Journal of the Association for Information Systems
JAIS

Special Issue

IT-Enabled Knowledge Creation for Open Innovation

U. Yeliz Eseryel
University of Groningen
u.y.eseryel@rug.nl

Abstract

Open innovation is increasingly important for researchers and practitioners alike. Open innovation is closely linked to knowledge creation in that, with open innovation, knowledge inflows and outflows are exploited for innovation. In the information systems field, open innovation has been closely linked to open source software development teams. However, the literature has not yet identified how open source software development teams use information technologies to create knowledge to bring about open innovation. This study fills in this gap by asking the following research questions: RQ1) How do innovative open source software development teams create knowledge?, and RQ2) What types of information technologies do innovative open source software development teams rely on for enabling knowledge creation? I answer these research questions with a revelatory case study. The findings contribute to the knowledge management theory by identifying how three of the four knowledge creation modes identified by Nonaka and Takeuchi (1995) manifest through different behaviors in the IT-enabled open innovation setting compared to behaviors observed in the organizational setting. The findings also contribute to information systems theory by identifying the role of information technologies in enabling knowledge creation for open innovation. This study further provides researchers and practitioners with ways of identifying knowledge creation by analyzing information technology artifacts, such as mailing lists, issue trackers, and software versioning tools.

Keywords: *Open Innovation, Open Source Software, Knowledge Creation, Knowledge Management, Case Study.*

* Lorraine Morgan was the accepting senior editor. This article was submitted on 5th August 2013 and went through two revisions.

Volume 15, Special Issue, pp. 805-834, November 2014

1. Introduction

Innovation is as an idea, practice, behavior, or artifact that is perceived as being new by the adopting unit (Daft, 1978; Damanpour, 1987; Tushman & Nadler, 1986; Zaltman, Duncan, & Holbek, 1973). The literature identifies two types of innovations: 1) product innovations, which involve the development of new technologies, products, and services; and 2) process innovations, which involve a change in the way a product is made or a service is provided (Robey, 1986; Zmud, 1982). Product or process innovations can be disruptive by replacing similar products or outperforming traditional ways of working, consequently giving the innovators a competitive advantage (Barlow & Li, 2010). While the term innovation usually brings to mind disruptive innovations that change markets, innovations can also be incremental in nature (Barlow & Li, 2010). Incremental innovations represent cumulative changes in a process or product in the form of "minor improvements or simple adjustments in current technology" (Dewar & Dutton, 1986, p. 1423). Orlikowski (1991, p. 5) suggests that "new versions of a particular computer software constitute incremental innovations as they enhance or extend existing features and capabilities" (as opposed to a new software design and architecture as in the case of IBM's database product DB2, which moved to a relational database structure from a hierarchical one). Thus, for the purposes of (open source) software, the development of new features constitutes an example of incremental innovation.

Open innovation is the process of systematically encouraging and exploiting a wide range of internal and external knowledge sources for accelerating innovation (Chesbrough, 2003; Conboy & Morgan, 2011). External factors such as technology explosion, shortening of technology development cycles, and the globalization of technology change how companies organize their research and development activities (Trott & Hartmann, 2009). Specifically, there has been a move towards increasing openness in innovation strategies stemming from shorter innovation cycles, industrial research, and the rising costs of development (Gassmann & Enkel, 2001). Various authors define openness is defined as the number of different sources of external knowledge that a firm draws on in its innovative activities (Conboy & Morgan, 2011, p. 537).

In the software development context, open innovation often brings to mind open source software development teams. This is perhaps because teams developing the Linux operating system and the Apache server (among others) have been wildly successful in exploiting knowledge that is spread around the world. They are able to tap into globally distributed knowledge to create high-quality software that competes against high-profile commercial software. Open source software development teams are teams of individuals who collaborate from around the world using information technologies to develop software, the source code of which is available to be viewed and changed. Von Hippel and von Krogh (2003) identify open source software development as a unique compound of private investment and collective action models of innovation. They call the open innovation model provided by open source software development teams as “private-collective innovation model” and define it as a model where participants “use their own resources to privately invest in creating novel software code…and choose to freely reveal it as public good” (Von Hippel & von Krogh, 2003, p. 213). The movement towards openness in software development is seen either in the form of traditional companies such as IBM increasingly utilizing open source software or companies such as Nokia incorporating the open innovation practices of open source software teams into their proprietary methods (e.g., Conboy & Morgan, 2011; Dinkelacker, Garg, Miller, & Nelson, 2002; Fitzgerald, 2006; Gurbani, Garvert, & Herbsleb, 2010). There is an ever-increasing movement towards using either open source software teams or their technology-intensive practices that foster open innovation (Dinkelacker et al., 2002; Gurbani et al., 2010; Morgan, Feller, & Finnegan, 2011; Wesselius, 2008). This trend can be explained by the fact that innovation and creativity are at the core of software development (Brooks, 1995; Carayannis & Coleman, 2005; Cougar, 1990).

In understanding how open source software development teams provide open innovation, we can use Nonaka and Takeuchi’s (1995) approach to investigating innovation, which they used to examine Japanese companies’ innovation processes. Nonaka and Takeuchi identify knowledge creation as the

source of innovation (Figure 1) and they explicate the innovation process of the Japanese companies by studying their knowledge-creation processes.

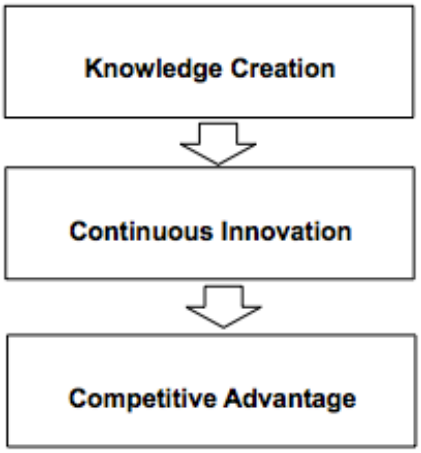


Figure 1. Knowledge as the Source of Innovation (Nonaka & Takeuchi, 1995)

They identify that innnovative Japanese companies continuously innovate by bringing in external knowledge, sharing it widely within themselves, and using it to develop new technologies and products. Innovative Japanese companies also share this knowledge back with those outside the company.

Knowledge is the most valuable resource for software development (Robillard, 1999) and a key ingredient of open innovation. Critical problems faced in software development may be addressed by improvisation and innovation (Brooks, 1995) using different types of knowledge. Innovative activities play a key role in all aspects of development starting from requirement definition through to program design (Cougar, 1990), which are all areas that require specialized knowledge. Innovation is required because rapidly evolving technology changes the software development in that it influences which resources should be utilized and how. Increasingly, software development requires more-complex solutions (Cougar, 1990).

There is a lack of understanding about how different methods and technologies can facilitate innovation in software development in general (Conboy & Morgan, 2011). As reflected in open innovation's definition (Chesbrough, 2003), internal and external knowledge transfer across boundaries is the core enabler of open innovation. While the IS field treats knowledge management and open innovation as separate literature streams, open innovation depends heavily on knowledge creation, knowledge sharing, and knowledge management. Therefore, the information systems theory should contribute to the open innovation theory by explicating how information systems enable knowledge creation in open innovation teams. To this end, I pose the following research questions

RQ1: How do innovative open source software development teams create knowledge? Specifically:

RQ1a: What type of knowledge-creation modes and behaviors exist in innovative open source software development teams?

RQ1b: How do the developers and users contribute to knowledge creation in innovative open source software development teams?

RQ2: What types of information technologies do innovative open source software development teams rely on for enabling knowledge creation?

To answer these research questions, I conducted a revelatory case study a highly innovative open source software development team.

2. Literature

Companies find knowledge sharing an important strategic goal, albeit one that is highly challenging (Davenport, 1996; Davenport, 1997; Orlikowski, 1993). According to the KPMG Consulting (2000), 80 percent of U.S. corporations reported having knowledge management initiatives (which are technology or process initiatives) that aim to improve knowledge creation and sharing in and outside themselves. Ambrosio (2000) reports that at least half of all knowledge management initiatives fail. Open source software development teams provide an ideal case of knowledge creation and sharing among developers and users[1]. Research organizations, governments, and corporations increasingly want to emulate the open innovation model presented by open source software development teams (Chesbrough, 2003; Davenport, 1997; Goldman & Gabriel, 2005; Morgan et al., 2011).

In order to emulate the open innovation model of open source software development teams, first, the types of knowledge creation that happens in these environments need to be identified. Second, researchers point out to the need to identify how different internal and external members contribute to knowledge creation. For example, Agerfalk and Fitzgerald (2008) find it crucial to understand the differences in the knowledge contributions of different types of developers (Agerfalk & Fitzgerald, 2008). Other researchers suggest that, in order to create open innovation communities similar to open source software development teams, companies need to engage in and harvest external knowledge by outside contributors (von Krogh, Spaeth, & Lakhani, 2003; West & O'Mahony, 2005).

Yet, the types of knowledge-creation processes and behaviors that contribute to open innovation to innovate and the role of information technologies in enabling such knowledge creation are not known. Because knowledge creation lies at the heart of open innovation, in Section 2.1, I discuss a prominent theory on knowledge creation. I use this theory to identify the knowledge-creation behaviors in open innovation settings and how information technologies enable them.

2.1. Knowledge Creation

While there are various definitions of knowledge (see Eseryel, Eseryel, & Edmonds, 2005 for a summary), I define it in this paper as applied information that actively guides task execution, problem solving, and decision making (Liebowitz & Beckman, 1998). I use this definition because it highlights two aspects of knowledge: 1) that it is processed information; and 2) that it guides action. Nonaka and Takeuchi (1995) argue that knowledge creation occurs first at the individual level, and then, over time, this knowledge becomes organizational knowledge.

At the epistemological level, Nonaka and Takeuchi (1995) identify two types of knowledge: tacit[2] and explicit. Tacit knowledge is difficult to codify and articulate because it arises out of experiences. In contrast, explicit knowledge can be documented and passed on from an individual to others with relative ease. In this study, in line with Nonaka and Takeuchi (1995), I operationalize knowledge creation as a movement between tacit and explicit knowledge.

[1] In this paper, I use the term "developers" to refer to the core team members in an open source software development team. Users are the peripheral members who are not part of the core team. Yet, users make contributions in the form of bug reports, and from time to time small fixes. In open source software, developers very typically also use their own software (Agerfalk et al., 2008). The distinction between developers and users is not based on whether or not an individual uses the software. Rather, the distinction is based on whether an individual is part of the core development team (i.e., playing a developer role) or not. It is possible to use core versus periphery to indicate this relationship; however in this paper, following Agerfalk et al. (2008), I use "developer" versus "user".

[2] While the conceptualization of Nonaka and Takeuchi is often cited, and also used in this paper, Polanyi (1966) coined the term tacit knowledge.

Table 1. Modes of Knowledge Creation (Adapted from Nonaka and Takeuchi, 1995)

Knowledge creation-mode	Knowledge creation (from → to)	Strategies for knowledge creation in this mode
Socialization	Tacit → tacit	Socialization refers to shared-experience via observations, imitation, and practice in a specific context (e.g., apprentice-master model).
Externalization	Tacit → explicit	Externalization refers to collective reflection. This happens by sequential use of metaphors, analogy, and modeling.
Combination	Explicit → explicit	Combination refers to reconfiguration of existing information. This happens by sorting, adding, categorizing, combining existing information found in written documents (e.g., reports, memos, e-mail, books, etc.) and verbal exchanges. Knowledge management technologies support creation of knowledge in this mode.
Internalization	Explicit → tacit	Internalization happens via self-reflection and documentation such as verbalizing or diagramming new personal experiences and knowledge into documents, manuals, and oral stories.

Nonaka and Takeuchi (1995) identify four modes of knowledge creation that transfers knowledge among the individuals resulting in organizational knowledge creation (Table 1). The modes of knowledge conversion include socialization (from tacit to tacit knowledge), externalization (from tacit to explicit knowledge), combination (from explicit to explicit knowledge), and internalization (from explicit to tacit knowledge).

2.2. Open Innovation and Knowledge Creation in Open Source Software Development Teams

Open source software development teams are open innovation teams that use a private-collective innovation model; that is, participants use their own resources to privately create novel software code and they share this code for free (von Hippel & von Krogh, 2003). In open source software projects, the developers create most of the innovation because the developers, rather than the software manufacturers, are the ones who make most of the contributions (von Hippel & von Krogh, 2003, p. 214). von Krogh et al. (2003) emphasize the importance of setting up structures that enable knowledge contribution. I discuss the open source software team structure and how it is set up for knowledge creation below.

We can summarize the open source software team structure as developers at the core and active users at the periphery (Crowston & Howison, 2006). While the developers at the core play a key role in designing and initially developing software, the users at the periphery contribute notably to software development and the sharing of relevant knowledge at open source software teams (AlMarzouq, Zheng, Rong, & Grover, 2005; Cox, 1998; Crowson & Howison, 2005; Gacek & Arief, 2004; Mockus, Fielding, & Herbsleb, 2002). The users at the periphery contribute their time and innovativeness to improve the product quality (Setia, Rajogopalan, & Sambamurthy, 2012). Users contribute to open innovation by infusing new ideas to improve the software (Chesbrough, 2003; Setia et al., 2012). Von Hippel and von Krogh (2003) typically identify the "novel software code" as the innovation provided by the open source software teams. They find that the cost to the innovator of freely sharing their software code is very low. The rewards, in return, include reputation and reciprocity (Lerner & Triole, 2000; von Krogh, 2002).

Setia et al. (2012) find that the peripheral participants (users) contribute to software's quality and its popularity. For example, with Linux, the users at the periphery generate patches and they report

software bugs (Lee & Cole, 2003) and thereby contribute their knowledge to the innovation process. Developers at the core then evaluate these contributions and provide suggestions for the work in progress to encourage improved variations in the new product development process (Lee & Cole, 2003). An important future research area identified by Setia and colleagues (2012) is whether information technologies can further facilitate more effective contributions from peripheral users, which may contribute to increased innovation.

At the core of open innovation is knowledge creation. For example, Lee and Cole's (2003) community-based knowledge-creation model includes an innovative process. They suggest that the community-based knowledge creation involves rules and structures that encourage members to critically evaluate existing knowledge, innovate, and rapidly eliminate error. Presence of knowledge creation in open source software teams can be evidenced by the existence of bountiful learning opportunities (Hars & Ou, 2001; Hermann, Hertel, & Niedner, 2000; Himanen, Torvalds, & Castells, 2001; Kohanski, 1998) and various practices for socialization (Crowston & Annabi, 2005; Weber, 2004), and product and process documentation.

An important aspect of knowledge creation in open source software development teams is providing criticism and correcting errors (Kogut, 2000; Lee & Cole, 2003). In most of the open source software development literature, error correction and criticism refer to the ability of the developers to: 1) review the source code that is submitted by other team developers, and 2) criticize and correct the code as needed (Lee & Cole, 2003). Lee and Cole (2003) observe that the peer review process meets a key condition for knowledge growth. The information technologies that are archived and publicly available allow developers to rapidly exchange knowledge, compare facts, search, and discuss changes (Lee & Cole, 2003).

3. Research Method

In this section, I discuss how I identified the context, selected the case, sampled the archival data, developed a content analysis schema, and analyzed the data with this schema.

3.1. Context and Case Selection

For this study, I selected an open source software development team in the Apache Software Foundation because Apache provides a revelatory setting for successful open source software development (Thomas & Hunt, 2004) and for successful open innovation. Secondly, Apache Software Foundation has managed to transfer the successful open innovation practices of the Apache Web server team to more than a hundred other open source software development teams. Understanding these open innovation practices are important for organizations for emulation purposes.

Because I wanted to identify the knowledge creation activities in innovative open source software development teams, I selected a highly successful and innovative team in the Apache Software Foundation based on informal conversations with Board Members of the Apache Software Foundation. These Board Members follow all Apache projects. Apache Lucene (referred to shortly as "Lucene") is a high-performance, full-featured text search engine library written in Java (http://lucene.apache.org) that's suitable for applications that require full-text search. Lucene provides Java-based indexing and search technology, spell checking, hit highlighting, advanced analysis, and tokenization capabilities.

Lucene exhibits both product and process innovation as explicated to me by the team's founder as follows. The team innovates at the product level by "using a very novel architecture by using a scalable indexing method, which permits incremental changes. Furthermore, by being object oriented, Lucene enables the developers to easily extend and customize its functionality". Furthermore:

> *Lucene is one of the earliest search systems using open source model of software development. This model enables our customers to easily build search into their applications without contracts and licensing fees, arguably also influencing the business model in the long run.*

The process Lucene used is an example of the "private-collective innovation model", in which participants "use their own resources to privately invest in creating a novel software code…and chose to freely reveal it as public good" as defined by von Hippel and von Krogh (2003, p. 214). Apache Lucene is used by more than 4,000 organizations worldwide. Due to the success of Lucene, in 2008, Lucene developers received a major startup funding, and a quarter of the developers founded the company Lucid Imagination based on the Lucene technology. The founder of Lucene acts as the company's advisor. The company's members still contribute to the development of the open source software. In 2010, Lucid Imagination became a finalist for the Red Herring 100 North America Award, which recognizes the 100 most innovative private technology companies, which are determined based on multiple criteria including technology innovation (Market Wire, 2010). This study focuses on the Apache Lucene team developers and their activities as reflected in the archival Apache Lucene data. I analyzed the activities of Lucid Imagination members only to the extent that they contributed to the Apache Lucene development, and I did not capture their activities outside of the Apache Lucene team (or the activities of non-Lucene members who contributed only to Lucid Imagination).

In identifying the case, I needed to find a team with sufficient interaction that enables the observation of knowledge creation and exchange. For that reason, I selected an active project with many members. Lucene's team has 7 developers and 132 users who develop the software and collaborate with each other on a daily basis. This level of interaction gave me the opportunity as a researcher to observe how they created knowledge. Furthermore, during the period of data collection, the team was at the development stage (Helfat & Peteraf, 2003) (as opposed to the founding or maturity stages), which means the team had organized itself with some stability in its membership, with clear norms and roles even though change in membership was still a possibility. The team had a stable version of the software and a common goal. Software development teams at the development stage present an active and dynamic software development environment, with much novel software development activity (i.e., open innovation activities, as defined by von Hippel and von Krogh (2003)) and much communication; they therefore provide the potential for a researcher to identify the knowledge creation activities.

3.2. Introduction of the Data

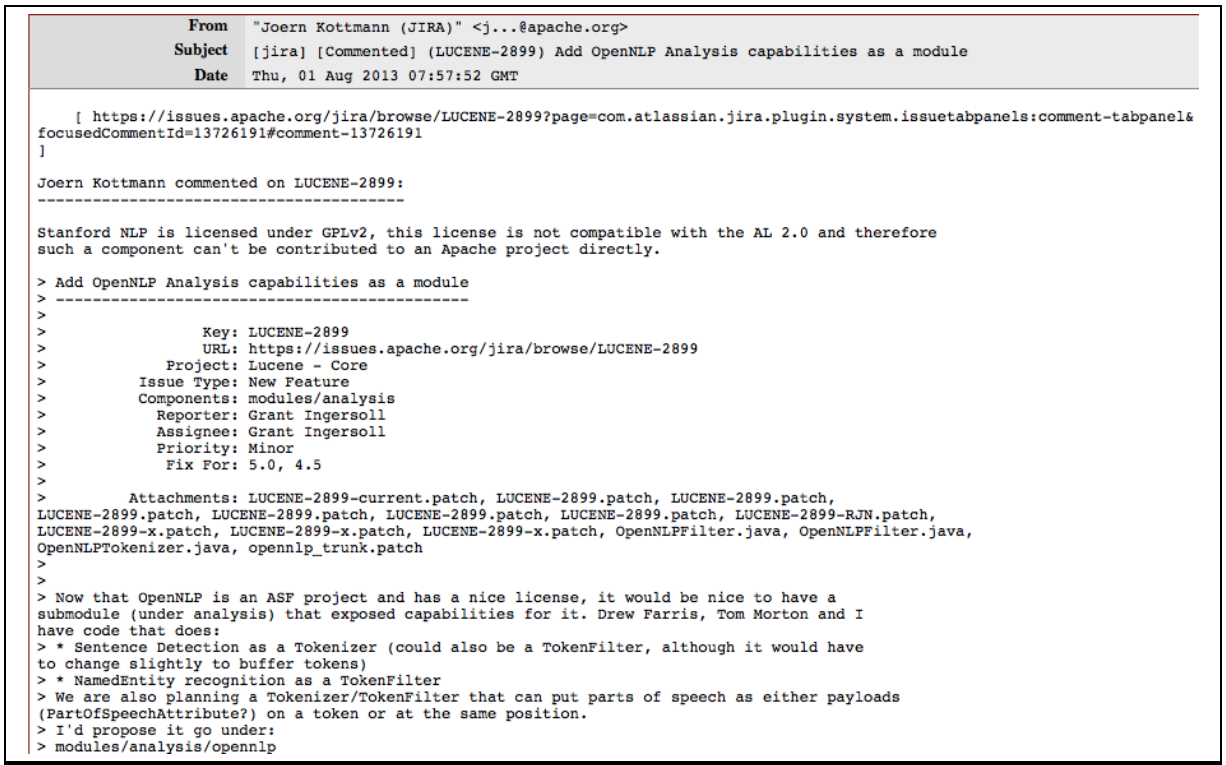
I used archival data to observe the developers' and users' knowledge-creation behaviors. For many organizational studies, archival data provide limited additional data to complement other main data sources. Yet, the situation is quite the opposite for open source software development teams: open source software developers communicate, coordinate their activities, and share their work mainly through information technologies. The archival data provide almost complete archive of behaviors of all team members (with the exception of bi-annual conference meetings) in the same way the team members exhibit them and in the same way the other team members observe them. Therefore, archival data enable an objective analysis of the full range of knowledge creation and exchange activities both among team members (developers) and between developers and users.

For identifying and coding knowledge, I used four sources of evidence from four information technologies: developers' email listserv (mailing list) for tracking knowledge exchange among the developers; users' email listserv for capturing the inflow and outflow of knowledge between the team and the individuals outside of the team (users); issue trackers, which capture the quality improvement and new ideas for innovation as Setia et al. (2012) mentions; and software version control tools that capture the contribution process to software development. I describe these four archival data sources next.

The developers' mailing list (developers' listserv) is the main communication forum for teams according to the Apache website. The discussions related to development activities and general strategic discussions that affect the teams take place on this forum.

Issue tracker (JIRA) was used to track software bugs and improvement plans. Analyzing the changes in the issue tracker was important because it captured teams' coordination of novel software development and related communication, thus providing valuable information about knowledge creation. Issue tracker stores the following information on each issue: the type of software issue or

problem, its URL, the project name, software version and the project component that the issue pertains to, the reporter of the issue, the person assigned to the task, and the issue's priority. Issue tracker allowed developers and users to type in their comments, to assign the issue to themselves or to others, and to change the status of the issue to "resolved". Developers often communicated and shared information using the issue tracker in a way that is similar to exchanging emails in the developers' listserv. The project team set JIRA issue tracker to forward changes in JIRA to developers' listservs in the form of a message with the same subject line. Thus, changes to the issue tracker could be observed in the form of JIRA message threads, which were similar to email threads. Figure 2 shows a sample JIRA message from an Apache team that was automatically sent to the listserv. This automatic forwarding of JIRA changes allowed project team members to follow software-related decisions and changes as soon as they occurred. For the purpose of this study, I referred to each instance of a JIRA issue as a "message" or a "JIRA message" and content analyzed it as I would an email.



From	"Joern Kottmann (JIRA)" <j...@apache.org>
Subject	[jira] [Commented] (LUCENE-2899) Add OpenNLP Analysis capabilities as a module
Date	Thu, 01 Aug 2013 07:57:52 GMT

[https://issues.apache.org/jira/browse/LUCENE-2899?page=com.atlassian.jira.plugin.system.issuetabpanels:comment-tabpanel&focusedCommentId=13726191#comment-13726191]

Joern Kottmann commented on LUCENE-2899:

Stanford NLP is licensed under GPLv2, this license is not compatible with the AL 2.0 and therefore such a component can't be contributed to an Apache project directly.

> Add OpenNLP Analysis capabilities as a module
>
> Key: LUCENE-2899
> URL: https://issues.apache.org/jira/browse/LUCENE-2899
> Project: Lucene - Core
> Issue Type: New Feature
> Components: modules/analysis
> Reporter: Grant Ingersoll
> Assignee: Grant Ingersoll
> Priority: Minor
> Fix For: 5.0, 4.5
>
> Attachments: LUCENE-2899-current.patch, LUCENE-2899.patch, LUCENE-2899.patch, LUCENE-2899.patch, LUCENE-2899.patch, LUCENE-2899.patch, LUCENE-2899.patch, LUCENE-2899-RJN.patch, LUCENE-2899-x.patch, LUCENE-2899-x.patch, LUCENE-2899-x.patch, OpenNLPFilter.java, OpenNLPFilter.java, OpenNLPTokenizer.java, opennlp_trunk.patch
>
>
> Now that OpenNLP is an ASF project and has a nice license, it would be nice to have a submodule (under analysis) that exposed capabilities for it. Drew Farris, Tom Morton and I have code that does:
> * Sentence Detection as a Tokenizer (could also be a TokenFilter, although it would have to change slightly to buffer tokens)
> * NamedEntity recognition as a TokenFilter
> We are also planning a Tokenizer/TokenFilter that can put parts of speech as either payloads (PartOfSpeechAttribute?) on a token or at the same position.
> I'd propose it go under:
> modules/analysis/opennlp

Figure 2. A Sample JIRA Issue, which Reflects a Comment on a Previous JIRA Bug Solution

A software version control system (Subversion; SVN) is the most reliable tool for tracking team members' software code contribution behaviors, a form of knowledge contribution (Morner & von Krogh, 2009). All changes to the software were made in SVN and it was possible to see the exact software code changed by each person. The Lucene team set up SVN to send an email to a separate listserv each time there was a change in the source code. This enabled all team members to follow changes to the software code immediately as they happen. Figure 3 presents a sample SVN message.

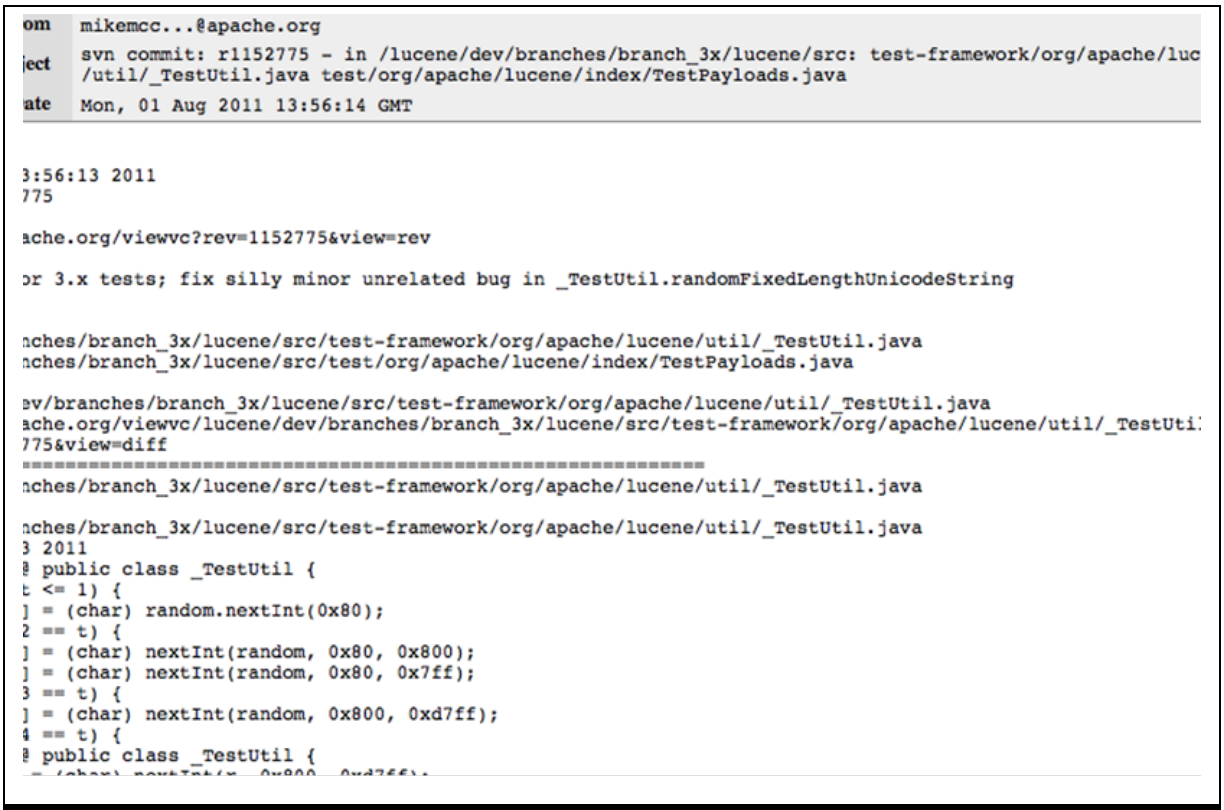


Figure 3. Sample SVN Message

A users' listserv is a mailing list that enables users to ask technical and other questions and get answers and support from core team members (developers). In this sense, users' listserv captured the communication between users and developers. As a result, users' listserv was an excellent data source for capturing knowledge inflows and outflows between the software developers that made up the team and the users, who were the beneficiaries and peripheral contributors to the team's work. While open source software developers also made great use of social media, their use typically signified their efforts to get to know each other at a personal level and to stay in touch as friends. Since the open source software developers did not use social media for developing software or communicating about their software development ideas and efforts, this data source was not relevant to this study on knowledge creation.

3.3. Archival Data Collection and Reduction: Sampling of Threads

The archival data (from two listservs, JIRA issue tracker and SVN) consisted of individual messages. To capture and meaningfully analyze the knowledge creation activities of open source software developers and users, I analyzed individual messages in the context of threads. Based on inspection of messages (emails, JIRA messages or SVN messages), I identified that the team was disciplined about keeping each thread on topic, and changing message subjects only when the subject is uniquely different. I removed single-email threads from analysis because they did not present opportunities for analyzing how knowledge creation activities unfold.

Instead of sampling the threads randomly over the life of the project, I collected the data from the time period just preceding the identification of Lucene as an innovative open source project because the innovativeness of the team may change over time. I based this identification on informal conversations with several Apache Software Foundation Board members, who followed all of the Apache projects and actively participate in a number of these projects. Based on these conversations, I identified that two types of contributions were relevant to knowledge creation: 1) members' ongoing regular contributions, and 2) their contributions to the crucial events (such as by fixing a major bug

that blocks the release of a software). The crucial events referred to highly memorable contributions (bug fixes, features, etc.) or events (such as a release or a decision) that the team members deemed to be highly important to the success of the project.

I identified and analyzed a total of 522 messages using the following data sampling method. To capture regular contributions, I used the following sample identification method. At the beginning, I analyzed five threads from each of the four different sources of data (developers' listserv, users' listserv, SVN, JIRA). When unique codes emerged that added to the theory, I added a new sample of five threads from each of the four data sources. After repeating this procedure four times, I determined that theoretical saturation was reached because adding new data did not contribute further to the theory (Morse, 2004). Therefore, I stopped adding new threads 20 threads per archival data type. This sampling resulted in a total of 80 regular contribution threads.

Table 2. Archival Data Analyzed (Total of 522 Messages)

	Regular contribution threads	Crucial event threads
Developers' email threads (communication)	20	3
Users' listserv email threads (communication)	20	7
JIRA threads (coordination)	20	12
SVN messages (work)	20	17
Total (522 messages)	**80**	**39**

I identified crucial events first by filtering the (JIRA) issue trackers for "major" issues (an important problem, which still allowed the users to use the software), "critical" (an important issue, which allowed a minor release, but not a major one), or "blocker" issues (an issue that blocked all releases). I selected crucial events only during the six months preceding the regular contribution threads in order to keep the collected data in the same time frame where I knew the team was innovative. A key informant member of the Lucene development team then reviewed and shortlisted these issues. Once I determined the crucial issues using the SVN system, I used the technical terms and the issue numbers and names to identify all relevant discussions on the developers' and users' mailing lists. Table 2 provides the resulting number of message threads. With this sampling strategy, I identified 522 messages (119 threads).

Table 3. Steps of Data Analysis

	Process and analysis	Outcome
1. Data reduction	1.1 Collection and sorting of archival data into threads. 1.2 Importing data into Atlas-Ti.	Data is ready to be analyzed with Atlas-Ti.
2. Data display and analysis	**Content analysis schema development** 2.1 Adaptation of the Nonaka and Takeuchi model to the open source team settings to develop content analysis schema. 2.2 Independent content analysis by two analysts to further develop the content analysis schema. 2.3 Establishment of content analysis schema reliability with sample data by two analysts. **Content Analysis** 2.4 Analysis of the 522 messages by the author based on the reliable coding schema. 2.5 Tabulation of frequency tables.	**Content analysis schema development outcome** Reliable coding schema is ready for use. **Content analysis outcome** All 522 messages are coded using the reliable content analysis schema.
3. Conclusion drawing	3.1 Written synthesis of the findings after sense making in light of the preliminary field study. 3.2 Comparing the findings with the extant literature.	Findings and discussion of the findings.

3.4. Content Analysis Schema Development & Content Analysis: Adapting the Knowledge Creation Framework to the Open Innovation Setting

Table 3 summarizes the steps of data analysis, which include data reduction, data display and analysis, and conclusion drawing (Miles & Huberman, 1994). Following the data reduction process, this section describes how I adapted Nonaka and Takeuchi's (1995) knowledge creation framework to the open innovation context.

In adapting the knowledge creation framework (Table 1) to the open innovation setting, I describe below the descriptions and examples given by Nonaka and Takeuchi (1995) as sources of deductive reasoning. These descriptions and examples helped me discern the intricate meaning of each element in the framework. The descriptions further helped us identify behaviors to support the same goal in the open innovation setting used in this study, thereby leading to the development of the coding schema. I describe the behaviors that make up the coding schema (see Table 4) for the identified open innovation setting below in the same way Nonaka and Takeuchi (1995) describe them in their book. I provide relevant examples in Section 3.4. Thus, I do not repeat them here.

3.4.1. Socialization

Nonaka and Takeuchi (1995) define socialization as a knowledge-creation mode, where tacit knowledge of one person becomes the tacit knowledge of another person through shared experience such as the one that happens in an apprentice-master model. They suggest that individuals can acquire technical skills directly from others by observation, imitation, and practice without using language. The authors give the example in which the staff of Matshushita Electric Industrial Company tried to mechanize the dough-kneading process. To gain the tacit knowledge of the master bakers, the head of software development volunteered to apprentice with the Osaka International Hotel's head baker. As a result of this socialization, she learnt that the baker not only stretches the dough but also twists it, a significant and previously non-described knowledge, which she then applied to the machine's design.

Table 4. Content Analysis Schema

Knowledge-creation mode	Knowledge creation-behavior	Description
Socialization (tacit to tacit)	Report bugs or improvement needs	Software bugs (problems) that are identified in JIRA issue tracker or logs of general requirements for the software to be improved.
	Submit patch	The patches (fixes to the software to improve the software functionality or fix a problem) that are submitted in either JIRA or developer's listserv.
	Commit patches	This code marks when the developers commit their patches or test (potentially change) and commit others' patches in the SVN.
Externalization (tacit to explicit)	Contribute to problem resolution by providing information	This code identifies the information that individuals provide in helping developers and users solve a problem they have during software development or software use. The information typically helps the individual troubleshoot or resolve their own issue. This may be information on how the software works that helps the others develop a better solution.
	Make suggestions and troubleshoot for the developers and users	This code captures the discussions on the developers' and users' listservs, where the individuals support the software development issues of developer or users by either troubleshooting the source of their problems or making clear suggestions on how to solve the issue.
	Explain logic behind suggestions	This code identifies discussions on the developers' listserv where a person provides extended explanations to justify their suggestions to the other members.
	Ask questions related to someone's suggestion	This code captures situations where an individual evaluates somebody's suggestion and asks probing questions to ensure that the individual who made the original suggestion has thought all relevant aspects through.
	Mentor and guide others	This code refers to situations where a person mentors others by teaching them the norms of working in the team, clarifying good software development approaches that are adopted by the team.
Combination (explicit to explicit)	Communicate issue resolution	This code refers to cases where individuals come back to JIRA to communicate how an issue is resolved. This provides closure to the issue to those who may not have followed the SVN.
	Communicate patch commit	This code refers to situations where individuals come back to JIRA and communicate that they have committed a patch and provide details about it. They may have developed the patch, or a user may develop it. Developers who commit user patches may provide details about it such as how they may have tested and/or updated the patch.
	Refer the users to other knowledge sources	This refers to cases where the developers share project-based knowledge with users. They may refer the users to earlier discussions, project webpage, wiki, or external resources about the project. They usually provide links or exact location of the information.

Table 4. Content Analysis Schema (cont.)

Knowledge-creation mode	Knowledge creation-behavior	Description
Internalization (explicit to tacit)	Document changes	This code refers to software documentation that is written to describe the change in the software or the use of the software. This documentation is written for developers or users of the software.
	Write tests	This code refers to the tests that are written in order to test the changes in the software.
	Develop webpages	This code refers to the documentation effort that includes developing or updating a webpage for the project.
	Contribute to wiki	This code refers to the documentation effort that includes developing or updating a wiki page for the project.

Socialization occurred in the open source context through users participating in an apprenticeship by reporting bugs and submitting patches to fix these bugs, and the developers testing, correcting (as needed), and committing these user patches (Ducheneaut, 2005). While the developers did not get a chance to see each other while developing software code, the transparencies provided by information technologies allowed the developers to inspect the changes that are made their team members.

Morner and von Krogh (2009) suggest that email and source code constitute data in the OSS team context. Here, data refers to observations coded into numbers, language, texts, or pictures (Willke, 2003). Yet, when email and source code data are integrated into a fitting context of experiences, they turn into knowledge (Willke, 2003). Based on this, when a user reads and classifies the content of email or software source code based on their existing knowledge, this user creates new tacit knowledge (Morner & von Krogh, 2009). Such sharing of tacit knowledge through shared artifacts such as software source code is hinted by early open source software development research (Haefliger & von Krogh, 2004). Individuals gain from others' tacit knowledge embedded in the source code by examining which algorithms others use and how they structure the source code (Morner & von Krogh, 2009). Morner and von Krogh (2009) explain that, due to the availability of the different releases of the software on the Internet, users can closely examine the source code's development by studying it at the different versions at different developmental stages. Moreover, software code does not make all of the knowledge (such as the vision of the developer, the general architecture, etc.) and the problem-solving mechanisms used by the developer immediately apparent (i.e., fully explicit) to any reader. In order to decipher the knowledge that is embedded in the source code, individuals need to have a certain level of intellectual engagement with the source code by questioning it, such as by asking: why is a certain problem-solving logic used in the source code instead of another possible method? How does this piece of code relate to the rest of the code (other modules, the general architecture)? What does the code developer aim at with this code? A more specific potential question in the context of a search engine, such as Lucene, may be: how did the developer make the search algorithm work faster with this code? In this sense, the code developed by a software developer may be likened to the move of a chess player. While the move/solution seems visible (explicit) to the outsider, the thought process preceding the move, the future moves that this move will enable, and the other alternatives that were considered and rejected by the player are not made explicit. Thus, observing the moves of a chess-master and learning from (i.e., transferring the tacit knowledge of) the chess-master requires an observer's intense intellectual engagement with the move.

Bringing the example back to the software development context, as a result of the process of users interacting with the software code (explicit knowledge), the "explicit knowledge can 'convey' tacit knowledge despite the fact that the tacit knowledge has never been shared and the different participants have never met in person" (Morner & von Krogh, 2009, p. 443). While the software code is explicit knowledge (Haefliger, von Krogh, & Spaeth, 2008; Morner & von Krogh, 2009), as explained previously, "the way it is structured and built conveys more tacit knowledge" (Morner & von Krogh,

2009, p. 443). The idea that an artifact of explicit knowledge can relay tacit knowledge could be understood better by seeing explicit and tacit knowledge not as "two separate entities, but rather as mutually complementary and based on the same continuum" (Nonaka & von Krogh, 2009, p. 640). Jha (2002) suggests that these two forms of knowledge are not competing but oscillating on a continuum to mutually enhance each other. According to Nonaka and von Krogh (2009), while explicit knowledge is being shared among the individuals, the knowledge gets internalized by losing its explicitness and increasing its tacitness. This happens as the acquirers of the knowledge act on the knowledge through action, practice, and reflection. Thus, activities that require an individual's intellectual engagement with the software bring about knowledge transfer: the tacit knowledge embedded in the software gets transferred to the minds of the individuals who are engaged with the software. The first level of intellectual engagement with the software (also the first step of socialization of the users towards being developers) is testing the software and reporting bugs (Ducheneaut, 2005; von Krogh et al., 2003). When a version of the software is bundled, this version is tested shortly before and after the release and the bugs are reported. The bug-reporting process represents the use and testing of the software, which is a process of engaging with the software that brings about learning from the developers through the boundary object, which is the software. Again, this learning happens through the apprentice's engagement with the software code, rather than any explicit explanation by the developer. Therefore, this learning represents a tacit-to-tacit knowledge transfer. Bug reporting is followed by submitting patches, which requires the individuals to learn the tacit knowledge embedded in the software and codify their own, which, according to Cowan, David, & Foray (2000), is a knowledge-creation form that, at the same time, enhances the actors' knowledge. The practice of identifying technical problems (i.e., bugs) and developing software code to resolve it are two key behaviors of socialization because they are central to successfully joining an OSS project (Ducheneaut, 2005; von Krogh et al., 2003). In Lucene, the developers with access to the source code directly test and commit their own patches (i.e., they incorporate their changes to the source code). Yet, the users do not have the commit-rights; therefore, they submit their patches, which are then tested, altered as needed, and committed by a developer. In this process, not only does the developer learn from the tacit knowledge of the user by their inspection of the patch, but the user gains additional knowledge from the developer's feedback on the user's work. The learning of the users and the transfer of tacit knowledge from developers to users as the developers test and correct user patches can be likened to students learning from their corrected assignment papers (therefore using socialization to transfer a master's tacit knowledge).

3.4.2. Externalization

Nonaka and Takeuchi (1995) define externalization as a process of articulating tacit knowledge into explicit concepts, and they suggest that writing is an act of converting tacit knowledge into articulable knowledge. They suggest that the externalization mode of knowledge creation is typically triggered by dialogue and collective reflection. As an example to this process, they point out to how Mazda and Honda created the right design concepts as part of their new car design/development process. Nonaka and Takeuchi (1995) point out to the use of metaphors and analogies as tools to trigger shared understandings, dialogue, and collective reflection. I went through six months of archival data for both regular and critical events, yet I did not observe metaphors or analogies in this setting. In the software development process, the externalization of tacit knowledge becomes very clear in the problem-resolution process, where individuals share the inner workings of the software as inputs to other people's decision making. Hemetsberger and Reinhardt (2006) identify conceptualizing problems and new ideas an important knowledge-creation process in open source software development teams. Based on their investigation of the KDE open source software development project, they found that this kind of knowledge creation occurred mostly on the project team's mailing lists, where the members explained, evaluated, rejected, corrected, or defended an opinion (Hemetsberger & Reinhardt, 2006). This type of knowledge creation identified by Hemetsberger and Reinhardt (2006) meets the requirements of Nonaka and Takeuchi's (1995) externalization knowledge-creation mode, which refers to creating shared understandings, dialogue, and collective action by explicating tacit knowledge. As part of the content analysis schema development effort, I identified the following externalization behaviors, which are similar to those mentioned by Hemetsberger and Reinhardt (2006).

When developers had issues that they could not resolve, they brought these to the developers' mailing lists, which resulted in three types of knowledge exchange. First, the developers provided relevant information, such as by explaining how the existing algorithm worked, and thereby explicated the tacit knowledge embedded in the software code to enable the developers to solve their own issues. Second, individuals helped troubleshoot these problems and made very specific suggestions, and hence shared their tacit knowledge on how to solve such problems. Furthermore, in an effort to justify their suggestion, they also explained their logic and thought processes, and thereby explicated their knowledge further. Tuertscher, Garud, & Kumaraswamy (forthcoming) observed the same justification behavior in their study of the development of Atlas, a complex technological system developed at the European Organization for Nuclear Research (CERN). They found that participants provided this kind of justification each time they proposed different technological options. These explanations resulted in the creation of interlaced knowledge, a partial overlap of knowledge across team members (Tuertscher et al., forthcoming). The evaluation process mentioned by Hemetsberger and Reinhardt (2006) happened at Lucene after developers or users made suggestions. Each suggestion was followed by questions from others, such as "Have you considered this other option?" or "How does your solution impact (another part of the code)?." These questions ensured that the individual who came up with the suggestion thought through important dependencies in the software.

This externalization process enables both the participants and the observers to learn. Observers and participants of this troubleshooting process learn from others' explicated knowledge on how to troubleshoot their own problems, and they improve their understanding of how the software is built.

Lastly the developers mentor and guide others on how to more effectively participate in the project, and therefore explicate their knowledge about the (software development) team and implicit norms on how they work together.

3.4.3. Combination

Combination is the process of organizing concepts into a knowledge system (Nonaka & Takeuchi, 1995). Nonaka and Takeuchi (1995) describe how knowledge is exchanged through documents, meetings, and conversations (such as by phone, using information technologies, etc.). They suggest that reconfiguring existing information through sorting, adding, combining, and categorizing explicit knowledge may lead to the creation of new knowledge. The open source software developers manifested knowledge combination in two ways. First, they referred the users to knowledge sources that resided elsewhere, and thereby combined and systemized knowledge in one location for other readers. Secondly, the developers close the loop on previous discussions on an issue or a patch by communicating how an issue was resolved, and how a patch was committed including the description of the changes in the patch. This information was already available to the careful follower of all mailing lists and all software commits. However, this communication (responding to an earlier JIRA issue that is open on the subject matter) systemized the knowledge, and created a database that can be queried. For example, one could do a search in JIRA (issue tracker) for all the issues that may block a new release of the software.

3.4.4. Internalization

Internalization is the converting of explicit knowledge into tacit knowledge through "learning by doing" (Nonaka and Takeuchi, 1995). Documentation (i.e., the process of verbalizing and/or diagramming knowledge into documents and manuals) is the main means of internalization by these researchers. According to Nonaka and Takeuchi (1995, p. 69) "documentation helps individuals internalize what they experienced, thus enriching their tacit knowledge". Indeed, writing to learn (i.e., enriching one's tacit knowledge) is not a new concept. Writing is used in academia both in earlier formative years of education, and later on specifically in software engineering education. Writing/documenting enables the writer/documenter to "recall, clarify and question what they know about a subject and what they still wonder about with regard to that subject matter" (Knipper, 2006, p. 462). Teachers use writing as a tool for students to learn because writing "engages students, extends thinking, deepens understanding, and energizes the meaning-making process" (Knipper, 2006, p. 462). Indeed, "considering a topic and then writing about it requires deeper processing than reading alone entails" (Fordham, Wellman, & Sandman, 2002, p. 151). In software development, knowledge construction involves testing and documentation

(Kautz & Thaysen, 2001) to the extent that proponents suggest their integration into the computer science and software engineering curricula (Jansen & Saiedian, 2006). Documentation required individuals to experiment and engage with the system, to recall what they knew about it, to play with the system to learn more about it, and to test what they knew. Writing and/or executing test scripts were also tasks that enabled individuals to further play with the system and learn about the procedures embedded in the system. To sum up, documentation brought about a deeper level of engagement with the subject matter that enabled meaning making, and hence created tacit knowledge in the minds of the documenters. Furthermore, "documents and manuals facilitate the transfer of explicit knowledge to other people, thereby, helping them experience the experiences of others indirectly (i.e., "re-experience" them)" (Nonaka and Takeuchi, 1995, p. 69). Taking these literature streams as a basis, my content analysis schema development effort pointed to four types of internalization. These types included writing documentation, developing tests for the software that can be used to understand and test the software, developing webpages, and developing wiki pages for the project.

Thus far, I have discussed the content analysis schema development by adapting the knowledge creation framework to open source software development (Step 2.1 in Table 3). Prior to data analysis, along with an independent analyst, I established the reliability of the content analysis schema. After I discussed the general framework, we (the analyst and I) coded 10 threads together to create shared understandings on the content analysis schema and approach. Subsequently, we independently coded and compared a second sample of 10 threads; we resolved inconsistencies of our coding through discussion. Each time we drew a new sample of 10 threads and independently coded until we reached an agreement rate of 86 percent on the coding schema, which is considered very good according general coding standards (Neuendorf, 2002). Through this process, we established the content analysis schema reliability (Step 2.3 in Table 3), after which I coded the previously identified 522 messages using the reliable content analysis schema. Lastly, I drew conclusions about the observed codes based on calculating frequencies and contrasting findings with the literature.

4. Findings

In this section, I answer the two research questions. In Sections 4.1 and 4.2, I identify the type of knowledge-creation modes and behaviors used (RQ1a) and how developers and users contributed to knowledge creation (research question 1b) in order to answer the RQ1: “How do innovative open source software development teams create knowledge?”. In Section 4.3, I identify the type of information technologies used for knowledge creation.

4.1. Knowledge-Creation Behaviors in Open Innovation Context

The coding identified 401 instances of knowledge-creation behaviors. Table 5 presents the knowledge-creation behaviors for each of the knowledge-creation modes based on this coding. I discuss these findings below. I identified fifteen types of knowledge-creation behaviors that fell into the given knowledge-creation modes (Table 5) of socialization (three behaviors), externalization (five behaviors), combination (three behaviors), and internalization (four behaviors).

Table 5. Knowledge-Creation Modes and Behaviors in Open Source Software	
Knowledge-creation mode	**Knowledge-creation behavior**
Socialization (tacit to tacit)	Report bugs
	Submit patch
	Commit patches
Externalization (tacit to explicit)	Contribute to problem resolution by providing information
	Make suggestions and troubleshoot for the developers and users
	Explain logic behind suggestions
	Ask questions related to someone's suggestion
	Mentor and guide others
Combination (explicit to explicit)	Communicate issue resolution
	Communicate patch commit
	Refer the users to other knowledge sources
Internalization (explicit to tacit)	Document changes
	Write tests
	Develop webpages
	Contribute to wiki

Table 6 presents these four knowledge-creation modes and the frequency of their use based on the archival data analysis.

Table 6. Four Knowledge-Creation Modes and the Frequency of Their Use		
Knowledge-creation mode	**Frequency of observation in the archival data**	**Percentage of all observations**
Socialization	94	23%
Externalization	227	57%
Combination	47	12%
Internalization	33	8%
Total	**401**	**100%**

In Section 3.4, I describe each of these six knowledge-creation modes along with how they apply to the open source software development team setting by identifying the behaviors through which the knowledge-creation modes manifested.

4.1.1. Socialization[3]

Socialization was the second most frequently used mode of knowledge creation with 23 percent of the coded knowledge-creation behaviors falling into this category. Socialization has originally been defined as a way of transferring tacit knowledge through dialogue, observation, imitation, or guidance. Reporting bugs and later on submitting patches were some of the first steps of socialization for the users. Hence, I identified that socialization at open source software development teams manifested through the key software development tasks of reporting bugs (problems in the software code),

[3] This section only presents findings. Readers who may have questions on why the specific behaviors constitute knowledge-creation behaviors should refer back to Section 3.4.1, which discusses why socialization behaviors are knowledge-creation behaviors and how they are operationalized in this context.

submitting patches, and committing patches. While only the developers with commit rights were able to commit software patches, all three software development tasks indicated knowledge creation based on the individuals' tacit knowledge. These actions created opportunities for others to observe and learn from. Bug reports and patches cannot immediately transfer explicit knowledge. To learn from them (i.e., to transfer tacit knowledge), developers had to engage in sense making by inspecting the software code and understanding the changes and the knowledge behind the changes.

4.1.2. Externalization[4]

Externalization was the most often used knowledge-creation mode identified in the data: 57 percent of all observed knowledge-creation behaviors in the archival data fell into this category. This indicated that the externalization of tacit knowledge to explicit knowledge made up more than half of the knowledge-creation behaviors of the Lucene team. Externalization occurred mostly in either the problem-resolution process, or when individuals supported others, such as the team members and the users of the software.

The problem resolution process in the software development appeared complex, with many variables to consider. In the email exchanges among the members, many ways of accomplishing a task were often discussed, and the team tried to determine the best way to execute a particular task. For instance, in an effort to resolve a bug, members made suggestions as to which approach to use by saying, for example, “Should we deprecate this method?”. When they made suggestions, they often clearly explained the logic behind their suggestions, such as by saying, “This makes the archetype simpler, because now it’s just doing configuration and reporting on the test modules.” Others contributed to the problem resolution by providing information, such as other aspects to consider, or by sharing the details of how the software code was originally developed as we can see in the following example:

> *The code right now takes two paths: if you pass a field, it just calls QP via super. If you don't pass a field, it loops through each of the fields it has, which I suspect is usually what one wants. But in this loop, it also calls super, which means you don't get a chance in your derived class to override it again. I think it would make more sense to not call super in the loop case. So derived classes can get access to the call both when no field is passed and when each field is passed.*

This problem-resolution process of making a suggestion and explaining the logic underlying it led other developers to provide either information or more suggestions. In trying to justify that their suggestions were appropriate ways of solving a problem, individuals explicated their tacit knowledge, and therefore transferred their knowledge to others. In turn, this knowledge transfer enabled others to come up with more suggestions by creatively building on new knowledge and thus creating opportunities for enrichment. This problem resolution process thus enabled the team to possibly find better and more innovative solutions to a given problem.

The archival data provided evidence for how developers were mentoring others both by helping them with the software-based knowledge and by showing them how to contribute to the team in general. For instance, archival data showed how one developer mentored a new member who was trying to submit new code and had problems during the process by telling to the member:

> *You have sent a lot of code—but that doesn't mean it's clear what you are doing, or what you expect—particularly since we can't see your data. If you are still having trouble, then you can send a self-contained example of the problem that anyone can run (JUNit test case) so that we can help you further.*

[4] This section only presents findings. Readers who may have questions on why the specific behaviors constitute knowledge-creation behaviors should refer back to Section 3.4.2, which discusses why externalization behaviors are knowledge-creation behaviors and how they are operationalized in this context.

Lastly, the data indicated how developers externalized their tacit knowledge during user support: developers troubleshooted, solved user problems, and provided information to enable users to troubleshoot their own problems. With these behaviors, developers codified their tacit knowledge and made it clear and explicit to the users and other readers of the messages. In this setting, outsiders also made contributions, and users also supported other users or contributed to the problem-solving process, and therefore made untapped tacit knowledge explicit and available for all to learn from.

4.1.3. Combination[5]

Combination refers to sorting, adding, and categorizing existing information that is already made explicit. Twelve percent of the observed knowledge-creation modes fell into this category. Individuals communicated issue resolution or patch commits in JIRA, and thereby created redundancies (in JIRA) with the information that was already available in SVN for code commits. These redundancies enabled all team members and external parties to see the resolution of the issue in one location (JIRA), which eliminated their need to search for it.

Lastly, combination activities occurred in the form of individuals providing either information about the project or external knowledge sources, such as websites, or concepts for the users. While these behaviors were conducted in an effort to share relevant knowledge with the users that they needed at the time, they also provided the side-benefit of keeping information resources related to a topic at one place for others to also benefit from.

4.1.4. Internalization[6]

Internalization refers to individuals converting explicit knowledge into tacit knowledge by means of reflection, which happens while they create documents, manuals, or stories based on new knowledge. Writing documentation, such as documenting changes, writing tests, developing webpages and wikis, were the ways that some of this team's members internalized others' knowledge. These documents and tests, when read and used by others, also enabled these third parties to internalize knowledge. About 8 percent of the knowledge-creation behaviors fell into this category. Internalization occurred through contributions to websites and wikis, which were all submitted through SVN.

[5] This section only presents findings. Readers, who may have questions on why the specific behaviors constitute knowledge-creation behaviors, should refer to Section 3.4.3, which discusses why combination behaviors are knowledge-creation behaviors and how they are operationalized in this context.

[6] This section only presents findings. Readers, who may have questions on why the specific behaviors constitute knowledge creation-behaviors, should refer to Section 3.4.4, which discusses why internalization behaviors are knowledge-creation behaviors and how they are operationalized in this context.

4.2. Developers' and Users' Contributions to Knowledge Creation

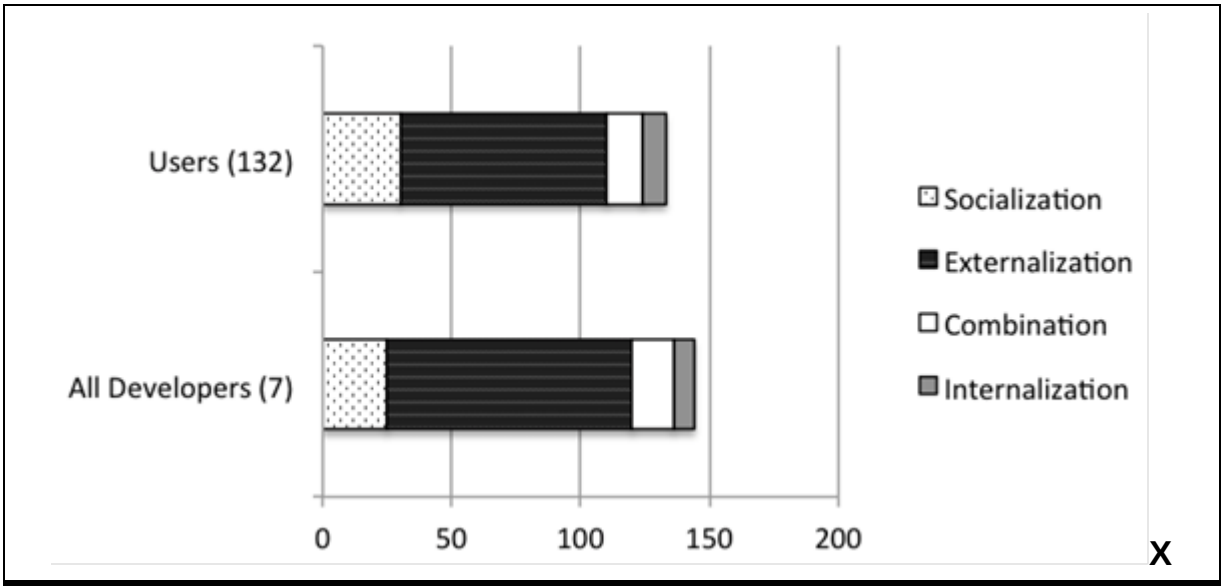


Figure 4. The Number of Knowledge-Creation Behaviors By Developers & Users During the Regular Period (X-Axis)

This section describes the findings on how developers and users contributed to the knowledge based on the regular contributions followed by the contributions of the developers and users during crucial events. Seven developers and 132 users participated in knowledge creation. Analysis of regular events showed that. while developers made up only 5 percent of all participants, they contributed 52 percent of all regular knowledge-creation activities (Figure 4).

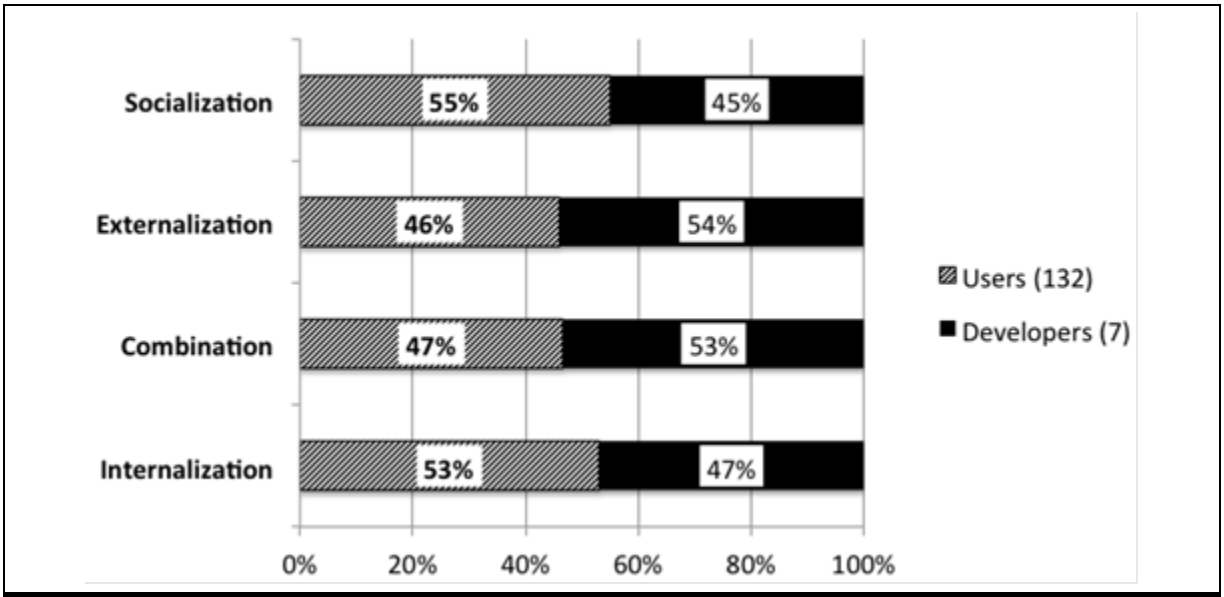


Figure 5. The Percentage of Overall Contribution to Knowledge Creation By Developers & Users During the Regular Periods (X-Axis)

While each user contributed very little to knowledge creation, when all users' knowledge-creation behaviors were combined, this amounted to knowledge creation that was roughly (quantitatively)

equal to that provided by the developers. Archival data analysis showed 144 instances (52%) of knowledge creation by developers and 133 instances (48%) by the users. This shows that, while each user created little new knowledge, the sheer number of users made a difference in the amount of new knowledge created.

Figure 5 shows how the team's knowledge-creation behaviors were split between the developers and users during the regular period. The developers and users created roughly equal amounts of knowledge. Developers provided 45 percent of socialization behaviors (25 instances out of 36), 53 percent of combination behaviors (16 instances out of 30), and 47 percent of internalization behaviors (8 instances out of 17).

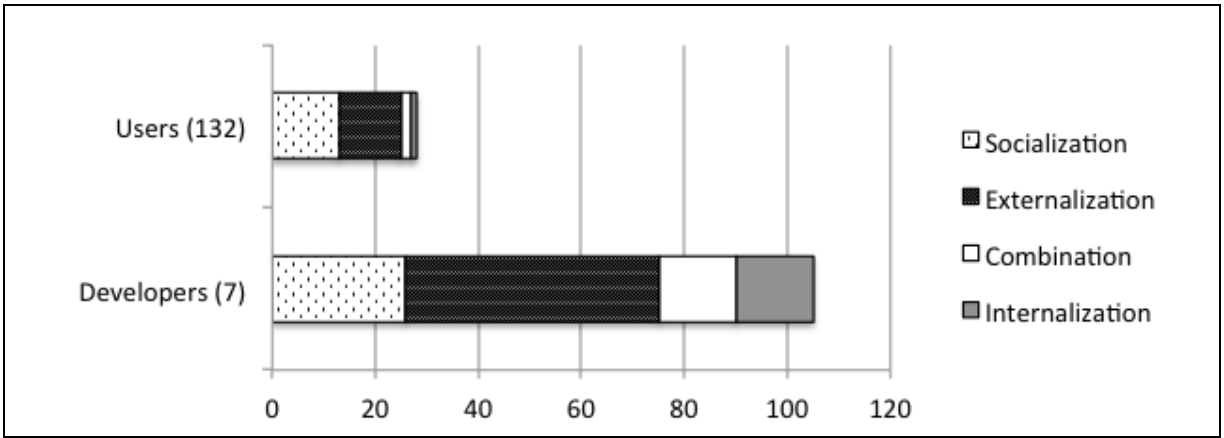


Figure 6. The Number of Knowledge-Creation Behaviors By Developers & Users During the Critical Events (X-Axis)

Figure 6 provides a dramatic contrast to Figure 4. While in regular events, users and developers created roughly equal amounts of knowledge, in critical events, users created only 21 percent of the new knowledge. Users contributed only 33 percent of all socialization, 20 percent of externalization, 12 percent of combination, and 6 percent of internalization knowledge-creation behaviors (Figure 7) during the critical events. This difference in activity of developers and users may be explained by the challenging nature of the critical events. While in the regular events many users could contribute to all aspects of the knowledge creation, the number of users who could contribute to critical events may have been limited by the knowledge and expertise required. In the few instances, users who had expertise in the key issues contributed with bug identification and patches (therefore contributing to socialization) and by making inputs to problem resolution process (thereby contributing to externalization). Then, the developers followed up with testing and committing user patches, documenting these issues, and communicating the resolution in JIRA (combination). Thus, even though users contributed to 33 percent of all socialization, their contribution to other modes of knowledge creation was much fewer for the critical events.

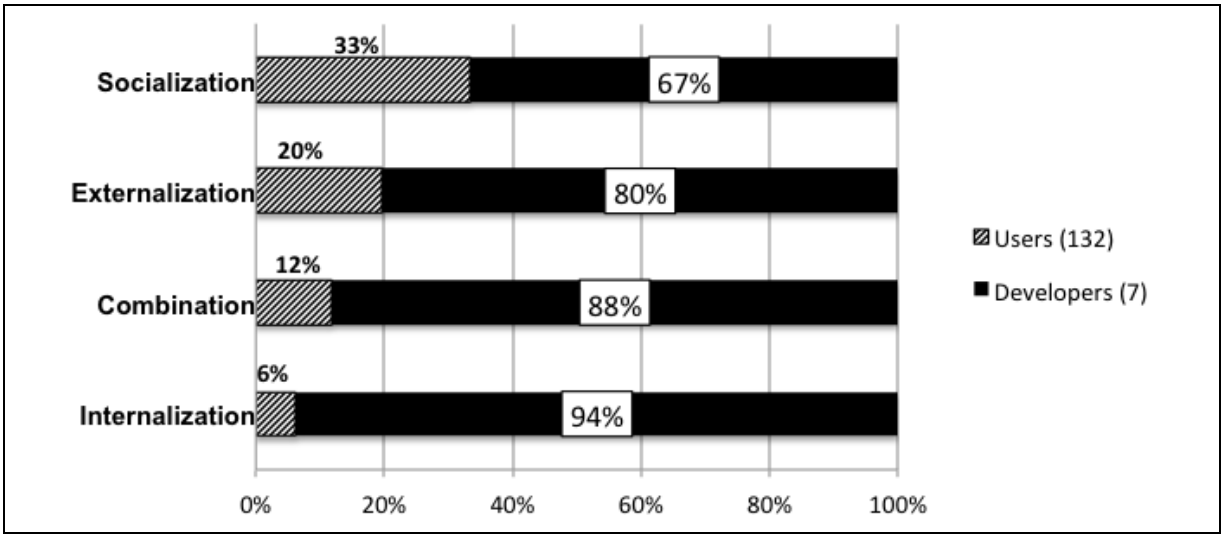


Figure 7. The Percentage of Overall Contribution to Knowledge-Creation Behaviors By Developers & Users During Critical Events

4.3. Information Technologies Used to Enable Knowledge Inflows and Outflows

In answering the second research question, I identified six types of information technologies for knowledge creation. Table 7 presents the frequency with which these technologies were used for knowledge-creation purposes. The developers' listserv and JIRA were the most frequently used tools for knowledge creation, making up 32.5 percent (133 instances out of 409 instances of knowledge creation) and 31.3 percent (128 instances) of the overall technology use, respectively. Users' listserv and SVN follow these information technologies in terms of frequency of use with 20.5 percent (84 Instances) and 13.7 percent (56 instances). Wikis and websites, which are keys to knowledge management in some companies, were used very little for knowledge creation at 1.5 percent (6 instances) and 0.5 percent (2 instances), respectively.

Table 7. The Frequency of Information & Communication Technologies Use for Knowledge Creation in Open Source Software

Information & communication technologies	Usage (%) for knowledge creation
Developers' listserv	32.5% (133 out of 409 instances)
JIRA	31.3% (128 out of 409 instances)
Users' listserv	20.5% (84 out of 409 instances)
SVN	13.7% (56 out of 409 instances)
Wiki	1.5% (6 out of 409 instances)
Website	0.5% (2 out of 409 instances)
Total	**100.0%**

Next, I discuss the use of information technologies in enabling each knowledge-creation mode. Table 8 shows the distribution of these knowledge-creation behaviors among various information technologies. Each knowledge-creation mode was enabled with at least two different information technologies. Most of knowledge-creation modes depended on one technology more heavily than the others.

Table 8. The Distribution of Information & Communication Technology Use For Each Knowledge-Creation Mode

Knowledge-creation modes	Information & communication technologies used for expressing the knowledge-creation behaviors						
	Developers' listserv	Users' listserv	JIRA	SVN	Website	Wiki	Total
Socialization			76%	24%			100%
Externalization	58%	29%	13%				100%
Combination		40%	60%				100%
Internalization				100%	6%	18%	124%(*)

(*) The total of internalization row is more than 100% because, while all internalization knowledge-creation modes used SVN, two knowledge-creation behaviors required the use of websites or wikis in addition to SVN.

For instance, socialization and combination were most dominantly enabled by JIRA, externalization by developers' listserv, and internalization by SVN. Internalization provided a small exception in that the team made all changes to all documentation (including websites and wikis) through SVN so that changes to these were traceable. Most of the socialization-based knowledge creation was done with JIRA (76%) followed by SVN (24%). The socialization mode of knowledge creation included individuals using information technologies to contribute to the work of software development. This created opportunities for others to benefit from the tacit knowledge that went into this work. Because everyone used JIRA for reporting and updating status of code changes due to bug fixing and feature adding activities, it allowed the team members to gain knowledge on the status of the development.

The developers used the concurrent versioning system (SVN) to submit code. SVN enabled multiple individuals to work on same or different parts of the code at the same time and commit software changes. SVN and JIRA together allowed individuals to work on software development, a complex task with many interdependencies, in the most independent way possible.

The externalization mode of knowledge creation depended mostly on listservs (namely, the developers' listserv (58%), the users' listserv (29%), and to a smaller extent JIRA (13%)) to explicate and communicate the developers' tacit information for the other developers and users. This indicates that communication through listservs created a major outlet for converting tacit knowledge to external knowledge. Some of this knowledge got used for software development at the socialization knowledge-creation mode; hence, individuals communicated the relevant information later through JIRA or SVN.

The combination mode of knowledge creation mainly depended on the issue tracker JIRA (60%), which was open to use by both team members and users. Part of this information was also available to those who follow the SVN to see which patches were committed. However, SVN provided access to only tacit information, which required sense making to learn from and may have been difficult for those who do not have the knowledge to make sense of a given tacit knowledge. While the SVN provided tacit knowledge transfer, individuals still logged the changes in the SVN and the details of the changes to JIRA, which created more opportunities for combining the different knowledge that had previously been externalized to be combined and shared with others in an organized manner. The developers' dependence on users' listservs in 40 percent of the combination mode of knowledge indicated a more ad-hoc way of knowledge creation. This was less systematic, and, while searching the whole listserv was always a possibility, the knowledge shared may have been more difficult to find later on.

The internalization mainly happened through documentation and all changes to documentation were controlled through SVN. However, the documentation itself already utilized different types of information technologies such as web technologies and wikis.

5. Discussion

This paper bridges the literatures on open innovation and knowledge creation, and contributes to open innovation theory by opening the black box of how knowledge creation occurs in innovative open source software development teams. Specifically, this study addresses: 1) the knowledge-creation modes, activities, and behaviors in open source software development teams; 2) how developers and users contribute to knowledge creation; and 3) how various information technologies are used to enable knowledge creation. In Sections 5.1 to 5.3, I discuss the theoretical and practical contributions of these findings.

5.1. Contributions to Theory

As I discuss in Section 1, various organizations increasingly want to emulate the open source software development teams' innovation model. However, emulation requires the understanding of the types of knowledge creation and how various participants contribute to knowledge creation as recommended by Agerfalk and Fitzgerald (2008). Earlier research suggests that the users contribute specifically to software quality improvement by generating patches and reporting bugs (Setia et al., 2012). This research also shows the existence of such contributions in the form of the socialization knowledge-creation mode. Indeed, the 132 users combined contribute 46 percent of the knowledge created through socialization. This suggests that each users' small contributions to knowledge contributions matter when the number of users is large because the total contribution adds up to be significant. In this sense, this study's findings support researchers' recommendation to engage and harvest outside contributors' external knowledge (von Krogh et al., 2003; West & O'Mahony, 2005). However, this study identified the importance of having few high-quality developers (rather than large numbers of users) in ensuring continuity of the project, a finding that differs from past research that does not differentiate the quality and complexity of contributions. In the face of critical events (which tend to include complex and challenging tasks), users' level of knowledge contribution is much lower. This finding also suggests a potential difference in the contribution quality by users and developers, which should be further analyzed.

Secondly, this paper contributes to knowledge management theory by adapting Nonaka and Takeuchi's (1995) knowledge creation framework to the open source software development setting. Table 4 provides the results of this adaptation, which other researchers can use as a content analysis schema in future research. While Nonaka and Takeuchi's (1995) framework provides a general understanding of the four types of knowledge creation, the content analysis schema provided here enables the operationalization of these concepts to open innovation settings, which is a uniquely different setting than Nonaka and Takeuchi studied. Despite the abundance of literature that follows Nonaka and Takeuchi, actual empirical operationalization of their knowledge-creation modes is rather rare (e.g., Nonaka, Byosiere, Borucki, & Konno, 1994), which is perhaps due to the difficulty of extrapolating the knowledge-creation modes to other contexts. This research did just that.

In the IT-enabled open innovation setting, three of the four knowledge-creation modes identified by Nonaka and Takeuchi manifested through different behaviors than those observed in the organizational setting. Researchers have previously found face-to-face interaction and physical activities as key to sharing tacit knowledge (Nonaka, Reinmoeller, & Senoo, 2000). Yet, this study identified that intellectual engagement with knowledge-embedded information technology artifacts such as software code enabled such knowledge transfer without requiring face-to-face interaction. Thus, tacit-knowledge embedded IT artifacts resides at the core of socialization in the IT-enabled open innovation settings.

Nonaka and Takeuchi (1995) found that innovative Japanese companies they studied used metaphors and analogies to develop shared understandings; Hemetsberger and Reinhardt (2006) also observed these metaphors and analogies in the KDE project. Despite my specific search for metaphors and analogies, I did not identify them in this study as elements explaining the organizational-knowledge creation. This is also in line with a previous study by Nonaka, Byosiere, Borucki, and Konno (1994), who studied 105 managers in Japanese firms. Instead of metaphors and analogies, the data from this study suggest that the developers used a very straightforward way of

explicating their knowledge through clear suggestions and justifications. Especially during the process of supporting developers and users in their development and use, the explication of tacit knowledge by participants became clearer. My study shows that metaphors and analogies were not the only means to sharing understandings. The developers used an iterative approach to explicate their tacit knowledge using a combination of questions, troubleshooting, suggestions, and justifications as part of problem resolution and software development. We can hypothesize that information technologies both require and enable the use of clear messages rather than messages subsumed in metaphors and analogies. This is especially clear in the case of mentorship behaviors that explicate norms, which are tacit knowledge in the form of implicit standards and values (Schein, 1992).

Lastly, the combination behaviors manifested in the open source software development contexts as behaviors aiming at better communication and coordination. The intentions of the developers were not to combine different types of knowledge. When individuals came back to JIRA and provided the information on the work they have just done, they made JIRA a queryable resource. Moreover, because JIRA was always up to date, participants could also follow which tasks were open, so they could participate more easily by picking an open task.

Among the four knowledge-creation modes, internalization was the one that was operationalized in exactly the same way in both innovative companies and IT-enabled open innovation setting.

5.2. Contributions to Practice

This paper contributes to practice by explicating how innovative open source software development team members create knowledge. For information systems managers and practitioners who incorporate open source software development teams into their organizational practice, this study suggests that they need to: 1) support existing users and encourage them to stay involved to keep the number of external knowledge contributors high, and 2) keep a small number of highly skilled developers who can deal with critical events and, at the same time, support the external users. These points may inform which skills organizations focus on when recruiting and training team members (such as written communication skills, ability to logically justify suggestions, problem resolution skills, and ability to work under pressure).

Setia et al. (2012) suggest the importance of information technologies in facilitating effective contributions from peripheral users. Practitioners can benefit from how the innovative team described in this case used information technologies to enable users and developers to independently contribute despite the many interdependencies presented by the complex software development task. In selecting the information technologies to support their company, practitioners should invest in multiple technologies. In creating an information technology infrastructure for open innovation, managers should set up IT infrastructures for open innovation, which enable each of the four types of knowledge-exchange models. Our recommendation is to allow the open innovation participants to have an input on how new information technologies can be adopted to support a variety of processes they use for innovation. The identified information technologies further may need to be tailored over time as practices may change.

Furthermore, this study adapted the four modes of knowledge creation to the open innovation team setting: 1) socialization, 2) externalization, 3) combination, and 4) internalization. In addition, this study explicated how information and communication technologies support each knowledge-creation mode. The transparency provided by using information technologies may change the nature of knowledge creation: these technologies enable all users, developers, and non-participants to observe and learn from the tacit and explicit knowledge created in the team, which thus creates opportunities for the socialization mode of knowledge creation for all observers even without face-to-face interaction. In comparison, in many settings that lack such transparencies, knowledge creation through socialization is limited to the few individuals who observe each other during work, which therefore excludes other team members and observers. To sum up the most unique aspect of the knowledge creation in IT-enabled open innovation setting: while my results suggest that information technologies enable all four knowledge-creation modes for open innovation, the nature of knowledge

creation, and what constitutes knowledge creation, differs in the open innovation context from knowledge creation in organizational settings where collaborators are co-located.

5.3. Limitations and Future Research

This paper presents findings from a revelatory case study. One of this study's limitations is the fact that it is a single case study. However, its findings can be generalized to similar contexts where the task is highly technical and knowledge-intensive and where the team is distributed globally and highly dependent on information technologies. The findings of this study can also be generalized to the companies that are using open source practices inside the environment of a company (i.e., inner sources).

In this paper, I define the open source software team that is conducive to knowledge creation and transfer as a community-based project, (Wasserman, 2009) which is organized in a team structure (Crowston & Howison, 2006). Of course, I acknowledge that not all open source software are developed the same way or by using the same team structure. There are certainly many open source software that are developed by only a single developer. The findings of this study do not apply to open source software development projects with only one member.

Case studies are an excellent way to obtain in-depth knowledge about a context. Yet, this study should be validated with additional case studies from both open source software development contexts, and other open innovation contexts. Furthermore, for statistical generalization to different settings, this study can be followed up with quantitative studies.

While the specific company I studied here had a spinoff company that was built on the Lucene software, according to Apache norms, the contributors to the Lucene project were expected to participate as individuals and represent themselves as individuals rather than representing their companies. Indeed, Apache software teams represent a community-based team under the structure of a non-profit organization; this is one of the four types of open source software development teams that Wasserman (2009) identifies. Accordingly, this study's findings can be generalized to the same type of community-based open source teams. Since all open source software development is not the same, the study should be replicated in other settings. Specifically, because leadership is crucial for the success of open source software (Eseryel & Eseryel, 2013), identifying how individuals with leadership roles and different formal or informal roles contribute to knowledge creation is crucial for the practitioners who aspire to lead such knowledge-creation efforts. Future research may need to distinguish between the knowledge contributions of leaders, developers, and users to provide more precise recommendations for IT-enabled open innovation team leaders.

In this study, I did not analyze social media data because social media does not allow team members to observe each other's work. However, social media (such as Twitter, Facebook, and LinkedIn) are used by open source software development team members, and they may help with knowledge preservation, as observed in the case of Drupal (Wang, Kuzmickaja, Stol, Abrahamsson, & Fitzgerald, 2014). Researchers may be interested in analyzing the role of social media in knowledge creation for open innovation purposes.

Lastly, this study analyzed knowledge-creation behaviors at the development stage of the software based on a single time period. While the findings illuminate the nature of knowledge creation in this context, the nature of open source software development may change in the software's lifecycle. Similarly, knowledge-creation behaviors that are most frequently observed and the participation of different internal and external groups in knowledge creation may change over the life of the team. Therefore, longitudinal studies are needed in order to understand how members of IT-enabled open innovation teams create knowledge over time.

References

Agerfalk, P. B., & Fitzgerald, B. (2008). Outsourcing to an unknown work- force: Exploring opensourcing as a global sourcing strategy. *MIS Quarterly*, *32*(2), 385-410.

AlMarzouq, M., Zheng, L., Rong, G., & Grover, V. (2005). Open source: Concepts, benefits, and challenges. *Communications of the Association for Information Systems*, *16*(37), 756-784.

Ambrosio, J. (2000). *Knowledge management mistakes.* Retrieved from http://www.computerworld.com/industrytopics/energy/story/0,10801,46693,00.html

Barlow, A., & Li, F. (2010). Disruptive technologies and applications. In K. Grant, R. Hackney, & D. Edgar (Eds.), *Strategic information systems* (pp. 77-102). Hampshire, UK: Cengage Learning.

Brooks, F. P. (1995). No silver bullet: Essence and accidents of software engineering. *Computer*, 20(4), 10-19.

Carayannis, E., & Coleman, J. (2005). Creative system design methodologies. *Technovation*, *25*(3), 831-840.

Chesbrough, H. M. (2003). *Open innovation*. Boston, MA: Harvard Business School Press.

Conboy, K., & Morgan, L. (2011). Beyond the customer: Opening the agile systems development process. *Information and Software Technology*, *53*(5), 535-542.

Cougar, J. (1990). Ensuring creative approaches in information system design. *Managerial and Decision Economics*, *11*(2), 281-295.

Cowan, R., David, P. A., & Foray, D. (2000). The explicit economics of knowledge codification and tacitness. *Industrial and Corporate Change*, *9*(2), 211-253.

Cox, A. (1998). *Cathedrals, bazaars and the town council*. Retrieved Jan 10, 2012, from http:// slashdot.org/features/98/10/13/l423253.shtml

Crowson, K., & Howison, J. (2005). The social structure of free and open source software development. *First Monday*, *10*(2).

Crowston, K., & Annabi, H. (2005). *Effective work practices for FLOSS development: A model and propositions.* Paper presented at the In Thirty-Eighth Hawaii International Conference on System Science, Kona, HI.

Crowston, K., & Howison, J. (2006). Hierarchy and centralization in free and open source software team communications. *Knowledge, Technology & Policy*, *18*(4), 65-85.

Daft, R. L. (1978). A dual core model of organizational innovation. *Academy of Management Journal*, *21*(2), 193-210.

Damanpour, F. (1987). Organizational innovation: a meta-analysis of effects of determinants and moderators. *Academy of Management Journal*, *34*(3), 555-590.

Davenport, T. H. (1996). Knowledge management at Hewlett-Packard, early 1996. Retrieved from http://www.bus.utexas.edu/kman/hpcase.htm

Davenport, T. H. (1997). Knowledge management at Ernst & Young. Retrieved from http://www.bus.utexas.edu/kman/e_y.htm

Dewar, R. D., & Dutton, J. E. (1986). The adoption of radical and incremental innovations: An empirical analysis. *Management Science*, *32*(11), 1422-1433.

Dinkelacker, J., Garg, P. K., Miller, R., & Nelson, D. (2002). *Progressive open source*. Paper presented at the Proceedings of the International Conference on Software Engineering, Orlando.

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

Eseryel, D., Eseryel, U. Y., & Edmonds, G. S. (2005). Knowledge management and knowledge management systems. In M. D. Lytras & A. Naeve (Eds.), *Intelligent learning infrastructures for knowledge intensive organizations: A semantic web perspective* (pp. 105-145). Hershey, PA: IDEA Group Publishing.

Eseryel, U. Y., & Eseryel, D. (2013). Action-embedded transformational leadership in self-managing global information technology teams. *The Journal of Strategic Information Systems*, *22*(2), 103-120.

Fitzgerald, B. (2006). The transformation of open source software. *MIS Quarterly*, *30*(3), 587.

Fordham, N. W., Wellman, D., & Sandman, A. (2002). Taming the text: Engaging and supporting students in social studies readings. *The Social Studies*, *93*(4), 149-158.

Gacek, C., & Arief, B. (2004). The many meanings of open source. *IEEE Software*, *21*(1), 34-40.

Gassmann, O., & Enkel, E. (2001). *Towards a theory of open innovation: three core process archetypes*. Retrieved June 24, 2013, from http://www.alexandria.unisg.ch/Publikationen/274

Goldman, R., & Gabriel, R. P. (2005). *Innovation happens elsewhere: Open source as business strategy.* San Francisco: Morgan Kaufmann.

Gurbani, V. K., Garvert, A., & Herbsleb, J. D. (2010). Managing a corporate open source software asset. *Communications of the ACM*, *53*(2), 155-159.

Haefliger, S., & von Krogh, G. (2004). Knowledge creation in open source software development. In H. Tsoukas & N. Mylonopoulos (Eds.), *Organizations as knowledge systems* (pp. 109-129). New York: Palgrave.

Haefliger, S., von Krogh, G., & Spaeth, S. (2008). Code reuse in open source software. *Management Science*, *54*(1), 180-193.

Hars, A., & Ou, S. (2001). Working for free?—Motivations of participating in open source projects. *Proceedings of the 34th Hawaii International Conference on System Sciences*.

Helfat, C. E., & Peteraf, M. A. (2003). The dynamic resource-based view: Capability lifecycles. *Strategic Management Journal*, *24*(10), 997-1010.

Hemetsberger, A., & Reinhardt, C. (2006). Learning and knowledge-building in open-source communities: A social-experiential approach. *Management Learning*, *37*(2), 187-214.

Hermann, S., Hertel, G., & Niedner, S. (2000). *Linux study homepage*. Retrieved from www.psychologie.uni-kiel.de/linux-study/writeup.html

Himanen, P., Torvalds, L., & Castells, M. (2001). *The hacker ethic*. New York: Random House.

Jansen, D. S., & Saiedian, H. (2006). *Test-driven learning: Intrinsic integration of testing into the CS/SE curriculum.* Paper presented at the 37th SIGCSE technical symposium on Computer science education, Houston, Texas, USA.

Jha, S. R. (2002). *Reconsidering Michael Polanyi's philosophy*. Pittsburgh: University of Pittsburgh Press.

Kautz, K., & Thaysen, K. (2001). Knowledge, learning and IT support in a small software company. *Journal of Knowledge Management*, *5*(4), 349-357.

Knipper, K. J. D. T. J. (2006). Writing to learn across the curriculum: Tools for comprehension in content area classes. *Reading Teacher*, *59*(5), 462-470.

Kogut, B. (2000). The network as knowledge: Generative rules and the emergence of structure. *Strategic Management Journal*, *21*(3), 405-425.

Kohanski, D. (1998). *Moths in the machine*. New York: St. Martin's Griffin.

KPMG Consulting. (2000). *Knowledge management research report*. Retrieved from http://www.kpmgconsulting.co.uk/research/othermedia/wf_8519kmreport.pdf

Lee, G. K., & Cole, R. E. (2003). From a firm-based to a community-based model of knowledge creation: The case of the Linux kernel development. *Organization Science*, *14*(6), 633-649.

Lerner, J., & Triole, J. (2000). *The simple economics of open source* (NBER Working Paper No. 7600).

Liebowitz, J., & Beckman, T. (1998). *Knowledge organizations: Why every manager should know*. New York: St. Lucie Press.

Market Wire. (2010). Lucid imagination is a finalist for the 2010 red herring 100 North America award (press release). Retrieved July 28, 2013, from http://www.reuters.com/article/2010/06/17/idUS111990+17-Jun-2010+MW20100617

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis* (2nd ed.). Thousand Oaks, CA: Sage.

Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2002). Two case studies of open source software development: *Apache and Mozilla. ACM Transactions on Software Engineering and Methodology*, *11*(3), 309-346.

Morgan, L., Feller, J., & Finnegan, P. (2011). *Exploring inner source as a form of intraorganisational open innovation*. Paper presented at the 2011 ECIS, Helsinki, Finland.

Morner, M., & von Krogh, G. (2009). A note on knowledge creation in open-source software projects: What can we learn from Luhmann's theory of social systems? *Systemic Practice and Action Research*, *22*(6), 431-443.

Morse, J. M. (2004). Theoretical saturation. In M. S. Lewis-Beck, A. Bryman, & T. F. Liao (Eds.), *Encyclopedia of social science research methods* (Vol. 1-3, pp. 1122-1123). Thousand Oaks, CA: Sage.

Neuendorf, K. A. (2002). *The content analysis guidebook*. Thousand Oaks, CA: Sage Publications.

Nonaka, I., Byosiere, P., Borucki, C. C., & Konno, N. (1994). Organizational knowledge creation theory: A first comprehensive test. *International Business Review*, *3*(4), 337-351.

Nonaka, I., Reinmoeller, P., & Senoo, D. (2000). The art of knowledge. In G. von Krogh, I. Nonaka, & T. Nishiguchi (Eds.), *Knowledge creation: A source of value* (pp. 89-109). New York: St. Martin's Press.

Nonaka, I., & Takeuchi, H. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation.* New York: Oxford University Press.

Nonaka, I., & von Krogh, G. (2009). Tacit knowledge and knowledge conversion: Controversy and advancement in organizational knowledge creation theory. *Organization Science*, *20*(3), 635-652.

Orlikowski, W. J. (1991). Radical and incremental innovations in systems development: An empirical investigation of case tools. Boston, MI: Massachusetts Institute of Technology.

Orlikowski, W. J. (1993). Learning from notes: Organizational issues in groupware implementation. *Information Society*, *9*(3), 237-250.

Polanyi, M. (1966). *The tacit dimension*. Garden City, NY: Doubleday and Co.

Robey, D. (1986). *Designing organization* (2nd ed.). Homewood, IL: Irwin.

Robillard, P. (1999). The role of knowledge in software development. *Communications of the ACM*, *42*(10), 87-92.

Schein, E. H. (1992). *Organizational culture and leadership*. San Francisco: Jossey-Bass Publishers.

Setia, P., Rajogopalan, B., & Sambamurthy, V. (2012). How peripheral developers contribute to open-source software development. *Information Systems Research*, *23*(1), 144-163.

Thomas, D., & Hunt, A. (2004). Open source ecosystems. *IEEE Software*, *32*(1), 89-91.

Trott, P., & Hartmann, D. (2009). Why "open innovation" is old wine in new bottles*. International Journal of Innovation Management*, *13*(4), 715-736.

Tuertscher, P., Garud, R., & Kumaraswamy, A. (Forthcoming). Justification and interlaced knowledge at ATLAS, CERN. *Organization Science.*

Tushman, M., & Nadler, D. (1986). Organizing for innovation. *California Management Review*, *27*(3), 74-92.

von Hippel, E., & von Krogh, G. (2003). Open source software and the "private-collective" model: Issues for organization science. *Organization Science*, *14*(2), 209-223.

von Krogh, G. (2002). The communal resource and information systems. *Journal of Strategic Information Systems*, *11*(2), 85-107.

von Krogh, G., Spaeth, S., & Lakhani, K. R. (2003). Community, joining, and specialization in open source software innovation: A case study. *Research Policy*, *32*(7), 1217-1241.

Wang, X., Kuzmickaja, I., Stol, K.-J., Abrahamsson, P., & Fitzgerald, B. (2014). Microblogging in open source software development: The case of Drupal and Twitter. *IEEE Software*, *31*(4), 72-80.

Wasserman, A. I. (2009). Building a business of open source software. In C. Petti (Ed.), *Cases in technological entrepreneurship (*pp. 107-121). Salento, Italy: Edward Elgar Publishing.

Weber, S. (2004). *The success of open source*. Cambridge, MA: Harvard University Press.

Wesselius, J. (2008). The bazaar inside the cathedral: Business models for internal markets*. IEEE Software*, *25*(3), 60-66.

West, J., & O'Mahony, S. (2005). *Contrasting community building in sponsored and community founded open source projects*. Paper presented at the 38th Annual Hawai'i International Conference on System Sciences, Waikoloa, Hawaii.

Willke, H. (2003). Auf dem Weg zur intelligenten Organisation: Lektionen für Wirtschaft und Staat. In N. Thom & J. Harasymowicz-Bimbach (Eds.), *Wissensmanagement im privaten und öffentlichen Sektor: Was können beide Sektoren voneinander lernen?* (pp. 77-98). ETH Zurich: vdf Hochschulverlag.

Zaltman, G., Duncan, R., & Holbek, J. (1973)*. Innovations and organizations*. New York: Wiley.

Zmud, R. (1982). Diffusion of modern software practices: Influence of centralization and formalization. *Management Science*, *28*(12), 1421-1431.

About the Author

U. Yeliz ESERYEL is an Assistant Professor of Information and Communication Technologies at the University of Groningen, Department of Innovation Management & Strategy. Her research informs the theory and practice on the novel and global forms of organizing through information technologies to increase performance, innovation, global competitiveness, and economic prosperity. She has investigated the divergent patterns of leadership, decision making, communication and coordination in innovative global teams through the use of information technologies. Her research has been published in high-quality journals such as the *Journal of Strategic Information Systems* (JSIS), *IEEE Transactions on Professional Communications*, *Journal for the American Society of Information Science and Technology* (JASIST), *Information and Software Technology Journal*, and the *Journal of Leadership and Management*. Her industry background includes business & strategy consulting and IT project management, in the areas of ERP implementations, enterprise architecture, change management, project management and IT governance. She has been teaching internationally on these topics at the corporate, executive, master's, and undergraduate levels since 2004.