Participation Architectures for Free and Open-Source Software Innovation

Michelle Purcell[1], Susan Gasson[1]
[1]Drexel University

Abstract
Free, Open-Source Software (FOSS) communities are centered on software product innovation. But the social organization of FOSS communities of practice and inappropriate technology platform design can discourage product users from engaging with FOSS communities. The process of suggesting, exploring, and gaining acceptance for new software features can be daunting. We study trajectories of user-participation and social network structures in a vertically-integrated FOSS community to explore what distinguishes successful from unsuccessful participation architecture. We identify socio-cultural and IT platform design (instrumental) barriers to new feature acceptance, to suggest socio-technical affordances for FOSS participation architectures.
Keywords: FOSS communities; participation architecture; design affordances; user engagement; distributed cognition.
doi: 10.9776/16242

Contact: Michelle Purcell mjw23@drexel.edu; Susan Gasson sgasson@drexel.edu.

1 Introduction

Increasingly, business organizations and nonprofit product communities are looking beyond their own boundaries to identify innovative ways to develop products and services by soliciting ideas from their users via the Internet. But innovation requires product developers to integrate user ideas and knowledge of external contexts with existing organizational knowledge and practice (Schlagwein & Bjørn-Andersen, 2014). In essence, users are thrown into an existing context of product design and constraints – this can present challenges to nontechnical users, who have no understanding of the product history, including what has been considered previously. External product users do not understand the conventions, communication genres, and practices expected in community participation, which may cause their ideas to go unheeded. Secondly, they lack knowledge of how organizations evaluate new products and services to make the idea implementable. Lastly, there is the challenge of extending (or reframing) how product developers define product goals and priorities.

One approach to addressing the challenges of peripheral participation is to promote the co-creation of ideas among other users and organizational representatives (Majchrzak & Malhotra, 2013). This requires supporting distributed cognition among networks of communities of practice to facilitate learning from knowledge stretched across those communities (Brown & Duguid, 1991; Lave, 1988). Free and open source software (FOSS) provides a viable context from which to study knowledge co-creation in communities of innovation. First, many open source software projects host technically-mediated new feature request systems as a method to crowdsource ideas to advance the software and employ other technically-mediated methods of engaging users. Secondly, as networks of communities of practice, e.g., peripheral user communities and experienced developer community, these projects exhibit a social structure that contributes to the organizational learning challenges mentioned previously.

This paper explores how new feature request systems function as *participation architectures*, "the socio-technical framework that extends participation opportunities to external parties and integrates contributions" (West & O'Mahony, 2008, p. 146). To that end, a relational affordance perspective, which "looks beyond program functions to inspect the social capabilities that certain technologies enable" (Sun & Hart-Davidson, 2014, p.3535) is used as a lens to elucidate how participation architectures may enable or constrain distributed cognition in support of idea co-creation. This paper explores how the socio-technical elements of a participation architecture – community structures, interaction processes, and technology support – afford users in FOSS communities the possibility to engage meaningfully in software product innovation.

2 Conceptualizing Participation Architectures for FOSS Communities

2.1 Mechanisms for Social Engagement and Visibility in FOSS Virtual Communities of Practice

In studying the concept of a participation architecture, we start with the conditions for effective user participation in FOSS virtual communities of practice. The literature on user participation in software product development distinguishes between *participation* in development, typically assessed by the degree to which individuals play an observable part in community discussions, and *involvement*, which requires a psychological state of identification with process outcomes (Kappelman & McLean, 1992). In the context of online community participation, a third state of *social engagement* appears critical for success. Social engagement denotes active commitment to the social facilitation and structuring of community interactions, learning, and a knowledge of who-knows-what. Mutual engagement is required to bind community members together into a social entity that constructs a repertoire of shared resources, such as collective understandings and expertise (Wenger, 1998). But FOSS communities typically present a system of distributed cognition (Crowston, Wei, Howison, & Wiggins, 2012; Wubishet, Bygstad, & Tsiavos, 2013). Acceptance as a community member is accomplished by means of legitimate peripheral participation, where peripheral participants engage in a socio-cultural apprenticeship to learn acceptable practices and conventions of work (Lave & Wenger, 1991). For example, new developers learn from others which communication channel to use to discuss bugs, or what process to follow for new feature submission.

We have some evidence as to how FOSS communities might support users in social engagement, but more in terms of constraints than enablers. The structure of FOSS community organization constrains peripheral participation. A core of socially-enculturated community administrators and software developers, responsible for strategic planning, goal-setting, decision-making and the coordination of work, is surrounded by annuli of co-developers, active users, and passive users. Participants have decreasing influence, the further they are from the core (Crowston et al., 2012; Terry, Kay, & Lafreniere, 2010). Leadership depends on the trust and mobilization of peers in a social network that spans multiple knowledge-domain boundaries – new ideas need to align with the implicit, shared model of product strategy that this produces. The burden of enculturation needed to engage with core decision-makers discourages participation by peripheral end-users or new developers (Crowston et al., 2012).

In addition, the mechanisms for innovation and strategic planning are embedded in the processes of (software) product release planning, development, and maintenance. Community administrators minimize work coordination needs by planning around superimposed individually-assigned tasks, planning work around increasing levels of additional functionality or complexity (Crowston et al., 2012). As development tasks are defined to be independent, the coordination of work across individuals depends on *heedful interrelating*, where a diverse set of knowledge-domain experts need to pay attention to what other participants are doing (Weick & Roberts, 1993). The triggers for collaboration depend on an individual awareness of task-status across the community – i.e. people have to monitor the work of others for a task-completion status that indicates a need for them to contribute effort (e.g. release testing) and monitor community discussions for indications of unanticipated problems that require their specialist knowledge (Dabbish, Stuart, Tsay, & Herbsleb, 2012). Task coordination thus relies on the mobilization of a closely-knit social network – another impediment to open participation.

Peripheral participation by users and other stakeholders appears to rely on the co-creation of ideas for innovation through debate, and the joint implementation of new features with core developers (Majchrzak & Malhotra, 2013). The co-creation of innovative features allows developers to mentor users in how to present new feature requests and how to understand the barriers to their adoption, while allowing users to engage socially with the community in legitimate peripheral participation (Lave & Wenger, 1991). This is not to suggest that core community membership requires users to engage in software development – there are many forms of community involvement that do not require users to produce code. Core community membership may simply involve users in being sufficiently familiar with community processes and roles to be able to overcome the barriers of “thrownness” identified above. This requires the community to develop socio-technical mechanisms to engage users, who may not be technical, in idea co-creation – for example, online tutorials in how to submit new feature-requests or social mechanisms that pair users with a developer-mentor. We know that end users have a difficult time engaging with FOSS communities on a peer basis and that this undermines the co-creation of software products, to increase their usability and their potential for innovation (Terry et al., 2010; Wubishet et al.,

2013). It is a matter of concern that we know so little about how to support wider participation in innovation processes.

This study builds upon the concept of participation architecture to understand how the sociomateriality of communities when engaging peripheral users may influence idea co-creation. However, the method of study and therefore the focus of what constitutes participation design parameters is different than identified by West & O'Mahony (2008). The authors identified design parameters through conducting interviews with sponsors of FOSS communities seeking to understand what they considered in designing their communities. While they identified intellectual property rights, development approach and governance model as design parameters, this study instead seeks to more fully understand the role of technology and organizational routines in the design of participation architecture to support idea co-creation. To that end, our first research question is:

***Research Question 1*:** What distinguishes successful vs. unsuccessful patterns of idea co-creation during the new feature request process in FOSS communities?

2.2 Social and Technical Affordances of FOSS Participation Architectures

In contrast with Norman's (1988) concept of perceived affordances as design-related visual cues, Gibson's (1977) original concept defined affordances as "action possibilities" that were latent in the environment, and related to the capabilities of a specific agent. For example, a human adult can negotiate high stairs but a human infant cannot – therefore a staircase affords different possibilities for action to an adult than to an infant. An *affordance* can therefore be viewed as a material (in terms of desired agency) instantiation of a possibility for action, by an agent who is capable of exploiting that possibility (Majchrzak & Malhotra, 2013; Treem & Leonardi, 2012). To elucidate how participation architectures may enable or constrain peripheral participation in FOSS community innovation processes, we adopt a relational affordance perspective, which "looks beyond program functions to inspect the social capabilities that certain technologies enable" (Sun & Hart-Davidson, 2014). While many studies have defined affordances for online community support, most focus on an instrumental level of analysis, deriving task-related "patterns" for interaction, rather than providing a situated, social analysis of affordances. Other studies reflect abstracted affordances that relate the social to the technical, but do not consider the implications of the social context. For example, Treem and Leonardi (2012) define four technology platform affordances for distributed collaboration over time: visibility (of other users' actions), persistence (of community communications over time), editability (of communication records), and association (establishing and sustaining social relationships across the community). While identifying social possibilities, these affordances do not reflect the task-related, instrumental materiality of technology-in-practice (Orlikowski, 2010) – i.e. technology in the context of use. Social and instrumental affordances are intertwined. To understand what affordances should be provided by the technical platform for a participation architecture, we need to relate a *social* perspective on social community interactions to a *technical* perspective that reflects the logic of task-related instrumentality within a specific context of use (Sun & Hart-Davidson, 2014).

We know very little about the extent to which online technology platforms support the forms of social engagement discussed above, or enable wide participation in FOSS software product innovation. We do know, however, that the technology platforms supporting FOSS community interactions provide specialized channels for communication that constrain open participation and limit end user visibility in the community (Dabbish et al., 2012; Majchrzak, Faraj, Kane, & Azad, 2013; Treem & Leonardi, 2012). Core community coordinators and developers coordinate work using a diverse set of technology-platform tools such as the mailing list, bug reporting tools, IRC and code repositories for specialized and local (to the community) purposes. It takes a technically literate, socially-connected, and enculturated end user to understand the difference between the discussions of product features that take place via one channel versus another.

West & O'Mahoney (2008), who originated the term participation architecture, structure the concept for use in tailoring a FOSS community's strategic policy concerning openness to innovation, identifying intellectual property, community management, and development approach as design considerations. However, it is clear that we know remarkably little about the socio-technical affordances needed for open participation in FOSS communities, so our analysis was focused on that aspect. We use the affordances of online communal knowledge sharing identified by Majchrzak et al. (2013) as a lens to examine instrumental affordances provided for peripheral social engagement:

1. *Metavoicing:* Engaging in the ongoing online knowledge conversation by reacting online to others' presence, profiles, content, and activities

2. *Triggered attending:* Engaging in the online knowledge conversation by remaining uninvolved until a timely automated alert informs the individual of a change in the specific content of interest
3. *Network-informed associating:* Engaging in the online knowledge conversation informed by relational and content ties
4. *Generative Role-Taking*: Engaging in the online knowledge conversation by enacting patterned actions and taking on community-sustaining roles in order to maintain a productive dialogue among participants

***Research Question 2*:** To what extent do FOSS community technology platforms support social and instrumental affordances for social engagement of peripheral users for idea co-creation?

3 Research Site and Method

3.1 FOSS Community Context

Evergreen is an open library system community that develops and maintains an open-source software product to support libraries, mainly in the USA and Canada. The software system helps library patrons find materials, and helps libraries manage, catalog, and circulate those materials. It is developed to be scalable and robust across any size or complexity of collections. The project uses Launchpad.net, an open software platform that allows open source software communities to manage bug reports, wishlist ideas, translations, and blueprints for the future development of their products. The project employs a deployment business model meaning that while the code is free users are willing to pay support, subscription, and professional services to maintain and customize the software (Chesbrough & Appleyard, 2007). The majority of development is done by paid developers at software companies and some of the institutions using the software. The project would be considered relatively small based on lines of code and users. As part of the process for submitting a new feature request, if it cannot be funded for development by the proposer participants are instructed to submit their request to Launchpad.net.

3.2 Research Method

The study combined enculturation though longitudinal analysis and observation of interactions in the online community forum, with stakeholder interviews, and trace ethnography, where the reconstruction of work practices from online records and communication traces over time (Geiger & Ribes, 2011). As part of this approach, artifacts from an open source integrated library system including web pages, bug reports, internet relay chat (IRC), and mailing lists were collected and analyzed using qualitative analysis, to answer the research questions posed above. We studied the library system project over a period of six months, to ensure enculturation in the processes of heedful interrelating that underpinned community production and maintenance. As we were interested in idea co-creation among peripheral users and other community members, we studied new feature requests submitted over five years (a sample of 207 feature requests and 2325 activity-log records were analyzed in total). New feature requests were chosen for two reasons. Firstly, besides the mailing list which is typically used for product support questions, new feature requests are a main method by which peripheral users engage with the community. Secondly, FOSS is particularly noted for lagging behind the commercial software world in engaging end-users, an area for innovation is to learn how to involve users in subsequent redesign (Nichols & Twidale, 2003).

Reassembling practice involves a process of systematic, post hoc assembly of the “vapor trails” left in electronic work logs, communication channels, and online forums. Communications and online traces were organized by product-change (bug-fix or feature implementation), topic, participants, and time-period, to identify trajectories of interaction that led to a specific outcome (Gasson & Waters, 2013).

We analyzed the social networks that underlay the trajectories of change, to understand community roles, interactions, and modes of social mediation (Amrit & Van Hillegersberg, 2010; Whelan, Parise, de Valk, & Aalbers, 2011). To explore the centrality of some community members vs. others, we evaluated the degree centrality of participants in the social network of interactions represented by activity-logs. To understand the extent to which core users facilitated social engagement by peripheral community members, we evaluated the betweenness centrality of participants in the social network.

Finally, we identified socio-technical affordances for community participation, analyzing the social mediation mechanisms and technology platform features that enabled or constrained various trajectories of interaction over time. We then related these social community and technology support features to the capabilities identified for peripheral social engagement in innovation.

4 Research Findings

4.1 Trajectories of Idea Co-creation

Several trajectories were identified, depending on the type of request submitted. This was done by identifying common patterns of participation and states of progression among the new feature requests. Figures 1 through 4 presents example trajectories. Standard font designates an actor or state of the request. Italicized font presents tool-mediation communication. These were assigned an importance (e.g., critical, high, wishlist) and a status (e.g., new, confirmed, fix committed) by one of the core development community members. At this point, the only mechanism to have these actioned was if the priority was set of Critical or High. If a feature request was assigned a priority of "Wishlist," it was unlikely to be implemented unless additional action was taken by the originating user or a core developer – for example, supplying additional rationale, or volunteering code development effort.

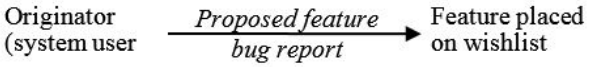

Figure 1. Trajectory of New Feature Requests Submitted Via Bug Report

It appears that a confluence of influence is required for a new feature request to succeed. It was clear from our analysis that developers maintained a clear mental model of system priorities and make decisions based on the synergies between new feature requests and planned work, as it was common to see a new feature request related to other bug reports (feature requests or product issues). But development priorities were driven by the request Importance. Socially-connected users were able to leverage their connections via email, or – better – via IRC discussions to engage with core developers. Engaging with a core developer via interactive discussions in online discussion boards or an IRC chatroom could lead to the feature or issue being acted upon.

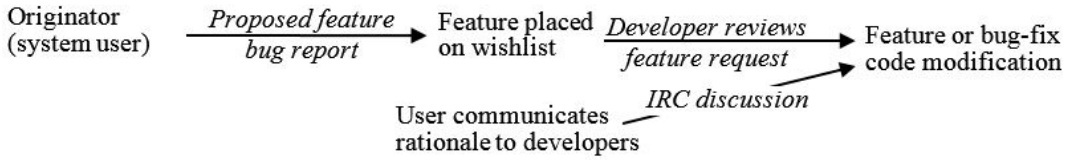

Figure 2. Trajectory of New Feature Requests Submitted By "Connected" User

As developers maintained a community-wide mental model of feature requests, issues, and planned work, they could relate a new feature request to prior requests, or issues in related areas of the product. It was common to see developers bringing in additional users, and occasionally (but less frequently), users adding themselves to the discussion of an issue in the bug report forum. Individuals would cast new light on the issue, provide additional information on how, where, or why it took place, and collaborate in constructing a big picture understanding of how the system worked in the context of specific practices.

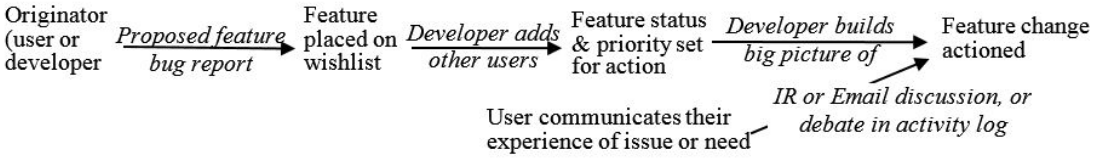

Figure 3. Trajectory of New Feature Requests, Supported by Other Users

The most common innovation trajectory was for a user/developer to submit a new feature request, based on discussions with users, or evaluation of the system software. Users working for large organizations wielded influence, especially as their organizations would often pay for their time in fixing issues. Developers would combine in discussions of the new feature, often calling in users or engaging in long exchanges of ideas and code across channels of communication (activity-log discussions, IRC, email).

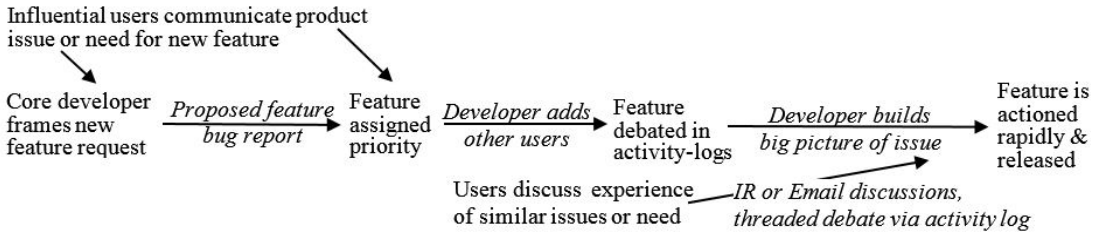

Figure 4. Trajectory of New Feature Requests, Submitted Via Developer

It is clear that implementation of new feature requests depends on user knowledge and connectedness:

1. The user must understand informal rules for submitting and justifying new feature requests. Rules are communicated in discussions with developers, but do not appear to be recorded anywhere.
2. The most influential users are those who can action a change themselves (modify the software), either because their employer pays them to do so, or because they have the time, technical background, and motivation to engage with code development.
3. The ability to modify code is not sufficient to achieve new feature implementation. Core community members need to appreciate and understand the rationale for change and its impact on existing code. Users and developers individuals would collaborate in small groups to explore issues - at least one group member needed to understand the *implications* of changes across a complex set of product version releases that behaved in different ways, for changes to be implemented.

4.2 Exploring FOSS Community Member Roles

We therefore analyzed the respective social roles of various FOSS community members in making different types of decisions. We performed a social network analysis to understand the enacted roles of community members (Amrit & Van Hillegersberg, 2010; Whelan et al., 2011). A sample of 2325 activity-log records for the 207 feature requests were analyzed. This data represents formal, product-related activity (not discussions of the bug reports, which might involve additional users, developers, or library stakeholders). Each activity-log interaction was analyzed to provide an interaction sender and receiver, revealing the core roles across a community network of actors who collaborated to investigate, develop ideas for, or manage the progression of feature requests. Because of the space limitations in a conference paper, we present only the findings, rather than discussing measures of centrality or connectedness in detail. The roles of key players identified by the social network analysis was explored further by means of a qualitative content analysis of the activity-logs.

An analysis of degree-centrality revealed five core members who played an administrative role in the community. Three of the five were also the most influential boundary-spanners in the community (as indicated by their betweenness centrality), although the other two core administrators also spent significant time facilitating boundary-spanning activities, indicating a high degree of distributed decision-making in community goal-setting. The core software developers (again, indicated by degree centrality) appeared less influential than those performing a more administrative role. The majority of developers seemed to have more degree centrality than betweenness centrality in the community network, indicating a community of individuals who like to work alone.

Activities to process feature requests only involved a small sub-network of densely-connected administrators and developers within the wider community. Five core community members acted as social network facilitators, indicated by their high network betweenness centrality. They engaged in administrative community facilitations such as determining the importance, status, designated milestone, and assigned-developer for feature requests. In addition, three of the five core administrators also facilitated idea-development and feature co-creation, connecting users with developers to get code modifications completed, inviting other knowledgeable users to join debates around the utility of new features, and expediting collaborative work around product releases. There is little that happened where they were not involved. The two other core administrators were less connected to the wider network, but still played a key role in community and product management decisions. These individuals spent much more time on code administration and much less on community facilitation. Of course, they may have been active in community facilitation behind the scenes, in IRC chats with users, or email communications. But we would expect this to reflect in the activity-logs, which generally record the results of behind-the-scenes transactions and decisions.

A subset of five core software developers played a central role in mediating community knowledge. These developers maintained group memory for the community, especially where design rationale was concerned. A smaller subset of four software developers appeared to be active in another form of boundary-spanning, bringing together groups of other developers and users, to discuss priorities and determine which features should be acted upon. One developer appeared to fulfill both roles, being consulted by other developers in a "knowledge expert" role whenever an issue was related to prior work, and also acting as a social facilitator who could suggest other community members who knew about specific issues. Other core developers, while influential because of their role in understanding how to implement specific features, were less influential in boundary-spanning, relying on the core administrators or the "group memory" software developers for community knowledge and the "user mediation" developers for the inclusion of users who were experienced in specific areas of product functionality.

The only area where we saw a relatively high degree of distributed community leadership was in expertise mapping. Most community members appeared to develop a mental map of *who-knows-what* through working together that was useful in coordinating their work and in aiding the work of others. Developers established strong ties with connections suggested by the community administrators and routinely called on other developers by name when they needed help with understanding the rationale behind the design of specific product features, or when they needed to be introduced to knowledgeable users who could evaluate the implications of a major product change.

4.3 Technology Affordances For Community Participation

Peripheral participation was relatively easy, as the community was welcoming and open to user suggestions. But successful adoption of new feature requests tended to occur when a peripheral user was either sufficiently well connected to mobilize network support for a product modification, or when a peripheral user had participated across communication channels and discussion forums over time, so had understood how various channels were used and had become relatively visible (recognized by) core community members. Users needed to be introduced to ways of participating – it would be easy to miss the existence of the informal community discussions that took place on the IRC channel and proceeded in parallel with the formal discussions (on the bug reports discussion boards, or via activity-log updates), if one did not know that these were taking place. As a result, the trajectory of interaction shown in Figure 1, where a new feature request was assigned with "Product Wishlist" status, then waited for ever for a core developer to take an interest in this feature, was depressingly common.

Meta-voicing was lacking as bug reports were not *categorized* in a way meaningful to product developers and users. This hindered attempts at *group memory*. The tags referred to planned product upgrades or fixes, using terms that were meaningful to core developers (e.g. "bitesize," "pickup," or "jspacremovalblocker"). This lack of affordance for meaningful feature-tracking even impeded *developers'* ability to locate specific feature requests – we saw quite a few replies to bug reports that contained variations of "This is a known issue which was discussed a few months ago, but I'll be darned if I can find that discussion."

For users, it was impossible to discover if a feature had been requested previously, except via a blanket search. Even then, users often defined the same issue in different terms based on their local organizational conventions, leading to developers pointing out that a new feature request duplicated a previous request or had been found impossible to implement. But as previous records used different keywords or descriptive terminology, users would have no way to discover this without reading through every single bug request (there were 1035 live bug requests and many thousands of unactioned, archived requests at the time of writing).

There was no triggered attending mechanism to further action on a bug report, except for subscriptions to bugs of interest. It was common to see feature requests stay incomplete for months, or years. While the platform did provide an indicator of heat which communicated its severity and impact, this was not used by the community as an indicator of importance that might trigger further action. One request in our sample was originated in 2011, then assigned to a developer who commented that he had a fix for the code. There was a reminder posted in 2012 and again in 2013. It had not been updated since that date, even though the requested change to the code appeared to have been made, but never committed or released. The developer in question was still active in the community, but just never followed up – presumably because the system did not tell him he still had work outstanding. The three mediating core coordinators discussed earlier acted as a human tracking system for the community, spending a lot of time reviewing incomplete bug reports and fixes.

Lastly, the bug reporting tool had no community coordination features. Its interface focused on product-issue tracking and software development progress rather than making the activities of other users

visible to community members. *Network-informed associating* was supported by threaded discussions in activity-logs, but as there was no way to easy way to identify connections within the community there was almost no interaction between users. This was not surprising, as the technology platform provided no mechanism to make users visible to each other, to identify users with similar issues, or to connect users with developers, except when users were invited into discussions of feature requests or bug reports by an individual developer to whom they were already known.

5 Discussion of Findings

5.1 Successful Vs. Unsuccessful Patterns Of Idea Co-Creation

The ability of peripheral community members to participate in product innovation appeared limited to interpersonal influence on core community administrators and developers, exerted via personal communications such as email or phone, or via the technical developer communication channel (IRC). To use the latter channel, a user needed to be technical and to understand the developer "language" used in such communications. This presented a high barrier to entry for non-technical community members and explains why feature-request submissions are dominated by the more technical, developer community members. When non-technical users attempted to request new features, these appeared less successful than technical user/developer requests.

However, the success of a new feature request also depended on being able to provide effort to implement the feature, which depended on social connectedness within the core developer network. This links us to the complementary roles of innovation import identified by Whelan et al. (2011). *Idea scouts* are people with extensive external networks, who can leverage their connections to identify innovation ideas that will fit with the company – but these are unlikely to be successful without a social network that includes *Idea connectors*, boundary-spanners who can facilitate the internal social network connections that will result in idea implementation. In our analysis, the core community administrators acted as idea connectors and also often as idea scouts. The social network analysis, coupled with our content analysis of activity-logs identified one non-administrative developer who was noticeable in acting as an idea connector, connecting users with developers who needed an understanding of product features in practice and supplying knowledge about the rationale underpinning prior feature development. This role may explain his extensive social network, as other community members recognized the key role he played in connecting them with individuals who could ensure implementation of their ideas.

It is clear from our analysis that it takes a village to make a code modification – or at least, a quorum of core administrators! One surprising aspect of the analysis was how centralized most of these roles were. While community members collaborated in sharing ideas and understanding of how the product did, or could work, there was little distribution of community or product management. In such a distributed community, we were expecting a diverse range of community leadership roles, with various individuals responsible for leadership in different areas. What we discovered was that almost all of the boundary-spanning, interpretation of importance, and community coordination was performed by five or six individuals. It may be that the vertical integration nature of the product, dominated by an administrative board that represents the largest user organizations, has led to this centralization of community roles. This is a relatively small community; a larger community might require more distributed leadership.

The exceptions to the centralization of leadership in this community were in the roles of group memory management and of boundary-spanning mediation between users and developers. This is encouraging. People need to place their work in the context of other work and with knowledge of lessons learned by the community about what works in this context. A widespread understanding of what the community/group knows is essential for this context, if communities are not to repeat past mistakes.

5.2 Affordances for social engagement and idea co-creation by peripheral users

Few of the affordances identified by Majchrzak et al. (2013) were supported by the technology platform. There were no mechanisms to enable user community members to engage in ongoing conversations (metavoicing), unless they were invited specifically by developers. There was no mechanism for triggered attending (except for subscribing to bugs of interest), although community administrators had operationalized this concept as part of the technology-in-practice framework of the project, by developing personal routines for checking bug reports at periodic intervals. There was no technology platform support for network-informed associating, although we did locate a web-page that listed the members of each subproject. However, without any context as to the goals or deliverables of subprojects, this was unhelpful to peripheral users. Generative role-taking appeared to be managed solely by the motivated practices of individual developers, who spent a great deal of time looking for and assisting peripheral

community members. However, the lack of technology platform support for this affordance meant that a huge number of new feature requests waited without feedback or attention following assignment to the product wishlist.

The findings identified three social success factors for social engagement by peripheral users: (i) understanding the community rules for submitting and evaluating new feature requests, (ii) the ability to engage in code development or joint feature development, and (iii) the ability to relate product modifications to planned product release versions. We found a basic process document that explained how to submit a feature request, but did not provide the information that a new user would need to have their request acted upon. The community WIKI did not provide guidance on how to participate or how to locate developers with whom to collaborate (although a long-superseded page in the previous WIKI did cover some basics of how to participate). The community affordances for these aspects of social engagement were therefore not supported. The socio-technical participation architecture provided very limited affordances for network-informed associating, although as discussed above, developers employed their own routines and coordinated network roles to overcome this limitation. Finally, because of the limited visibility of other community activities, many opportunities for generative role-taking were missed, unless an individual core developer was especially proactive and aware. This meant that core administrators had to monitor and mediate bug, feature, and code discussions.

We identified three areas of significant deficiency in affording community support for instrumental work-tasks. *Categorization structures* embedded in the technical report system obscured the significance of a change request and impeded attempts to retrieve historical design rationale. *Group memory* was poorly supported by the technology platform, but supported by an active social network of facilitative core developers and administrators. *Heedful interrelating* was impeded by the technical platform (the launchpad.net system) by not providing a robust triggered alerting mechanism, preventing community coordination of work (Weick & Roberts, 1993) and obscuring task-visibility (Dabbish et al., 2012). To compensate for this, the three core administrators provided a human work-tracking-system for the community.

6 Conclusions

This paper provided a framework for participation architectures in online FOSS communities. We presented findings from a case study of a vertically-integrated open-source software community, to explore the social and technical dimensions of social engagement by users. We found that while work coordination and release management in the online FOSS community that we studied was relatively centralized, group memory management and understanding of who-knows-what was distributed across more individuals. When organizational leadership is centralized in a few individuals, this provides the high reliability required for distributed organizations to function. Centralized decision-making may reduce the variety of perspectives required for innovation – but it may alternately provide single decision-points that facilitate the outside-in knowledge transfer needed for innovation (Whelan et al., 2011). The relatively centralized decision-making and administration combined with distributed product knowledge to provide a collective intelligence that was enabled through heedful interrelating. But this was poorly supported by the technical community platform, which did not afford distributed possibilities for group memory management or distributed collaboration over time.

Because the technology platform was designed around software process reliability rather than community knowledge exchange, this constrained its potential to support peripheral participation by product users and limited their social engagement. Even the meta-framework suggested by Majchrzak et al. (2013) to resolve tensions between individual and community aims in collaborative participation proved too complex for a platform that was obviously chosen on its ability to coordinate software releases among people enculturated in specific engineering practices. Our social network analysis of community roles demonstrated that the ongoing survival of the community depended on social mediation by motivated individuals. Our future work will explore other online innovation communities, with more diverse structures and constraints, to develop a general model of affordances and inform the design of technology-platforms for FOSS participation architectures.

7 References

Amrit, C. & Van Hillegersberg, J. (2010). Exploring the impact of socio-technical core-periphery structures in open source software development. *Journal of Information Technology, suppl. Special Issue on Social Networking, 25*(2), 216-229.

Crowston, K., Wei, K., Howison, J., & Wiggins, A. (2012). Free/Libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR), 44*(2), 7:1 - 7:35. doi: 10.1145/2089125.2089127

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). *Social coding in GitHub: transparency and collaboration in an open software repository.* Paper presented at the Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work.

Gasson, S. & Waters, J. (2013). Using A Grounded Theory Approach To Study Online Collaboration Behaviors. *European Journal of Information Systems, 22*(1), 95–118.

Geiger, R. S. & Ribes, D. (2011). *Trace Ethnography: Following Coordination through Documentary Practices*. Paper presented at the Proceedings of the 44th Hawaii International Conference on System Sciences, Los Alamitos, CA.

Gibson, J. J. (1977). The Theory of Affordances. In R. Shaw & J. Bransford (Eds.), *Perceiving, Acting, and Knowing*. Hillsdale, NJ: Lawrence Erlsbaum Associates.

Kappelman, L. A. & McLean, E. R. (1992). Promoting Information System Success: The Respective Roles of User Participation and User Involvement. *Journal of Information Technology Management, 3*(1), 1-12.

Lave, J. & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge UK: Cambridge University Press.

Majchrzak, A., Faraj, S., Kane, G. C., & Azad, B. (2013). The Contradictory Influence of Social Media Affordances on Online Communal Knowledge Sharing. *Journal of Computer-Mediated Communication, 19*(1), 38–55.

Majchrzak, A. & Malhotra, A. (2013). Towards an information systems perspective and research agenda on crowdsourcing for innovation. *Journal of Strategic Information Systems 22 (2013), 22*(4), 257–268.

Norman, D. A. (1988). *The Psychology of Everyday Things* (Also published as: The Design of Everyday Things 1990 Doubleday New York ed.). USA: Basic Books, Harper-Collins.

Orlikowski, W. (2010). The sociomateriality of organisational life: considering technology in management research. *Cambridge Journal of Economics, 34*(1), 125-141.

Schlagwein, D. & Bjørn-Andersen, N. (2014). Organizational Learning with Crowdsourcing: The Revelatory Case of LEGO. *Journal of the Association for Information Systems, 15*(11), 754-778.

Sun, H. & Hart-Davidson, W. F. (2014). *Binding the Material and the Discursive with a Relational Approach of Affordances.* Paper presented at the 32nd. Annual ACM Conference on Human Factors in Computing Systems New York, NY.

Terry, M., Kay, M., & Lafreniere, B. (2010). *Perceptions and practices of usability in the free/open source software (FOSS) community*. Paper presented at the 28th International Conference on Human Factors in Computing Systems CHI' 10.

Treem, J. W. & Leonardi, P. M. (2012). Social media use in organizations: Exploring the affordances of visibility, editablity, persistance, and association. *Communication Yearbook, 36*, 143-189.

Weick, K. E. & Roberts, K. H. (1993). Collective Mind In Organizations: Heedful Interrelating on Flight Decks. *Administrative Science Quarterly, 38*.

Wenger, E. (1998). *Communities of Practice - Learning Meaning and Identity*. Cambridge UK: Cambridge University Press.

West, J. & O'Mahony, S. (2008). The role of participation architecture in growing sponsored open source communities. *Industry and Innovation, 15*(2), 145–168.

Whelan, E., Parise, S., de Valk, J., & Aalbers, R. (2011). Creating employee networks that deliver open innovation. *MIT Sloan Management Review, 53*(1), 37-44.

Wubishet, Z. S., Bygstad, B., & Tsiavos, P. (2013). A Participation Paradox: Seeking the Missing Link between Free/Open Source Software and Participatory Design. *Journal Of Advances In Information Technology, 4*(4), 181-193.