Crowdsourcing Software Requirements and Development: A Mechanism-based Exploration of 'Opensourcing'

Completed Research Paper

Damrongsak Naparat
Australian School of Business
University of New South Wales
d.naparat@unsw.edu.au

Patrick Finnegan
Australian School of Business
University of New South Wales
p.finnegan@unsw.edu.au

ABSTRACT

Many commercial software firms rely on open source software (OSS) communities as a source of innovation and skilled labor. One specific form of interaction with OSS communities, termed 'opensourcing', involves firms collaborating with an OSS community by 'crowdsourcing' software production. However, beyond the existence of the phenomenon, little is known about how opensourcing, as a model of software production, works. The objective of this study is to explore opensourcing arrangements in a vertical software domain with a view to delineating enabling mechanisms that explain how firms can collaborate with communities to crowdsource the production of software. Using an in-depth case study of the production of hospital software in Thailand, this study explores how opensourcing is used to determine requirements, identify bugs, and provide user-to-user support in addition to the more traditional approach of crowdsourcing software code. The analysis reveals the operation of six high-level mechanisms (motivation, coordination, effective communication, filtering, integration, and nurturing) and reveals how they operate in conjunction with each other to facilitate opensourcing.

Keywords

Opensourcing, crowdsourcing, mechanism-based theorizing, vertical domain software, software production.

INTRODUCTION

Opensourcing is a specific form of software development that employs crowdsourcing as part of the software production process. In crowdsourcing *"…a company or institution taking a function once performed by employees and outsourcing it to an undefined (and generally large) network of people in the form of an open call. This can take the form of peer-production (when the job is performed collaboratively)…"* (Howe, 2006). Following the crowdsourcing approach, '*opensourcing*' refers to situations where *"commercial companies and open source communities collaborate on development of software of commercial interest to the company"* (Ågerfalk and Fitzgerald, 2008). Commercial firms have practiced crowdsourcing for software production for three decades. Beginning in 1990, Netscape attempted to outsource its browser development project, Mozilla, to the OSS community. However, the outsourced project failed completely and required Netscape to reconsider its action (Cusumano, 1999). Later on, commercial firms, such as Apple and IBM, employed a similar practice (cf. West, 2003).

Extant OSS literature has been dominated by the study of horizontal domain software (e.g. a web browser that can be used across all sectors). Nonetheless, Fitzgerald (2006) proposes that as the OSS development model evolves towards a more "business-friendly footing", firms will seek to leverage OSS communities for the development of vertical software (i.e. software specific to firms in one sector). This is challenging as business processes and user requirements can be complex; suggesting that opensourcing may not be as successful for vertical software as for the horizontal software. Opensourcing, therefore, can be problematic if there isn't a good understanding of business processes that the software needs to support, or the business process is too complex for firms to effectively communicate to a community of developers. It is not surprising therefore that while, by definition, opensourcing involves the whole process of software production, research to date has focused on acquiring software code.

This paper seeks to advance understanding of opensourcing in a vertical software domain, where the use of an external community of users is necessary to determine software requirements. The next section examines opensourcing as a model of software production and proposes that the challenges of opensourcing can be better understood by exploring the process in the context of the mechanisms that facilitate opensourcing. This is followed by a description of the case study research design. The findings are presented in the form of eight propositions that explain how various mechanisms work separately and in unison to facilitate opensourcing.

UNPACKING THE CROWDSOURCING OF SOFTWARE

Traditional models of production through firms and markets (cf. Coase, 1937; Williamson, 1975) have been applied to software production. Although hybrid forms of production operate, software production typically takes place within a firm or through price-based contracting in a market situation. The governance of software production thus typically relies on formal management control (within firms) or contracts (in markets). In contrast, OSS production is known for a lack of formal governance mechanisms, and has led to the identification of a new form of production called commons-based peer production (cf. Benkler, 2002). Although an OSS community may have structure and show signs of hierarchy, it is generally governed by informal social mechanisms, such as norms and reputation (Sagers, 2004). Opensourcing, thus, presents an interesting governance dilemma; while firms can have full control over its employees through management arrangements, it cannot assume the same control in crowdsourcing the production of software (Dahlander and Wallin, 2006). This dilemma presents several challenges for opensourcing arrangements.

First is the under-provision of contributions. The extant OSS literature has shown that there are only a handful of OSS projects that are successful over time. Failure is frequently the result of either failing to secure contributions from participants, or not attracting sufficient participants (Dahlander 2007). The fact that IBM's Eclipse project and Netscape's Mozilla project failed to attract external developers (Cusumano, 1999; Fitzgerald, 2006) is significant in explaining the failure of these opensourcing arrangements.

Second is the coordination challenge. The extant OSS literature reveals that the OSS community defines tasks in accordance with OSS ideology and through community consensus. Participants are self-selected, assigned, or referred to perform tasks, with self-selection being common (Crowston et al., 2007). Moreover, the OSS community has a structure around different types of contributors, e.g. passive users, active users, developers, and release coordinators (Crowston et al., 2006). Autocratic and democratic approaches to decision making are both found (German, 2003). However, social mechanisms play important roles in governing participants' actions (Sagers, 2004). Thus, coordination in OSS communities runs contrary to firm-based governance.

Third, preserving a symbiotic relationship between a firm and the OSS community is difficult. Dahlander and Magnusson, (2005) propose three typologies of firm-community relationships: parasitic, communalistic, and symbiotic. They posit that a symbiotic relationship, in which the firm and community act as 'good members' of the ecosystem, is required to preserve collaboration. Nonetheless, literature has shown that participants, either firm or community, can behave in a way that undermines the relationship, leading to community collapse. Failure to deliver promised code, code misappropriation, free riding, low firm involvement, or attempts by firms to influence the community in their own interest (Benkler, 2002; Cusumano, 1999; De Laat, 2004; Shah, 2006) have all proven detrimental to opensourcing arrangements.

Finally, there is difficulty with specifying requirements. The use of opensourcing to acquire software requirements is attractive when the knowledge of business processes is dispersed through a community. However, the complexity of business processes and user requirements in vertical software poses a challenge to opensourcing, as it requires firms to accurately specify and communicate requirements to developers to produce software. Literature has suggested that poor requirement specification and communication severely undermine the success of the software project (Reel 1999). This is because poor specification of requirements leads to developing the wrong software, and results in the need for extensive modifications at a later stage (Sumner 1999). Moreover, it is the biggest cause of software project failure (Hofmann and Lehner, 2001).

Opensourcing can be conceptualized as firms and an OSS community confronting a software production problem that requires them to act collectively and exchange the necessary resources to produce software. This points towards considering opensourcing as a process; thereby necessitating the consideration of the underlying mechanisms (i.e. "*the nuts and bolts processes by which cause and effect relationships in the social world come about*") (Gross 2009, p. 368). In order to understand 'how' opensourcing works, we employ mechanism-based theorizing. Instead of identifying relationships between variables, mechanism-based theorizing is explanatory, and emphasizes 'how' and 'why' things work (Davis and Marquis 2005); thereby seeking to explain how a process or a structure S converts input I into outcome O (Hedström and Swedberg 1998). This allows us to explore deeper to see the 'cogs' and 'wheels' that work in conjunction with each other to enable opensourcing to work. To follow Gross (2009), a mechanism underpinning opensourcing, can be unpacked as an aggregation or a sequence of actors (A), problem situations (P), habits (H) along with other resources, and responses (R), or an A-P-H-R chain. For example, participants, firms, or OSS communities are actors. Opensourcing to produce software is a problem situation that actors need to solve. Habits (e.g., desire and belief; cf. Hedstörm and Swedgerg 1998) are evidenced as motivations and ideology that drive actors to act accordingly to their desires. To solve the software production problem, actors respond by performing collective actions or activities relate to software development (e.g. user requirements gathering, and coding), using their resources (e.g., knowledge, and programming skill).

RESEARCH DESIGN

The objective of this study is to explore opensourcing arrangements in a vertical software domain with a view to delineating enabling mechanisms that explain how firms and communities can collaborate to produce software. The study is exploratory and we thus employ a single case study to gain deep understanding of the opensourcing phenomenon (c.f. Yin, 2003).

We selected the HOS community (described in the next section) as a case study, with community participants as the embedded unit of analysis. First, we conducted 30 interviews with three groups of participants: (i) the project founder and CEO, (ii) eight OST employees who participate in the project, and (iii) 20 volunteers from hospitals. Each interview lasted 60 – 90 minutes. Interviews were conducted face-to-face and via videoconference. Second, we examined online discussions over a six-month period amongst participants on the community's online forum and Facebook. Third, we conducted direct observations by participating in a four-day training event, two leisure activities, as well as several formal meetings and informal dinners with community participants. Finally, project documents and code were examined to identify mechanisms that enable opensourcing.

The interviews were transcribed, yielding more than 400 pages of text. Other data (document project, conversations, and code) were downloaded and analyzed. We employed coding techniques proposed by Corbin and Strauss (2008) to analyze the data. The main ideas and concepts were determined using open and axial coding; thereby revealing construct categories and sub categories. Mechanisms were identified from constructs by using the A-P-H-R chain (c.f. Gross 2009) to reveal the operation and use of these constructs identified during open coding. For the purpose of presenting the analysis, the operation of the mechanisms is presented as theoretical propositions.

THE CASE STUDY AND ANALYSIS

The Case Study Background

The Hospital Operating System (HOS) started in 1999 as a government funded project. The objective was to produce a hospital information system, namely HOS, which would help district hospitals to increase their internal process efficiency through the better use of information. This initiative corresponded with the introduction of the 'universal healthcare coverage' scheme by the Thai government. The government wanted to improve the quality of national healthcare but were under financial constraints. In 2000, the HOS founder established Open Source Technology Co., Ltd. (OST) and turned HOS in to a commercial venture, with HOS released as open source under the GNU Public License (GPL). The HOS project employed the community model for its software production through the HOS community. OST initiated this community and was highly involved in it by, for example, providing developers, and hosting the online discussion forum. Whilst participants mostly use rather directly develop the software, the community/company relationship does not resemble the 'arms length' user model characterized by Fitzgerald and Ågerfalk (2005) as the Berkeley Conundrum. Specifically, there are distinctive features. First, participants were highly involved in software production through providing business process insights, suggesting change requests, coding, reporting bugs, and providing user-to-user support. Second, by closely collaborating to develop HOS, the HOS community participants established strong ties to each other both online and offline, not just an arm's length relationship between firm and community.

Analysis

Following the mechanism-based theorizing approach the analysis reveals that opensourcing involves a chain of six mechanisms that allows the community to attract participants, gather user requirements, communicate requirements effectively, and enrich participants. The motivation mechanism focuses on creating the desire for participation. The coordination mechanism forms the actions of participants to contribute. The filtering mechanism plays the crucial role of filtering out contributions that undermine opensourcing. In addition, the integration mechanism ensures that meaningful contributions get integrated to a final product. Finally the nurturing mechanism facilitates the relationship building and enriches participants. The operation of the mechanisms is expressed through eight theoretical propositions.

The Motivation Mechanism

We found that the motivation mechanism is crucial to create the desire for participants to take part in opensourcing. We identified a broad range of desires including: utilitarian motivations, friendship, fun and challenge, altruism, as well as reciprocity norms and shared belief.

Utilitarian motivation involves participants' beliefs about receiving material rewards for their participation. OST expected to overcome its deficits in financial and human resources by leveraging the community of practice in public healthcare service to collect domain expertise and user requirements. The founder revealed: *"We started with 4 employees so we could not* *gather user requirements from over 700 district hospitals around the country… [so] we invited them to join our community to share their ideas."* On the other hand, participants wish to use high quality software and overcome problems with day-to-day software use. According to one participant; *"they [OST] know how to develop software, but they don't know how we [hospitals] do our jobs. So we need to tell them"*.

Friendship, as the desire of participants to keep a long-term positive relationship with other participants, encourages participants to take part in opensourcing as a way of helping others. Informants believed that helping friends is a good way to build and maintain friendship. Furthermore, participants in the HOS community desire to have fun and challenge themselves by tackling technical problems. Moreover, the study found 'altruism' to play a role in driving participation. According to one community member *"if I help to improve the software [by giving user requirements and reporting bugs], certainly the whole community [other district hospitals] or even the whole country will gain the benefit."*

Finally, the norm of reciprocity and shared belief in OSS, result in participants contributing to the community. Participants believe that they are obliged to repay favors that they received from other participants. One informant stated *"it is important that there is 'give' and 'take'… if you have received, then you have to give back or otherwise the community would collapse"* Moreover, participants share a belief in mutual learning and sharing. The community's vision and mission reflect this well. The community's ultimate goal of *"A Sustainable Learning Community"* reveals that continuous mutual learning is important to the community. The community's motto, *"One Knows, Everyone Knows",* reflects that sharing is an obligation. Participants are therefore motivated to reciprocate. We thus present our first proposition:

Proposition 1: The motivation mechanism facilitates opensourcing by instilling participant desire to take part in the opensourcing arrangement.

The Coordination Mechanism

The coordination mechanism enables the community to define necessary tasks, allocate participants to tasks, and facilitate the transformations of motivations into contributions. The analysis reveals that the HOS community employs hybrid governance, somewhere between 'hierarchy' and 'self-assignment' typical of commons-based peer production, to define tasks and allocate participants to tasks. OST assumes a leading role in designing tasks, and allocating its staff to major positions related to system analysis, design, and code development. Volunteers are self-assigned to perform tasks based on their expertise. However, volunteers seem to be short of coding skills. Nevertheless, as domain experts, skillful practitioners, and software users, volunteers provide user requirements, report software bugs, perform pilot tests, and offer user-to-user support.

The HOS community uses "pull" and "push" strategies to solicit contributions. The push strategy suits end-users who actively contribute to the community. Active end-users submit contributions to their local IT staff, called 'Admins'. A pull strategy involves actively soliciting contributions from participants. This strategy allows the community to reach more passive participants and end-users, who typically do not contribute unless requested. OST's staff and Hospital-based IT staff (Admins) perform active solicitation either by personally contacting end-users or organizing regular meetings. The community uses consensus to describe complex business processes and requirements. Consensus requires iterations of discussion about a business process until every participant agrees with how it has been described. However, only selected participants, who are known to the community as experts on the issue, participate in these discussions. We now present our second proposition:

Proposition 2: The coordination mechanism facilitates opensourcing by enabling the community to 1) identify necessary tasks and to assign participants to perform tasks through hybrid governance, 2) crowdsource contributions from both passive and active participants and end-users, and 3) better specify complex business processes and user requirements.

The Effective Communication Mechanism

The effective communication mechanism is crucial for the HOS community because it enables participants to specify and communicate complex business processes, user requirements, and bugs accurately. The HOS community achieves effective communication through the use of prototypes, dialogue, narratives, screenshots, error messages and system logs, as well as technical diagrams.

The HOS community employs a process prototype as a collaboration tool to brainstorm complex business processes. The prototype is a dummy model depicting a complex business processes in a hospital in a simple way; making it easier for non-technical participants to understand. It is thus a starting point for participants to discuss, modify, extend, and correct. The HOS founder mentioned, *"It would be very difficult for participants to start from scratch. Therefore I hand them a dummy model that they can expand on"*. A software prototype also serves as a proof of concept for new software functions that can be tested by several pilot sites before being improved based on participant feedback.

To communicate user requirements and business processes, participants use a narrative that contains a list of user requirements with descriptive explanations. Participants also use screenshots of the HOS interface to increase the accuracy of the narrative. Communicating bugs and errors are done differently. Participants use error messages from the software, error logs, and screenshots of errors to increase the accuracy of communication. *"I prefer screenshots to a narrative because other participants can quickly understand the problem and it eliminates ambiguity",* reported one informant. Communication amongst OST staff and community participants with IT education/background is made more effective through the use of technical documents and diagrams. The use of standard diagrams and documents, such as a data dictionary, flow charts, and Unified Modeling Language (UML) enables participants to understand user requirements, logical software design, and data structure. Nonetheless, to improve communication such participants sometimes require a further 'dialogue' to reach a common understanding. This dialogue is a conversion driven by a series of questions and answers designed to help reach a common understanding. We now present our third proposition:

Proposition 3: The effective communication mechanism facilitates coordination by 1) increasing accuracy and clarity in specifying complex business processes and user requirements, hence 2) lowering the cost of coordination because accurate and clear communication decreases the number of transactions required amongst participants.

The Filtering Mechanism

The filtering mechanism is essential to facilitate opensourcing as it filters out unusable contributions that could undermine the community. The HOS community uses a peer review mechanism to verify the accuracy of business processes, and voting mechanisms to filter unnecessary user-requirements.

The peer review mechanism allows HOS participants to verify the quality of contributions. Broad participation ensures diverse knowledge that enables a contribution to be peer-reviewed well. User requirements and business processes submitted by community members are always subject to review and correction. This assures that user requirements are of high quality and accurate. The OST staff admitted *"We are not the end-user so we are not 100% sure [about a business process]…they [volunteers] have to review [the business process] and send feedback to us".*

The HOS community employs a democratic method of harnessing broad participation to filter out unimportant user requirements. Due to the enormous number of user requirements submitted by participants, the HOS community gets participants to vote on which requirements should be incorporated in future releases of the software, and which ones should be ignored completely. According to OST staff, *"any proposed requirement that receives more than 50 votes will get implemented".* This approach allows the community to direct effort to work with more meaningful outcomes. We now present our fourth proposition:

Proposition 4: The filtering mechanism facilitates opensourcing by 1) eliminating unusable contributions and actions that potentially undermine the community, 2) assuring the quality of contributions, and 3) lowering the cost of coordination as participants focus on meaningful contributions.

The Integration Mechanism

New HOS releases are important to the HOS community. The community requires a way to integrate user requirements and convert them into software in a timely fashion in order to respond to rapidly changing requirements in the healthcare service industry. The integration mechanism includes modularity and granularity. The analysis reveals that the logical design of HOS is broken down into smaller modules that correspond to business functions in a hospital (e.g., a dental clinic, and the outward patient). This enables participants to work with the module that they are knowledgeable about. This also means that participants only require knowledge of some parts of the software/process in order to contribute. It also makes it easier for them to specify what they want as they know their own routines well. Through the use of the process prototype, discussed previously, the community can collectively gather knowledge modules from different departments in a hospital before assembling them to form a complete process for a hospital. We now present our fifth proposition:

Proposition 5: Modularity and granularity facilitate opensourcing by 1) enabling the concurrent gathering of expertise from dispersed locations, 2) enabling concurrent software design and development, and 3) minimizing participant's effort by requiring only knowledge in their own business domain.

Nurturing Mechanisms

Nurturing mechanisms consist of mechanisms that allow the HOS community to attract participants, build positive relationships amongst participants, and enrich participants' resources. The study revealed that at the early stage of community building, OST got practitioners from hospitals involved by conducting a pro-active marketing strategy whereby they sent out 700 letters inviting practitioners to take part in the HOS development. The focus was on gathering domain expertise, i.e.

healthcare service knowledge. A significant number of practitioners joined in, providing a critical mass to form the community. At later stages, word of mouth played an important role in attracting new participants as current or former participants spread their good experience with using HOS and participating in the community. Furthermore, the media exposure that HOS received by winning several national software awards highlights the quality of the software. This helps attract new participants and helps sustain the commitment of existing participants by emphasizing the value of participants' contributions. We, thus, present our sixth proposition:

Proposition 6: The "marketing the community" mechanism facilitates opensourcing by 1) building awareness for the industry that the community is active and produces quality software, hence 2) attracting in new participants to the community, and 3) maintaining current participants.

Educating the community involves using various methods to enhance participants' technical knowledge. The company, targeting different groups of participants, regularly organizes training sessions. Fundamental courses focus on the use of the software (HOS) and maintaining infrastructure software (e.g. operating systems and databases). Advanced courses deal with SQL, open source reporting tools, JAVA programming, and HOS module development. The goal is to transfer necessary knowledge to participants so that they can contribute more to the community. Community-led mutual learning was also evident with participants learning by discussing various topics (e.g. SQL coding, trouble shooting, and system maintenance) with other participants. A user guide is another source of education. User guides and software documentation are freely available from the community website, as well as from OST and other participants. Finally, 'demonstrating' is an effective mechanism to transfer knowledge between participants. This involves a participant demonstrating how to overcome a particular problem (i.e. trouble shooting) to other participant. This can be done either using online communication tools (e.g. Skype and remote connection tool) or face-to-face. We now present our seventh proposition:

Proposition 7: Training facilitates opensourcing by transferring necessary technical knowledge from the company to the community.

The analysis reveals that informal face-to-face meetings provide a relaxing environment for building relationships amongst participants. The HOS community frequently organizes leisure activities after formal meeting and training sessions. An activity such as playing soccer encourages participants to mix and work as a team, while also having fun. Many stay at the same hotel, thus helping participants to get to know each other. Informal dinners encourage participants to get to know about the non-work aspects (e.g., hobbies, and personal life) of each other's life. Such casual relationship building enables participants to see each other as friends rather than colleagues. As discussed previously, participants are more likely to help their friends. We now present the final proposition:

Proposition 8: The face-to-face meeting mechanism facilitates opensourcing by building casual relationship and re-enforcing the desire to help friends.

CONCLUSION

This study revealed that opensourcing might be conceptualized as a chain of six embedded and cascading mechanisms that work in conjunction with each other. First, it revealed that the symbiotic relationship between the firm and the community is crucial for opensourcing to work. Second, the study revealed a hybrid governance mechanism involving elements of hierarchy and commons-based peer production. This governance mechanism facilitates collaborative actions, and when coupled with effective communication, enables the specification of complex business processes and user requirements as well as the development of quality software in a vertical domain. Third, nurturing the community was revealed as important for opensourcing, with technology transfer between the firm and the community and mutual learning amongst community participants enhancing participants' desire and ability to contribute. Finally, the study proposed eight theoretical propositions that enable opensourcing. Having done this, we call for future research to test these propositions. Specifically future research needs to consider the applicability and operation of these propositions to other contexts (e.g. different types of software, different sectors, and different economic/social environments). The cultural context provided by being situated in Thailand is likely to have affected the nature of the results. Collaboration models and company/community relationships are likely to manifest differently in western and multi-cultural environments, and thus are worthy of further study. The findings are undoubtedly influenced by the sector (medical) and the company leading the initiative. We therefore hypothesize that mechanisms may operate differently in other opensourcing contexts, and are worthy of further research. Finally, in addition to testing the propositions, mechanism-based theorizing necessitates consideration of the cascading relationships between mechanisms across a variety of these opensourcing contexts.

REFERENCES

1. Ågerfalk, P. and Fitzgerald, B. (2008) Outsourcing to an unknown workforce: Exploring opensourcing as a global sourcing strategy, *MIS Quarterly*, *32*, 2, 385–409.
2. Benkler, Y. (2002) Coase's Penguin, or, Linux and the nature of the firm. *Yale Law Journal*, *112*, 3, 367–445.
3. Coase, R. H. (1937) The nature of the firm. *Economica*, *4*, 16, 386–405.
4. Corbin, J. J. M. and Strauss, A. (2008) Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory Sage, Thousand Oaks CA.
5. Crowston, K., Li, Q., Wei, K., Eseryel, U. Y. and Howison, J. (2007) Self-organization of teams for free/libre open source software development. *Information and software technology*, *49*, 6, 564–575.
6. Crowston, K., Wei, K., Li, Q. and Howison, J. (2006) Core and periphery in free/libre and open source software team communications, *Proceedings of the Thirty Ninth Annual Hawaii International Conference on Systems Sciences*, January 4-7, Kauai, HI, USA. IEEE Computer Society, 118a.
7. Cusumano, M. A. (1999) Mozilla gambit reveals risks of open sourcing. *Computerworld*, *33*, 42, 34.
8. Dahlander, L. (2007) Penguin in a new suit: a tale of how de novo entrants emerged to harness free and open source software communities. *Industrial and Corporate Change*, *16*, 5, 913–943.
9. Dahlander, L. and Magnusson, M. G. (2005) Relationships between open source software companies and communities: Observations from Nordic firms. *Research policy*, *34*, 4, 481–493.
10. Dahlander, L. and Wallin, M. W. (2006) A man on the inside: Unlocking communities as complementary assets. *Research policy*, *35*, 8, 1243–1259.
11. De Laat, P. B. (2004) Evolution of Open Source Networks in Industry. *The Information Society*, *20*, 4, 291–299.
12. Fitzgerald, B. (2006) The transformation of open source software. *MIS Quarterly*, *30*, 3, 587–598.
13. Fitzgerald, B. and Ågerfalk, P. (2005) The mysteries of open source software: Black and white and red all over?, *Proceedings of the 38th Annual Hawaii International Conference on Systems Sciences,* January 3-6, Waikoloa, HI, USA. IEEE Computer Society, 196a.
14. German, D. M. (2003) The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice*, *8*, 4, 201–215.
15. Gross, N. (2009) A Pragmatist Theory of Social Mechanisms. *American Sociological Review*, *74*, 3, 358–379.
16. Hedström, P., and Swedberg, R. (1998) Social mechanisms: An analytical approach to social theory, Cambridge University Press, New York.
17. Hofmann, H. F., and Lehner, F. (2001) Requirements engineering as a success factor in software projects. *IEEE software*, 18, 4, 58–66.
18. Howe, J. (2006) Crowdsourcing: A definition (available online at http://crowdsourcing.typepad.com/cs/2006/06/crowdsourcing_a.html; accessed March 2012).
19. Reel, J.S. (1999) Critical success factors in software projects. *IEEE software*, 16, 3, 18–23.
20. Sagers, G. W. (2004) The influence of network governance factors on success in open source software development projects, *Proceedings of the Twenty Fifth International Conference on Information Systems*, December 12-15, Washington, DC, USA, 427–438.
21. Shah, S. K. (2006) Motivation, governance & the viability of hybrid forms in open source software development. *Management Science*, *52*, 7, 1000–1014.
22. Sumner, M. (1999) Critical success factors in enterprise wide information management systems projects, *Proceedings of the 1999 ACM SIGCPR conference on Computer personnel research,* April 8-10, New Orleans, LA, USA, ACM Press, 297–303.
23. West, J. (2003) How open is open enough?:: Melding proprietary and open source platform strategies. *Research policy*, *32*, 7, 1259–1285.
24. Williamson, O. E. (1975) Markets and hierarchies: analysis and antitrust implications: a study in the economics of internal organization, Free Press, New York.
25. Yin, R. K. (2003) Case study research: design and methods, Thousand Oaks, California.