolution) and a working group under ERCIM (the European Research Consortium on Informatics and Mathematics), which also meets annually. The subtitle of the upcoming 2010 special issue of *IEEE Software* on software evolution, "Maintaining Stakeholders' Satisfaction in a Changing World", expresses well the subfield's goals and concerns. These have been pursued with the traditional SE emphases on tool development, theoretical modeling, prediction, control, scalability, and so on [19], although with at least some historical interest in an empirical component to theory development [20].

Three subfields of SE research that might benefit from a CSCW perspective, but seem underexamined thus far in CSCW research proper, are software architecture (but see [10] for a discussion of social structure and work structure in cooperative work generally); legal and economic questions in software development; and free/libre/open source software ("FLOSS") development. In SE, FLOSS projects have inspired a series of generative studies in software evolution, some descriptive and some prescriptive; e.g., [21], [22], [23], [24], and a number of workshops. Much SE research involving FLOSS, however, abstracts from its unique organizational, legal, or economic conditions of development; it only happens to involve FLOSS, for technical reasons, for convenience, or by convention. Similarly, research on FLOSS projects per se is surprisingly sparse in CSCW (but see e.g., [25], [26]). The architectural, organizational, technical, legal, and economic complexities that arise (and are often managed) in FLOSS development offer a unique terrain for CSCW-flavored studies of distributed software engineering practice, and such studies can illuminate the complex interactions between architectural, technical, economic, legal, and organizational norms, objectives, conditions, and constraints in distributed software development more broadly, with relevance for software engineering research and practice (see [27] for a short but provocative example exploring organizational learning in the Linux community). In the present study, we propose to focus on 'boundary' or 'hybrid' development ecologies which mix FLOSS and proprietary code, models (organizational and economic), norms (architectural, technical, and social), and practices broadly in an effort to explore these interactions in their full complexity.

RELATED WORK

We propose a 'CSCW-flavored' study intended as a resource for ongoing discussions in both CSCW and SE. As such, we propose to examine 'technical' considerations like software architecture at greater length than does mainstream CSCW research and, simultaneously, to develop an empirical, descriptive orientation rather than the formal, prescriptive orientation common in mainstream SE. We will pay careful attention however to the ways in which prescriptive understandings of SE and formal representations of software process circulate in our field sites and are used as resources in software work.

In this section, we report in greater detail on a small handful

of papers in SE and CSCW that our work relates to directly.

Herbsleb [11] articulates a "desired future" for global software development (GSD), which he observes is "becoming the norm" despite growing evidence that "global distribution of a project impairs critical coordination mechanisms." He reviews SE research on GSD and articulates research challenges in software architecture, requirements engineering, software tool development for GSD support, and coordination strategies. He offers a useful definition of coordination in software development—management of dependencies among tasks—pointing out that "if tasks carried out at different sites shared no dependencies, global projects would not pose significant challenges." In his view, "the key phenomenon of GSD is coordination over distance" and "the fundamental problem of GSD is that many of the mechanisms that function to coordinate the work in a co-located setting are absent or disrupted in a distributed project."

Summarizing relevant software architecture research, Herbsleb notes a widespread belief that "the extent and nature of task dependencies in development work are a product of dependencies in the software architecture" and that "the structure of a product [i.e., its architecture] tends to resemble the structure of the organization that designed it." That is, software and organization co-structure one another. Software architecture tends to determine coordination requirements, so it has serious organizational implications. Although research suggests that software architects are aware of these implications, there is little guidance in the literature for managing them. Herbsleb highlights three challenges presented by GSD for software architecture research. First, "the complex relationship between software dependencies and task dependencies" is not well understood: decoupling software components, for example, "may or may not decouple tasks", especially if a dependency is "semantic rather than syntactic". Second, no strategies exist to "determine how well equipped a given organization is to carry out the design and implementation of a system with a particular architecture"; strategies to assess this "architectural/organizational fit" are needed. Finally, "tactics" are needed for improving this fit on one side or the other, as appropriate.

These are already significant research challenges; Herbsleb articulates additional challenges of similarly intimidating scope in requirements engineering, tool development, and coordination. We omit a summary of these here, but note that just as the tools of a technically and architecturally engaged CSCW research program seem well-equipped to contribute to Herbsleb's architectural challenges, so too do CSCW approaches seem relevant to the challenges in these other domains.

Whitehead [9] takes a model-oriented (i.e., formal) perspective on collaboration in software engineering, offering the following goals as central drivers of that collaboration: "establish the scope and capabilities of a project"; "drive convergence towards a final architecture and design"; "manage dependencies among activities, artifacts, and organizations"; "reduce dependencies among engineers"; "identify, record and resolve errors"; and "record organizational memory." Whitehead's review of strategies for enabling collaboration focuses on software tools and formal processes. An interesting terminological choice highlights an opportunity for cross-disciplinary dialogue with CSCW research: Whitehead's notion of "infrastructure" is technology-centered, and misses opportunities to leverage knowledge about "human infrastructure" (e.g., [13]).

Dearle [12] explores tasks and tools in software deployment. For Dearle, "deployment is a post-production activity that is performed for or by the customer of a piece of software." It is a "process consisting of a number of inter-related acitivities including" release, configuration, installation, activation, monitoring, deactivation, updating, reconfiguration, adaptation, redeploying, and undeploying. Dearle explores software tools relevant to these tasks. Such tools are important and useful for the skilled deployers who have access to them (literally and conceptually), but of course such software tools add logistical and conceptual complexity to deployment, and must themselves be deployed. This means that in many cases deployment will remain a complex set of tasks that must be performed manually. As such, it is another domain in which SE and CSCW research can interact fruitfully.

Shaikh and Cornford [27] present a nontraditional (even for CSCW) approach to analyzing learning in FLOSS development. Studying discussion, selection, and deployment (in Dearle's [12] broad sense) of version control software in the Linux community, they draw on literature in organizational studies to offer a new perspective on organizational learning in software development. "To learn," they write, citing Weick and Westley [28], "is to disorganize and increase variety. To organize is to forget and reduce variety." The Linux community, they observe, must 'learn' somewhow. But where, when, and how does this learning occur? How does the community reorganize itself in order to take advantage of this learning? To explore these questions they focus not on "the organization as such, the static frameworks or functional divisions," but on "the performative activity of organizing that it engages in." They use the concept of "becoming"—"a process through which an organization exists"—defined as the totality of the "generative duality" of "learning" and "organizing", the processes of increasing and reducing variety. This mutually defined triad of concepts, they argue, help to "account for the complex unfolding" of the decisions made around the use of version control software in the Linux community. The course of events as they recount them (drawing from the Linux Kernel Mailing List) cannot be explained fully by technical decisions alone, or even by the politics flowing from differences of opinion on technical matters. Rather, a descriptive account of the technical decisions that were made involves a complex interleaving of logistical problems, questions over legal protections and licensing issues, security concerns, interpersonal politics, ideological differences of various flavors, and conflicting opinions about the relative merits of "stability" and "constant improvisation" in FLOSS development. In highlighting this diversity of inputs to the decision making process, Shaikh

and Cornford underscore the need for a broadly descriptive approach to understanding software process, troubling the claim that an understanding of software process must begin with formal models, with empirical research functioning only to verify or repudiate the accuracy of these idealizations (e.g., [6]). If outputs of empirical research on software process were limited to an affirmative or negative answer to the question "Does this formal representation of software process describe what is seen 'in the wild'?" they might be of limited utility: they would likely tell us "no" every time.

Cohn et al. [29] develop a more fruitful approach to understanding the relation between formal representations of software process and empirical work, and present a novel conceptualization of the role of formal process models in software practice. They note that "software process can be viewed as comprising both models and enactments which work together to form a *generative system*. Software process as a generative system is made up of patterns of action which people can recognize (models) as well as specific performances of work (enactments). Models and enactments mutually constitute the software process which is produced and reproduced through the interplay between the two." We adopt this view of software process as neither determined completely by process models nor entirely free from their influence. As in Suchman's [30] now-canonical account of the relationship between plans and situated actions, software process models function as resources for enactments of work; these enactments in turn serve as resources in the development of new models and the interpretation and deployment of existing models.

METHODS AND FIELD SITES

We propose mixed-methods studies of two phenomena in particular massively distributed technological ecologies: modding in World of Warcraft and Facebook application development. Differing technical characteristics and drivers of collaboration in these ecologies will facilitate a comparative analysis.

Modding in World of Warcraft (WoW) involves using the Lua programming language and XML, the eXtensible Markup Language, to create modifications ("mods"). Mods are integrated into the game interface by the WoW software itself, a desktop application which provides an application programming interface (API) for mod authors ("modders"). *Facebook application ("app") development*, in contrast, involves web application development, often in PHP, JavaScript, Ruby on Rails, Python, and/or Flash. Facebook itself is a web application and also provides an API for app developers. WoW and Facebook are hybrid technological ecologies where free/open source and proprietary code circulate and interact, driving developers, users, and other stakeholders to develop practices to negotiate the complexities that this hybridity presents.

We propose to engage in ethnography and participant observation to collect qualitative and quantitative data relevant to developing a rich understanding of these complexities and the ways in which stakeholders manage them. The technical, logistical, legal, and economic characteristics of these practices shape our research design: because WoW mods cannot be developed for profit, much mod development coordination happens in public (if sometimes obscure) electronic fora. This allows us to study WoW modding *in general*. We propose to collect and analyze discussions in modding fora and in chat rooms where modders meet for real-time conversation. All code for WoW mods must legally be publicly available; we propose to use this code to ground our architectural and broadly technical inquiries. In addition to close inspection of the code, a review of code comments and architectural analysis can furnish insights on collaboration patterns and motivations. This rich kaleidoscopic approach to data collection allows us to answer quantitative questions about patterns and trends in modding collaboration—do most modders work alone? collaborate serially on mods? collaborate by participating in peer review? and so on—in addition to developing nuanced descriptive accounts.

The technological ecology around Facebook, by contrast, supports both informationally and organizationally closed companies which exist to profitably develop and maintain apps and individuals and informal groups who develop apps for nonfinancial reasons. Like WoW, Facebook itself is a closed codebase, but the API provided for developers calls a remote application, not a desktop client. Thus Facebook apps run remotely and 'install' to a user's *profile* rather than their local machine; Facebook app users may be exposing their personal data to malicious code, but their *machines* are not at any greater security risk than in ordinary web usage. This architectural difference removes the security argument for making code public that in part animates this regulation in the WoW modding community. The absence of this security risk this plays a role in creating space for proprietary, for-profit software development in the Facebook ecology.

We propose an ethnographic study (and, if possible, participant observation in the form of an internship) of a financially successful New York-based Facebook app development company, Social Fu. Social Fu produces proprietary code, and its business models rely on proprietary analytical tools, even while the business motivation for their clients and the technical possibility for their work is bound up intimately both with the 'open platform' and the many free/open source software products that they use both *in* their code and *to make* that code. How do the different conditions around Facebook as a technological, organizational, legal, and economic platform shape software practice in its technological ecology? What tensions emerge in the course of developing proprietary code embodying proprietary techniques for profit for deployment in interaction on an 'open platform' which may simultaneously host nonprofit apps with public codebases? How are 'viral' free software licenses managed? How does this affect what can be built (and when) and how (and for whom) it can deployed? How are collaborative practices among Facebook app developers different to those among WoW modders, and how do these collaborative practices reflect or work around the technical, organizational, legal, and economic conditions in their respective ecologies? How do the infrastructures that develop for facilitating that collaboration come to reflect these conditions? Careful at-

tention to the design and use of physical space and the organization of development activity and coordination in space and time can yield findings about the ways in which 'ecological' constraints come to shape particular practices. Participant observation in development at Social Fu may provide us viewing access to some code, but it is unlikely that we will be able to publish any of their code as part of our analysis. We intend to supplement this field study with a content analysis of relevant documents and technical artifacts, including end-user license agreements for Facebook app developers and users, API documentation, and so on, and to include forum and chat room participation within the scope of our ethnographic study to the extent Social Fu developers integrate them in their own software work.

EXPECTED OUTCOMES

The study will contribute to the theoretical toolkit in CSCW by refining and extending Nardi and O'Day's widely-cited ecological metaphor and developing the related notion of *edge effects* in technological ecologies. It will contribute to empirical SE research by providing a detailed account of the ways in which architectural and other technical characteristics; use and development patterns; roles of and relations between users and developers; patterns of interactions among stakeholders; technologies and practices which mediate and facilitate collaboration and coordination; and legal and economic constraints and relations all interact to explain outcomes of software work. This detailed portrait can serve as a resource to prescriptive or formal SE research by foregrounding underexamined variables and workflows in formal and generalizable representations of software process. Incorporation of these field data into process models can help make process models both more general (explaining more patterns of software work) and more precise (explaining work patterns better that they explain at all), and may help to highlight practices which are not yet adequately represented in prescriptive software process research.

Additionally, the large technological ecologies around WoW modding and Facebook app development suggests an alternative perspective on global software development (GSD). Herbsleb [11] discusses GSD primarily as a process characterized by physically separate developers working on a single well-defined project, but the concept can be abstracted to describe many geographically distributed user-developers working simultaneously on multiple projects which enrich use of a core platform or software product. The study thus can make both theoretical and practical contributions to GSD research: an ecological view of GSD can offer analyses of software work not just as they take place within a single geographically distributed organization (and/or its contractors) but within a broader ecosystem of organizations, technologies, and practices. This view provides the analytical leverage required to answer questions like, 'When is it time to build an API?' By examining coordination between developers within a core organization and users with programming capability, the study will furnish new strategies for GSD which maximize efficient use of core developer time and creative flexibility for user-programmers. These patterns of 'emergent coordination' may also suggest strategies for leveraging coevolution of organization and software architecture, another challenge Herbsleb presents for GSD research. In large technological ecologies, APIs bound the space of user-programmer creativity and set the terms on which user-programmed software interacts with (or in some cases is incorporated into) the core software. The API as a point of leverage is underexamined in GSD research; more generally, our study can seed rich multidisciplinary inquiry into the technical, architectural, organizational, social, legal, economic, and political constraints on and implications of API design.

REFERENCES

1. Nardi, B. A. and V. L. O'Day, 1999. *Information Ecologies: Using Technology With Heart*. MIT Press.

2. Dittrich, Y., D. W. Randall, and J. Singer, 2009. Software engineering as cooperative work: editorial. *Computer Supported Cooperative Work* **18**(5-6): 393-399.

3. Dourish, P., 2006. Implications for design. *Proc. CHI '06*: 541-550.

4. Crabtree, A., T. Rodden, P. Tolmie, and G. Button, 2009. Ethnography considered harmful. *Proc. CHI '09*: 879-888.

5. Basili, V. R., 2006. Is there a future for empirical software engineering? Keynote, *ISESE '06*, Sept. 21-22, Rio de Janeiro.

6. Parnas, D. L., 2003. The limits of empirical studies of software engineering. *Proc. ISESE '03*: 2.

7. Kitchenham, B., H. Al-Khilidar, M. A. Babar, M. Berry, K. Cox, J. Keung, F. Kurniawati, M. Staples, H. Zhang, and L. Zhu, 2006. Evaluating guidelines for empirical software engineering studies. *Proc. ISESE '06*: 38-47.

8. Sjoberg, D. I. K., T. Dyba, and M. Jorgensen, 2007. The future of empirical methods in software engineering research. *Proc. ICSE '07: Future of Software Engineering*: 358-378.

9. Whitehead, J., 2007. Collaboration in software engineering: a roadmap. *Proc. ICSE '07: Future of Software Engineering*: 214-225.

10. Hinds, P. and C. McGrath, 2006. Structures that work: social structure, work structure and coordination ease in geographically distributed teams. *Proc. CSCW '06*: 343-352.

11. Herbsleb, J. D., 2007. Global software engineering: the future of socio-technical coordination. *Proc. ICSE '07: Future of Software Engineering*: 188-198.

12. Dearle, A., 2007. Software deployment, past, present, and future. *Proc. ICSE '07: Future of Software Engineering*: 269-284.

13. Lee, C. P., P. Dourish, and G. Mark, 2006. The human infrastructure of cyberinfrastructure. *Proc. CSCW '06*: 483-492.

14. Balka, E. and I. Wagner, 2006. Making things work: dimensions of configurability as appropriation work. *Proc. CSCW '06*: 229-238.

15. Crabtree, A., J. O'Neill, P. Tolmie, S. Castellani, T. Colombino, and A. Grasso, 2006. The practical indispensability of articulation work to immediate and remote help-giving. *Proc. CSCW '06*: 219-228.

16. Simone, C., G. Mark, and D. Giubbilei, 1999. Interoperability as a means of articulation work. *Proc. WACC '99*: 39-48.

17. Schmidt, K. and L. Bannon, 1992. Taking CSCW seriously: supporting articulation work. *Computer Supported Cooperative Work* **1**(1): 7-40.

18. Munkvold, G., G. Ellingsen, and H. Koksvik, 2006. Formalizing workreallocating redundancy. *Proc. CSCW '06*: 59-68.

19. Mens, T., 2009. The ECRIM Working Group on Software Evolution: the past and the future. *Proc. IWPSE-EVOL '09*: 1-4.

20. Lehman, M. M. and J. F. Ramil, 2001. An approach to a theory of software evolution. *Proc. IWPSE '01*: 70-74.

21. Nakakoji, K., Y. Yamamoto, Y. Nishinaka, K. Kishida, and Y. Ye, 2002. Evolution patterns of open-source software systems and communities. *Proc. IWPSE '02*: 76-85.

22. Capiluppi, A., J. M. González-Barahona, I. Herraiz, and G. Robles, 2007. Adapting the "staged model for software evolution" to free/libre/open source software. *Proc. IWPSE '07*: 79-82.

23. Capiluppi, A., J. Fernandez-Ramil, J. Higman, H. C. Sharp, and N. Smith, 2007. An empirical study of the evolution of an Agile-developed software system. *Proc. ICSE '07*: 511-518.

24. Wang, Y., D. Guo, and H. Shi, 2007. Measuring the evolution of open soure software systems with their communities. *ACM SIGSOFT Software Engineering Notes* **32**(6). Article No. 7. 7 pp.

25. Duchenaut, N., 2005. Socialization in an open source software community: a socio-technical analysis. *Computer Supported Cooperative Work* **14**(4): 323-368.

26. Sack, W., F. Détienne, N. Duchenaut, J.-M. Burkhardt, D. Mahendran, and F. Barcellini, 2006. A methodological framework for socio-cognitive analyses of collaborative design of open source software. *Computer Supported Cooperative Work* **15**(2-3): 229-250.

27. Shaikh, M. and T. Cornford, 2005. Learning/organizing in Linux: a study of the 'spaces in between'. *Proc. WOSSE '05*.

28. Weick, K. E. and F. Westley, 1996. Organizational learning: affirming an oxymoron. In Clegg et al., eds., *Handbook of Organization Studies*: 440-458. Sage.

29. Cohn, M. L., S. E. Sim, and C. P. Lee, 2009. What counts as software process? Negotiating the boundary of software work through artifacts and conversation. *Computer Supported Cooperative Work* **18**(5-6): 401-443.

30. Suchman, L. A., 1987. *Plans and Situated Actions: The Problem of Human-Machine Communication*. Cambridge.

31. Edwards, P. N., 1995. *The Closed World: Computers and the Politics of Discourse in Cold War America*. MIT Press.

32. Latham, R. and S. Sassen, 2005. Digital formations: constructing an object of study. In Latham and Sassen, eds., *Digital Formations: IT and New Architectures in the Global Realm*: 1-33. Princeton University Press.

33. Sassen, S. Electronic markets and activist networks: the weight of social logics in digital formations. In Latham and Sassen, eds., *Digital Formations: IT and New Architectures in the Global Realm*: 54-88. Princeton University Press.

34. Bowker, G. C., 2001. Book review, Nardi and O'Day, *Information Ecologies*. *Computer Supported Cooperative Work* **10**(1): 143-145.