Documentation and Access to Knowledge in Online Communities: Know Your Audience and Write Appropriately?

Carsten Østerlund
Syracuse University, School of Information Studies, Hinds Hall, Syracuse, NY 13244–4100.
E-mail: costerlu@syr.edu

Kevin Crowston
Syracuse University, School of Information Studies, Hinds Hall, Syracuse, NY 13244–4100.
E-mail: crowston@syr.edu

Virtual collaborations bring together people who must work together despite having varied access to and understanding of the work at hand. In many cases, the collaborations are technology supported, meaning that the work is done through shared documents. We develop a framework articulating the characteristics of documents supporting collaborators with access to asymmetric knowledge versus those with access to symmetric knowledge. Drawing on theories about document genre, boundary objects, and provenance, we hypothesize that documents supporting asymmetric collaborators are likely to articulate or prescribe their own (a) purpose, (b) context of use, (c) content and form, and (d) provenance in greater detail than documents supporting symmetric collaborators. We explore these hypotheses through content analysis of documents and instructions for documents from a variety of free/libre open-source projects (FLOSS). We present findings consistent with the hypotheses developed as well as results extending beyond our theory-derived assumptions. When participants have access to the same knowledge, the study suggests that prescriptions about the content of documents become less important compared with prescriptions about the context, provenance, and process of work. The study contributes with a dynamic perspective on communicative practices that consider an often-uneven distribution of knowledge in virtual collaborations.

Received March 12, 2017; revised September 5, 2018; accepted October 2, 2018

 • Published online Month 00, 2018 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/asi.24152

Introduction

The information-technology revolution has led to the proliferation of virtual collaborations both within and across organizations. For many virtual collaborators, documents constitute the primary means for knowledge sharing and exchange. Research has suggested the importance of mutual knowledge (Cramton, 2001), shared mental models (Cannon-Bowers & Salas, 1993), or common ground (Clark & Brennan, 1991) as a basis for communication and collaboration. Yet community members often bring divergent understandings and knowledge from non-converging frames of reference to the production and use of documents, potentially hampering communication. For example, a novice programmer with no history in a particular project may get some sense of the work completed from a report written by a software engineer on the project. However, without knowledge of the individual and organizational practices that went into creating the code and the report, the novice may be unable to determine how to contribute to the project. In contrast, an expert engineer with experience on similar projects may simply need a few key words to guide his or her future work. One document does not fit both audiences. How then can researchers and practitioners best understand and support such heterogeneous virtual collaborations in environments with thousands of users, some deeply involved, many only peripherally so?

In such situations, one is often advised to, "know your audience and write appropriately." But (a) Who is the audience? and (b) How does one write appropriately? In this article we conceptualize different answers to the first question by studying two types of relations among writers and their audience: relations characterized by access to symmetric knowledge versus access to asymmetric knowledge.

By symmetric and asymmetric we mean simply that writers and readers may have access to the same (symmetric) or different (asymmetric) knowledge and assumptions about a context. This comparison allows us to explore the second question by examining what aspects of documents can be tailored based on audiences' different knowledge and what strategies online collaborators can apply to "write appropriately" to these different audiences. Specifically, we address the following broad research question:

What features are expected of documents that link people with access to asymmetric knowledge compared with documents used among people with access to symmetric knowledge?

Answering this question is important for understanding the nature of document use in virtual collaborations and for ensuring the utility of such collaborations, especially as they grow and include participants that are more diverse. Theoretically and empirically, the research contributes to the growing literatures concerned with the interaction of documents, boundary objects, and provenance (Trace, 2016; Huvila, 2011, 2016; Levy, 2016; Shankar et al., 2016; Trace, 2016).

Theory Elaboration and Hypotheses

The study builds on three bodies of theory that describe documents and how they might span groups: genre theory, work on boundary objects, and studies of provenance. These theories were chosen because they address the relation between users' stocks of knowledge and how they use documents. The first perspective focuses on the common knowledge people bring to document production and use (particularly applicable to the case of what we refer to as symmetric knowledge). The second addresses how artifacts, such as documents, can bridge people with few shared points of reference (what we refer to as asymmetric knowledge). The third speaks to how people preserve the history and genealogy of documents to alleviate a lack of shared reference points and knowledge (characteristics of asymmetric knowledge). In short, each perspective focuses on different positions on the continuum from people having a shared understanding of documentation (symmetric knowledge) to groups with few common references (asymmetric knowledge). By combining the three perspectives, we develop a dynamic approach to communicative practices that allows us to investigate how communities adopt different strategies to "write appropriately" for different constituencies.

Genre Theory

Document genre has been defined as typified communicative action invoked in response to a recurrent situation (Bazerman, 1995; Crowston & Kwaśnik, 2003; Orlikowski & Yates, 1994). People engage genres to accomplish social actions in particular situations characterized by a particular purpose, content, and form, with participants in specific times and places. Identification of documents' genres makes them easier to recognize and understand, reducing the effort required to convey meaning. For genres to aid in communication, though, they must be shared by potential collaborators (Swales, 1990). Thus, a genre's utility depends on access to symmetric knowledge among members of a community. When it comes to writing reviews of journal submissions, for instance, senior scholars familiar with the review genre of their field are likely to know the expectations. Conversely, new graduate students who do not share that community's background are unlikely to know the genre and, in turn, bring few, if any, expectations about what purpose, content, and form a document in that genre is likely to convey. We argue that to facilitate communication among people with access to asymmetric knowledge, there will be explicit statements about the genre, namely, a document's expected purpose, form, content, appropriate participants, and time and place of the communication.

Boundary Objects Theory

To further understand the facilitation of communication, we turn to Star and Bowker's work on boundary objects (Bowker & Star, 1999; Star, 1989). People from different communities, with few shared points of reference and little common knowledge (i.e., asymmetric knowledge), must manage the tension between their divergent viewpoints. Star and Bowker introduce the concept of a *boundary object* to explain how such heterogeneous communities maintain productive communication. Accordingly, we posit that documents shared among people with access to asymmetric knowledge may serve as boundary objects. Star describes four types of boundary objects. The first type, repositories (collections of documents) is not applicable for our discussion of individual documents, but the remaining three types offer some helpful ideas.

Star defines *coincidence boundaries* as common objects that have the same boundaries but different internal content. In her work, these boundaries become relevant when work is distributed over a large-scale geographic area. Star points to the state of California as a coincidence boundary for the collaboration among citizen scientists and professional biologists at UC Berkeley. Work occurs in different sites and with different perspectives and can be conducted autonomously, whereas cooperating parties share a common spatial referent. Extending Star's thinking, we suggest that shared documents can specify commonly recognized temporal or participatory boundaries that similarly situate different uses of the document. Extending the publishing example, the editorial and production staff of a journal can agree on an "article" as a bounded unit of work, even while having different interests in and perspectives on what an article contains (a scholarly contribution on the one hand and a chunk of publishable text on the other).

Ideal types are documents such as diagrams, atlases, or other descriptions that provide an exemplary instance of a document without precisely describing the details of any

DOI: 10.1002/asi

particular locality, thing, or activity. It is this quality that makes them useful to people with different points of reference and stocks of knowledge. Such documents demarcate general elements, processes, or organization of the shared context while suppressing distracting or conflicting details. Because their purpose is to span differences in perspective, people with access to symmetric knowledge should not need to use ideal type documents. Contrariwise, people who share little common knowledge and exist at the periphery of the community may find them useful for navigation and orientation. For instance, a scholar might give junior graduate students ideal examples of reviews to guide them when writing their first reviews.

Finally, the fourth type of boundary object, *standardized forms*, offers a uniform way to index communicative content and form. A basic structure for the document's content and form is articulated and is key to the document's genre and particular communicative relationship. Accordingly, people with intimate knowledge of the work at hand have less need for standardized forms. They know what needs to be done and what information will be relevant. Contrariwise, novice review writers might find helpful a review outline that specifies the required elements. Indeed, some journals build such standardized forms into the review system interface.

Documents, standardized forms in particular, often draw on classification schemes to structure the document's content and form, using regularized semantics and objects. Two issues are important when understanding classification systems designed for heterogeneous groups: *comparability* and *visibility*. Comparability means being able to connect instances even when classified differently. Visibility refers to how the classification system exposes or suppresses various features.

People with access to symmetric knowledge have different requirements for comparability and visibility from people with access to asymmetric knowledge. If people intimately know the situation and its practices, little standardization is needed to compare the content of different documents. In contrast, facilitating comparison across less-known settings require regularity in semantics and objects. For instance, journal abstracts often follow a certain format (a "structured abstract"), which helps readers make comparisons to related studies.

The same dynamic plays out for visibility. When creating documents to support work activities, one must differentiate areas of work that are invisible and visible. Invisibility can be regarded as acknowledgment that some information is unimportant. Some work just gets done without needing documentation. Invisibility can also stem from intimacy: A group that has worked together for a long time may no longer need to describe certain activities. But for people at some distance, the document would require more detailed description and an associated classification scheme. For instance, journal article method sections strike a fine balance in how visible to make the activities undertaken by the researchers. More detail would help novice readers; too much might annoy seasoned scholars.

Power plays an intimate role when it comes to a document's comparability and visibility requirements (Bowker & Star, 1999; Shankar et al., 2016). A dissertation advisor trying to keep abreast with a doctoral student's progress will likely require the student to follow a specific format and include certain content to facilitate comparability and visibility of the student's work. By emphasizing some formats and content over others, the advisor can steer the student to pay attention to some research activities over others.

Based on these two theories of genre and boundary objects, then, we posit three hypotheses:

Hypothesis 1. A document shared among people with access to asymmetric knowledge is more likely to require an explicit statement of purpose than one shared among people with access to symmetric knowledge.

Hypothesis 2. A document shared among people with access to asymmetric knowledge is more likely to require an explicit statement of the expected context of use:

- by specifying the appropriate participants, times, and places of its production and use,
- through presentation of ideal types that demarcate the specific elements or organization of the shared work, and
- by demarcations of the boundaries of the shared work. These boundaries can be geographical but can also be defined by the scope of the work required by the project and the specific document.

Hypothesis 3. A document shared among people with access to asymmetric knowledge is more likely to require an explicit statement of the form and content of its communication by:

- bringing regularity in semantics and objects covered by one document to the next, and
- requiring the users to make more details of their work visible in their descriptions.

Provenance Theory

The final theoretical perspective we draw on is provenance theory from archival studies (Gilliland-Swetland, 2005; Ram & Liu, 2012). Historical documents offer an extreme case of a highly asymmetric relationship between what a document provides and the users' knowledge. Accordingly, archivists have long been concerned with preserving background needed to contextualize the use and meaning of historical documents. In particular, archivists record (a) the origin or source of a document and (b) information regarding the origins, custody, and ownership of an item or collection.[1]

People with deep knowledge of a community, for instance, a principal investigator (PI) on a research project, may simply need to know the author, title, and date to

[1] Definition from http://www2.archivists.org/glossary/terms/p/provenance

DOI: 10.1002/asi

position a document in its historical context and the project's evolution. By contrast, a new team member who has yet to become acquainted with the community will require additional details of a document's history to understand its fit in the larger work process. Therefore, we suggest that documents used in settings with asymmetric knowledge require more details about the provenance of their communication and will explicitly state their history to allow better contextualization of a documents' use. This leads us to our final hypothesis:

Hypothesis 4. A document shared among people with access to asymmetric knowledge is more likely to require an explicit statement of the provenance of the communication by referring to:

- the origins of the communication, and
- the genealogy of the communication's use and its ideas.

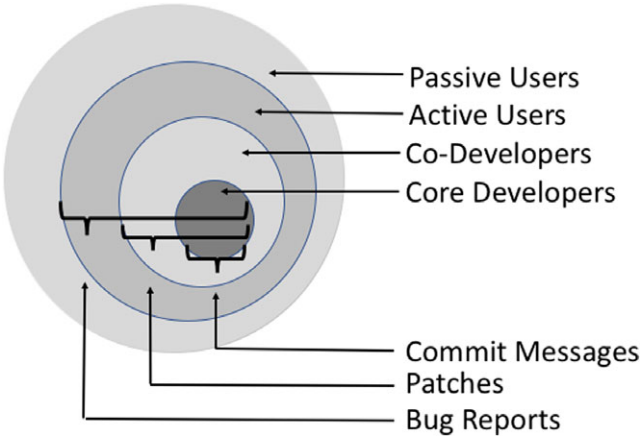


FIG. 1. FLOSS projects' onion structure, showing groups of participants and how documents of different genres span these groups. [Color figure can be viewed at wileyonlinelibrary.com]

Design of the Research

To test our hypotheses and characterize documents linking people with various degrees of access to symmetric or asymmetric knowledge, we chose a structured content-analytic methodology comparing what is required of documents in different collaborative situations. To determine the normative expectations for documents, we examined both a sample of documents and a sample of instructions for creating those kinds of documents. In doing so, we have drawn on the traditions of many genres, boundary objects, and provenance studies that use content analysis, building on an interpretive foundation, as a main methodological approach (Orlikowski, Yates, Okamua, & Fujimoto, 1995; Star, 1989). In general, boundary-object and genre studies approach documents as instantiations of unfolding communication practices. Likewise, current provenance studies in computer science tend to track work practices as content flow between applications and people (Lonsdale, Jensen, Wynn, & Dedual, 2010). Although agreeing with the importance of these approaches and their epistemological stand, in this study we strive to generate a panoramic view where we compare documents across settings with different degrees of access to asymmetric or symmetric knowledge.

Setting

To test the hypotheses developed above, we chose to study documents used in Free/Libre Open Source Software (FLOSS) development projects, a setting in which we could observe documents being used by people with different kinds and levels of shared knowledge. Key to our interest is that most FLOSS projects are developed by primarily virtual teams comprising professionals and users (von Hippel, 2001; von Hippel & von Krogh, 2003) that coordinate their activity primarily through computer-mediated communication tools (Raymond, 1998; Wayner, 2000). As development proceeds, evidence of the processes and interactions between tasks and participants is left in repositories of documents characterized by genre, such as email lists, issue trackers, and source code management systems.

A particular interest is how document use depends on the relationships among FLOSS team members. Several authors have described successful FLOSS teams as having an onion-like structure (Cox, 1998; Gacek & Arief, 2004; Moon & Sproull, 2000; Rossi, 2004). At the center are *core developers* (see Figure 1), who contribute most of the code and oversee the design and evolution of the project (Moon & Sproull, 2000). They are the only participants with the right to add source code to the shared code repository. Scozzi, Crowston, Eseryel, and Li (2008) reported that core developers had shared mental models about the project and the development process. We therefore suggest that the small group of core developers will share a high level of background knowledge about programming in general, and about the project in particular.

Surrounding the core are about an order of magnitude more *codevelopers* (Crowston, Wei, Li, & Howison, 2006). These individuals contribute sporadically by reviewing or modifying code or by contributing bug fixes. They have a much lower level of interaction with the project (Crowston et al., 2006; Mockus, Fielding, & Herbsleb, 2002). We suggest that they thus share less background knowledge than the core developers do about the project specifically. Nevertheless, to be able to make code contributions they must at least share some knowledge of programming practices.

Surrounding the developers are the *active users*: individuals who use the latest releases and contribute bug reports or feature requests (but not code). Because they are not involved in development, we suggest that active users share even less knowledge with developers. A few may be proficient developers, but many others may not be programmers at all. Steinmacher, Graciotto Silva, Gerosa, and Redmiles (2015) identified "previous knowledge" as a major category of hurdles for a newcomer to an open-source

DOI: 10.1002/asi

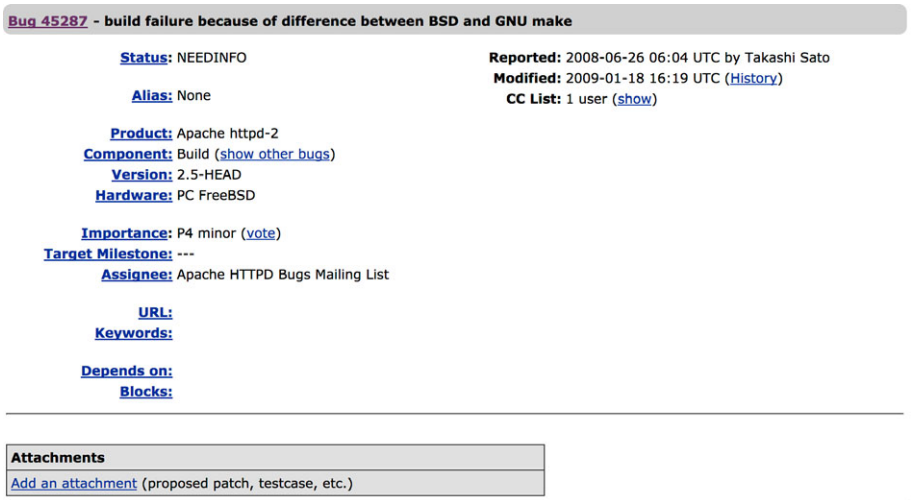


FIG. 2. Example of a bug report. The Apache httpd project (from https://issues.apache.org/bugzilla/show_bug.cgi?id=45287). [Color figure can be viewed at wileyonlinelibrary.com]

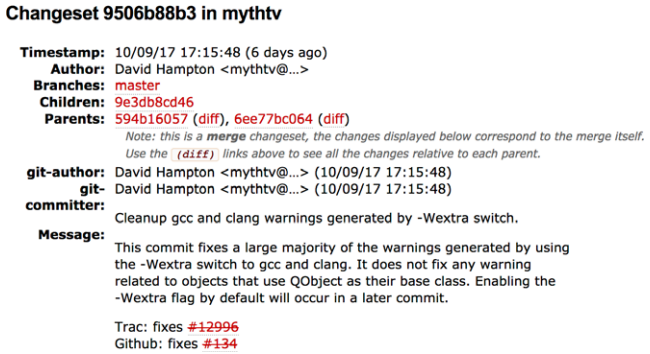


FIG. 3. Example source code control system check in message. The MythTV project (from https://code.mythtv.org/trac/changeset/9506b88b3bc2c3717b32102ece45794fa71f791b/mythtv). [Color figure can be viewed at wileyonlinelibrary.com]

project (that is, for an active user to become a codeveloper). Users' interaction with developers is often channeled through a constrained set of genres; they must present questions and bug reports in the "right way" (Raymond & Moen, 2006) to ensure that the reports communicate information needed for the developers to take action.

Sample

FLOSS projects create a variety of documents. To emphasize our initial theoretical comparison, we chose three kinds of documents whose audiences share knowledge of different degrees of asymmetry or symmetry, specifically *bug reports*, *source code patches,* and *source code commit messages* (see Figure 1). These genres of documents are found in all FLOSS projects, as they are central to the processes of creating and maintaining source code by a distributed group to be used by others.

Bug reports (see Figure 2) are used to report problems with a system. Created by both end users and developers, they are intended for developers because only developers can actually fix bugs. Bug reports can include discussions between users and developers, if, for example, developers request more information about the bug. Thus, bug reports often span two distinct communities (users and developers) who we hypothesize have access to asymmetric knowledge. Projects often maintain a bug reporting system and provide instructions about how and when to report a bug.

The second pair of document types, the source code patch and associated source code commit message, are tightly linked genres. FLOSS projects grow through incremental development, as various developers contribute code to fix bugs or implement new features (Howison & Crowston, 2014). These additions to the code are shared with other developers in the project through patch files, machine-readable files that record the changes made by the developers to move from one version of the source code to another. Patch files can be applied to source code files maintained by other developers even if those developers have made some changes of their own, as long as the changes do not directly conflict. Submitted patches will also be accompanied by additional data, such as a comment describing the changes made. Patch files are created and used primarily by developers, individuals who have considerable shared knowledge.

Most FLOSS projects use a source code control system (SCCS) to maintain the code for a project (for example, Subversion or, more recently, git and github). The SCCS keeps track of the various versions of the code and allows core developers to apply patches that are shared with other developers and then become part of the program that is released to the public. When a patch is added to the SCCS (called a commit), it is usual for the core developer to write a short log message describing the change, creating a source code commit message (see Figure 3). These messages are intended for use by developers, that is, individuals with access to symmetric knowledge.

To test our hypotheses, we searched for explicit instructions and statements of how bug reports, patches, and

DOI: 10.1002/asi

1.4 What to report

When reporting a bug, you should include all information that will help us understand what's wrong, what you expected to happen and how to repeat the bad behavior. You therefore need to tell us:

- your operating system's name and version number

- what version of curl you're using (curl -V is fine)

- versions of the used libraries that libcurl is built to use

- what URL you were working with (if possible), at least which protocol

and anything and everything else you think matters. Tell us what you expected to happen, tell use what did happen, tell us how you could make it work another way. Dig around, try out, test. Then include all the tiny bits and pieces in your report. You will benefit from this yourself, as it will enable us to help you quicker and more accurately.

Since curl deals with networks, it often helps us if you include a protocol debug dump with your bug report. The output you get by using the -v or --trace options.

If curl crashed, causing a core dump (in unix), there is hardly any use to send that huge file to anyone of us. Unless we have an exact same system setup as you, we can't do much with it. Instead we ask you to get a stack trace and send that (much smaller) output to us instead!

The address and how to subscribe to the mailing lists are detailed in the MANUAL file.

FIG. 4. Instructions for reporting a bug. cURL (from http://curl.haxx.se/docs/bugs.html).

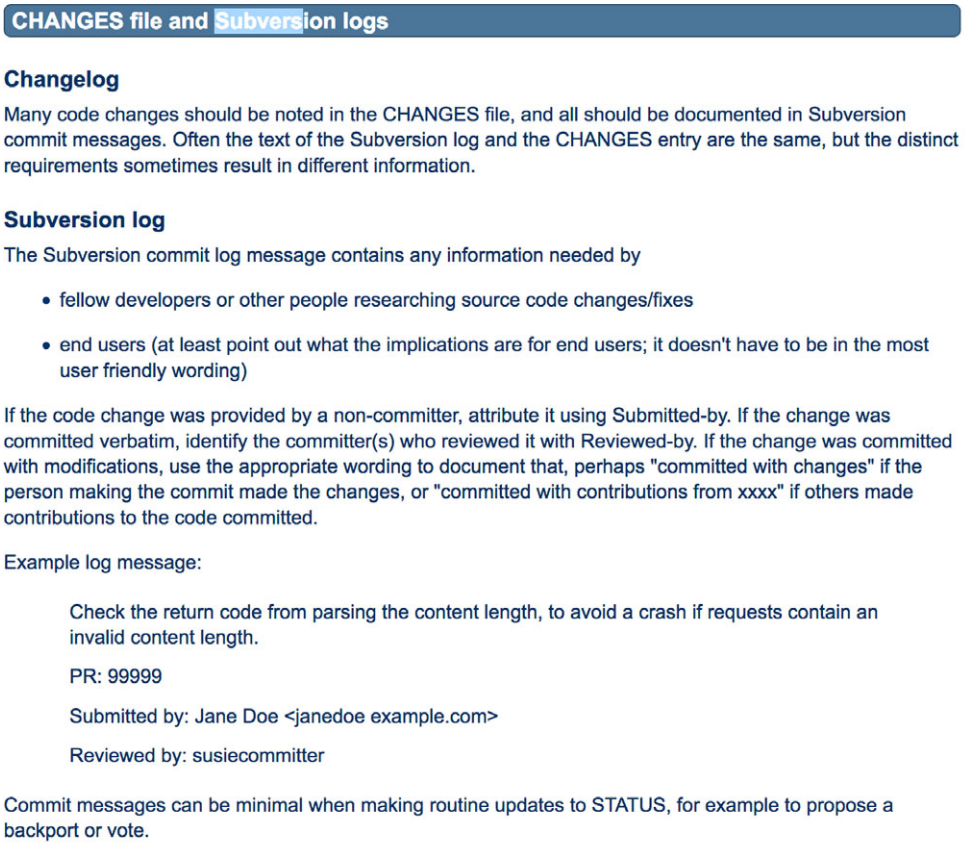
```
The guidelines are not mandatory, and you can decide for yourself which one to follow. But note,
that 10 mins that you spare writing a comment, for example, might lead to significally longer delay
for everyone.

When submitting a patch, please:

 - make a single patch for a single logical change
 - follow the policies and coding conventions below,
 - send patches in unified diff format,
   (using either "cvs diff -u" or "diff -u")
 - provide a log message together with the patch
 - put the patch and the log message as attachment to your email.

The purpose of log message serves to communicate what was changed, and *why*.
Without a good log message, you might spend a lot of time later, wondering where a strange piece of
code came from and why it was necessary.

The good log message mentions each changed file and each rule/method, saying what happend to it,
and why. Consider, the following log message

Better direct request handling.

* new/build-request.jam
  directly-requested-properties-adjuster): Redo.

* new/targets.jam
  (main-target.generate-really): Adjust properties here.

* new/virtual-target.jam
  (register-actual-name): New rule.
  (virtual-target.actualize-no-scanner): Call the above, to detected bugs, where two virtual target
  correspond to one Jam target name.
The log messages for the last two files are good. They tell what was changed. The change to the
first file is clearly undercommented.

It's OK to use terse log messages for uninteresting changes, like ones induced by interface changes
elsewhere.
```

FIG. 5. Instructions for submitting a patch. Boost (from http://www.boost.org/doc/libs/1_39_0/tools/build/v2/hacking.txt).

commit messages should be created or used, thus identifying the norms around these genres. For example, Figure 4 shows an example of instructions for creating a bug report; Figures 5 and 6, for creating a patch and commit message, respectively. By including guidelines for bug reports, source code patches, and commit messages, we span espoused and enacted communication practices among FLOSS participants. The espoused practices, articulated in the instructions, indicate "how one writes appropriately" for the audiences associated with each of these document types. As well, the prevalence or absence of such instructions suggests areas where the community faces communication challenges or conflict. If a document does not lead to major communication problems, there is little need to explicitly state what goes into them. But, if a community struggles with certain types of communications, guidelines may emerge to address when "writing appropriately" does not come easily.

We chose examples of a bug report, source code patches, and commit messages and instructions for these through a purposeful sample of different FLOSS projects. A purposeful sampling was used because, first, there is no complete sampling frame for FLOSS projects to support random sampling. Researchers often use forges (websites such as SourceForge that support multiple projects) as a

DOI: 10.1002/asi



CHANGES file and Subversion logs

Changelog

Many code changes should be noted in the CHANGES file, and all should be documented in Subversion commit messages. Often the text of the Subversion log and the CHANGES entry are the same, but the distinct requirements sometimes result in different information.

Subversion log

The Subversion commit log message contains any information needed by

- fellow developers or other people researching source code changes/fixes
- end users (at least point out what the implications are for end users; it doesn't have to be in the most user friendly wording)

If the code change was provided by a non-committer, attribute it using Submitted-by. If the change was committed verbatim, identify the committer(s) who reviewed it with Reviewed-by. If the change was committed with modifications, use the appropriate wording to document that, perhaps "committed with changes" if the person making the commit made the changes, or "committed with contributions from xxxx" if others made contributions to the code committed.

Example log message:

> Check the return code from parsing the content length, to avoid a crash if requests contain an invalid content length.
>
> PR: 99999
>
> Submitted by: Jane Doe <janedoe example.com>
>
> Reviewed by: susiecommitter

Commit messages can be minimal when making routine updates to STATUS, for example to propose a backport or vote.

FIG. 6. Example instructions for SCCS commit messages (from http://httpd.apache.org/dev/guidelines.html). [Color figure can be viewed at wileyonlinelibrary.com]

basis for sampling, but there are many forges, and many interesting projects use their own infrastructure instead of forge. Second, and more important, given the skewed distribution of project sizes, a random sample would include many small and inactive projects and few, if any, larger projects. However, small projects are less interesting for our purpose, as there is less opportunity for communication across knowledge boundaries. Finally, to examine the validity of our hypotheses, it did not seem critical to generalize statistical estimates of parameters of the entire population of FLOSS projects, for which random sampling would be necessary.

These projects were purposively selected to achieve variation in size, formality of organization, and target audience. With the final dimension, we further manipulated the expected level of symmetry or asymmetry of knowledge by choosing some projects that primarily served programmers or system administrators versus others that served end users. To improve comparability, we selected several projects from the general domains of web services, software development, and multimedia. Table 1 shows the projects examined organized by the dimensions of the sampling. Table 2 presents additional information about each of the projects.

From the 14 project websites, 246 guideline documents were collected for analysis—142 about bug reports, 91 about source code patches, of these 24 cover both bugs and patches, and 13 that mention commit messages. Two coders did the search; the choice of documents was confirmed through weekly discussion with the authors.

Coding

To test our hypotheses, we developed a content-analysis coding system (Krippendorff, 2004; Neuendorf, 2002) for the various document characteristics in the hypotheses. Content analysis was chosen as an appropriate approach because it provides a way to connect our theoretical concepts to the empirical evidence. In other words, the coding system provides an approach to measuring our theoretical constructs, which is necessary to empirically test the proposed hypotheses.

To develop the coding system, we first defined each concept in the hypotheses from the theoretical sources. We then inductively coded a small set of documents to refine these definitions and develop a coding system. We then applied this system to the collected documents. Coding was an iterative process (Hruschka et al., 2004; White & Marsh, 2006), done by two coders using the NVivo program. Initial coding disagreements were discussed to consensus; unresolved issues were discussed at regular meetings with the authors to arrive at an agreed set of

DOI: 10.1002/asi

TABLE 1. FLOSS projects examined.

		Big		Small	
	Corporate	Foundation	Nonorganizational	Foundation	Nonorganizational
Users are programmers (more symmetric)	WebKit	gcc	Boost libraries	ncurses wget	cURL
Users are admins		Apache httpd			phpMyAdmin
Users are not developer (more asymmetric)	VirtualBox	Firefox OpenOffice	MythTV FFmpeg		Pidgin

TABLE 2. Additional project information.

	Projects examined	Size[a]	Sponsorship	Website
1.	WebKit (browser engine)	13 M	Sponsored by Apple	http://www.webkit.org/
2.	gcc (compiler)	6.5 M	Free Software Foundation	http://www.gnu.org/software/gcc/
3.	ncurses (programming library)	240 K	Free Software Foundation	http://www.gnu.org/software/ncurses/
4.	Boost libraries	23 M	Community developed	http://www.boost.org/
5.	FFMPEG (digital video library and tool)	1.1 M	Community developed	http://ffmpeg.org/
6.	cURL (command line web tool)	270 K	Individual-led community	http://curl.haxx.se/
7.	wget (command line web tool)	49 K	Free Software Foundation	https://www.gnu.org/software/wget/
8.	Apache httpd (web server)	700 K	Apache Software Foundation project	http://httpd.apache.org/
9.	phpMyAdmin (web-based database admin tool)	400 K	Software Freedom Conservancy project	http://www.phpmyadmin.net/
10.	VirtualBox (PC emulator)	7 M	Sponsored by Oracle	http://www.virtualbox.org/
11.	OpenOffice (office suite)	23 M	Apache Software Foundation project	http://www.openoffice.org/
12.	Firefox (web browser)	36 M	Mozilla	http://www.mozilla.org/en-US/firefox/new/
13.	MythTV (digital TV recorder)	2.4 M	Community developed	http://www.mythtv.org/
14.	Pidgin (IM client)	270 K	Community developed	http://pidgin.sourceforge.net

[a]Lines of code, from https://www.openhub.net/ except Apache Open Office, from http://bit.ly/KIB_linescode

codes for each document. We also asked the coders to identify regularities in the instructions not previously considered. Such emergent codes were also discussed and if they seemed interesting, were then coded systematically. The resulting coded-document collection was then analyzed quantitatively (i.e., comparing the number of documents with each code) and qualitatively (i.e., examining the content of documents with each code), as described in the following section.

Findings

We compared the documents associated with, first, the wider span between active users and core developers and, second, the narrower gap between active users and codevelopers. Specifically, we compared the instructions given for creating and using bug reports, patches, and commit messages. We present first some general observations before systematically assessing support for each hypothesis.

First, for the wider gap between active users and core developers, as shown in the gap between instructions for bug reports and SCCS commit messages, the differences are striking: More than 142 documents across the 14 projects detailed how active users should communicate about newly-found bugs; only 13 documents from eight projects (six large and two small) explicitly addressed developers with instructions on committing patches. Several projects had no specific instructions for the communication around committing code. Even when there were instructions, it was often in the form of documentation for how to handle security-related issues rather than about the mechanics of day-to-day code commits.

Larger, more established projects (for example, Firefox and OpenOffice) tended to explicate communication expectations more clearly than did smaller projects (for example, cURL and wget). A number of the smaller projects we included (for example, cURL and wget) mainly serve programmers as active users, in addition to their roles as codevelopers and core developers. As a result, even the active-user participants in these small projects have access to more symmetric knowledge compared with larger projects involving large numbers of heterogeneous participants, which we hypothesized would make explicit instructions unnecessary.

Second, for the narrower gap between active users and codevelopers, we examined the difference between instructions for bug reports involving active user and for patches submitted by codevelopers. We found 91 documents associated with source code patches compared with the 142 guidelines for bug reports. Of these, 24 documents covered both bugs and patches. However, our analysis of patch-related documents revealed that a majority of them actually addressed newcomers to the FLOSS project, consistent with the expectation that these instructions would be more useful in case of asymmetric knowledge. That is, patch-related documents often specified the communication process involved in patch creation and submission as a way to help newcomers become

DOI: 10.1002/asi

TABLE 3. Bug reports vs. patches vs. commit messages.

	Bug reports	Patches	Commit messages	χ^2
Hypothesis				
Purpose	11	7	1	8.00+
Hypothesis				
Time	0	0	0	
Boundaries	15	11	1	11.56*
Ideal types	1	10	4	8.40+
Participants	9	17	3	10.21*
Place	30	17	1	26.38***
Hypothesis				
Visibility	12	3	5	6.70+
Standardized form	12	2	5	8.32+
Format	13	9	7	1.93
Content	58	26	8	41.83***
Hypothesis				
Provenance	5	13	2	9.70*
New				
Process	67	79	7	58.35***
Total # of documents	166	115	13	

Note. Count of documents showing number found that exhibited the hypothesized feature. Documents may show multiple features and describe multiple genres, so the total is not the column total. The χ^2 column gives the results of a χ^2 test of independence (df = 2) performed for each feature to test the significance of the difference in the number of documents found per genre.
+$p < .1$, *$p < .05$, **$p < .01$, ***$p < .001$, blank n.s.

involved in the FLOSS endeavor. Many documents discussed both bug reporting and patches as a way to become involved, encouraging bug submissions as a first step. Some projects explicitly suggested that bug reporters themselves should try to fix the code and then submit a patch.

A χ^2 test of independence was performed to test the significance of the differences in the number of documents across the genre for which instructions were provided. The null hypothesis for this test was that instructions should be found equally for all three genres. The actual distribution was statistically significantly different, χ^2 (2, $N = 294$) = 124, $p < .001$. We also conducted an overall χ^2 test to test for a relationship between the document features identified in our hypotheses and the document genre. This test was significant, χ^2 (20, $N = 471$) = 72, $p < .001$, meaning that some features are more common for some genres.

In the remainder of this section, we compare in more detail the instructions for bug reports, patches, and commit messages and examine their consistency with each of our four hypotheses (see Table 3). For each hypothesis, we compare the number of documents found against the null hypothesis that the documents should be equally distributed across the genres. As we are conducting multiple tests, a Holm–Bonferroni correction was performed to control the overall error rate.

Hypothesis 1

We find a span in the degree to which projects articulated the purpose of their bug reports (11), patch submissions (7), and commit messages (1). For instance, the instructions for filing a bug report for the cURL project (Figure 4) clearly state their purpose: to let developers know about problems so they can fix them. The instruction pages for other projects are similarly explicit. By contrast, there are few instruction pages for using the SCCS (for example, Figure 6), and the purpose of the commit messages is not spelled out; rather, it seems to be assumed that the creator understands the role of commit messages. A χ^2 test of independence performed to examine the relation between document genre and number of documents found showed that these differences were statistically significant after the Holm–Bonferroni correction, χ^2 (2, $N = 19$) = 8, $p < .1$.

Hypothesis 2

Consistent with this hypothesis, bug report and patch instructions appear more explicit about the expected context of use compared with a commit message (see Table 3). Looking at the subelements of this hypothesis brings out some interesting similarities and differences.

First, none of the projects specify the *timing* of communication for any of the three types of documents. This could indicate that the projects do not rely on tight temporal schedules or coordination of activities.

Boundaries receive most attention in the bug report (15) and patch (11) instructions, whereas only one project articulates the scope of communication for commit messages (1). Boundaries arise for bugs in specifying which software is concerned. For example, a complex system such as MythTV is built from many components, but users rarely perceive these internal components, and so consider all bugs as originating with the application. Therefore, bug-reporting instructions need to explain how to localize a bug. In addition, instructions give caveats about what kinds of bugs can be fixed and what kinds of new features will be considered. In contrast, the description of a commit message almost never specifies such boundaries. The difference was statistically significant, χ^2 (2, $N = 27$) = 12, $p < .05$.

Ideal types are rarely given for bug reports (1) while they seem a lot more common for patches (10) and interestingly also for commit messages (4). The distribution was statistically significantly different, χ^2 (2, $N = 15$) = 8, $p < .1$. The lack of focus on ideal types for bug reports could be associated with the use of standardized forms for bug reports by many projects.

The same holds true for specifying the relevant *participants,* with relatively few references for bug reports (9) compared with patches (17) and commit messages (3). The distribution was statistically significantly different, χ^2 (2, $N = 29$) = 10, $p < .05$. In part, these expectations are enforced by the technology, as systems limit access to code (for creating, editing, updating, and disposing code) to developers who have these privileges. The bug report instructions from FFMPEG (Figure 7) distinguish different participants and where they are supposed to post their messages.

Reference to the *place* was more common for bug reports (30) compared with patches (17) and commit

DOI: 10.1002/asi

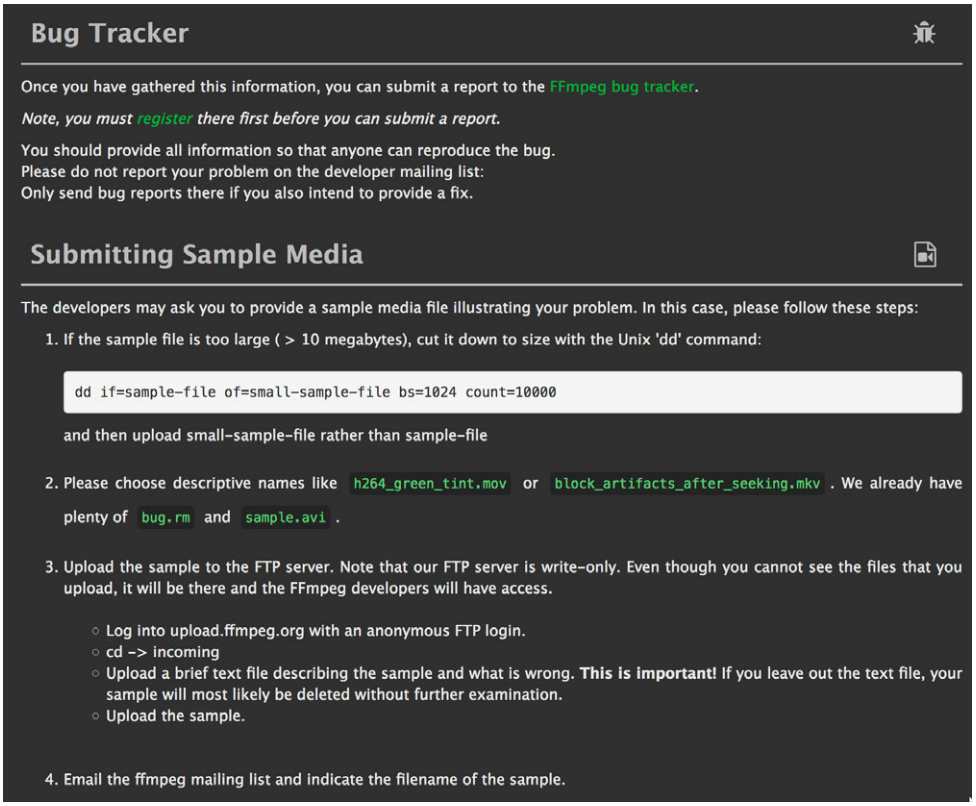


FIG. 7. Example instructions for FFMPEG bug reports (from http://www.ffmpeg.org/bugreports.html). [Color figure can be viewed at wileyonlinelibrary.com]

messages (1). The distribution was statistically significantly different, χ^2 (2, N = 48) = 26, p < .001. For instance, in the instructions for reporting a bug in cURL (Figure 4), we find numerous references to relevant places such as the known bugs list, the bug tracking system, mailing lists, and cURL-users list

Hypothesis 3

As predicted by the hypothesis, more explicit statements about the form and content were found for bug reports (92) compared with patches (37) and commit messages (25). Drilling down into the subhypothesis in Table 3, we find that bug report instructions include a lot more statements about what work should be made *visible* (12). We found only (3) such instructions for patches and interestingly more for commit messages (5), making the distribution statistically significantly different, χ^2 (2, N = 20) = 7, p < .1.

The same was the case for *standardized forms* built into the documentation. Bug reports (12) had the most, followed by commit message forms (5) and patches (2). The distribution was statistically significantly different, χ^2 (2, N = 19) = 8, p < .1. The number of structured fields is greatest for the most institutionalized project, Apache, which uses the Bugzilla bug tracking system. Interestingly, the cURL project simply encourages submissions by email, asking for only basic information. This difference may indicate that cURL users are sophisticated enough to submit good bug reports without explicit guidance because cURL is a command-line tool used primarily by programmers.

Format requirements were most prevalent for bug reports (13), after that patches (9) and commit messages (7); however, the distribution was not statistically significantly different from equal, χ^2 (2, N = 29) = 2, n.s., as there were many more documents than expected for patches and commit messages. The majority of patches and SCCS commit messages are just plain text and can be long or short. However, some projects do suggest fields to include, such as the relevant bug report that the patch fixes, even though these are rarely required and exactly how the patch should be described is left to the developer.

Finally, *content* instructions constitute the most prevalent category across all three document types: bug reports (58), patches (26), and commit messages (8). That content expectations stand out as the most salient dimension of a document for most users should not come as a surprise. The distribution was statistically significantly different, χ^2 (2, N = 92) = 42, p < .001.

Hypothesis 4

Partially consisted with the hypothesis, bug reports (5) and patches (13) called for an explicit statement of the provenance of the communication compared with commit messages (2). The distribution was statistically significantly different, χ^2 (2, N = 20) = 10, p < .05. Yet it should be noted that we found three times as many instructions for patches than bug reports, counter to our expectations. The

DOI: 10.1002/asi

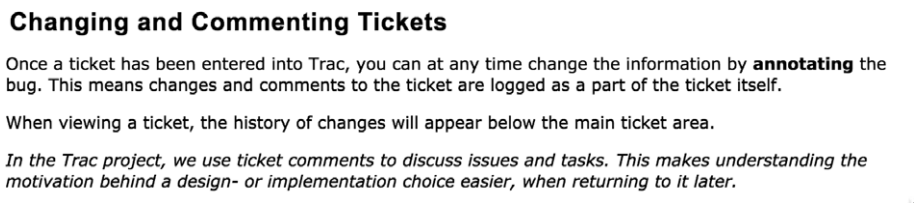


FIG. 8. Discussion of recorded history of changes in VirtualBox bug report instructions (from https://www.virtualbox.org/wiki/TracTickets).

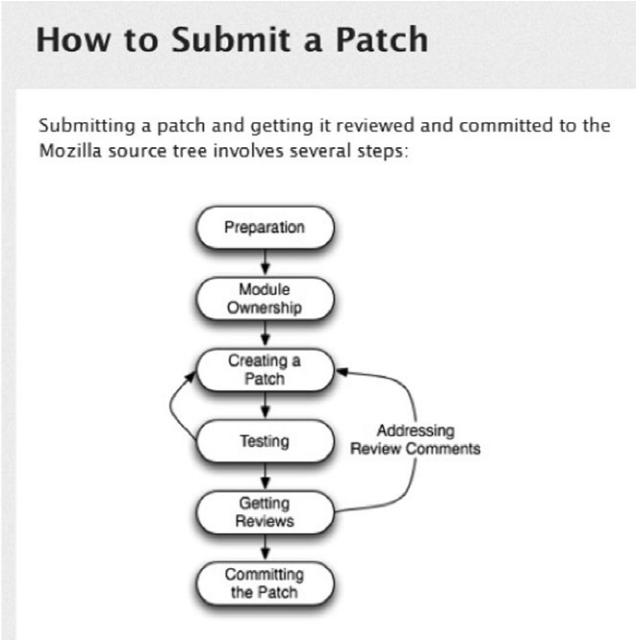


FIG. 9. Example of patch process instruction. Firefox (from https://developer.mozilla.org/en-US/docs/Developer_Guide/How_to_Submit_a_Patch).

relatively paucity of provenance instructions for bug reports might be explained by the system infrastructure used by most FLOSS projects that requires authors to register in the system, allowing others to track the authorship of their documents, developers cc'ed, and important dependencies. Nevertheless, some projects do specify the importance of keeping a bug report's genealogy, as illustrated in an example from VirtualBox (see Figure 8).

Patches committed to the SCCS identify the document's creator, but the commit messages provide no further detail about the sources the author has consulted. This being said, core developers do put some effort into tracking the committed code's genealogy. For example, the Boost project documentation details the benefits of keeping track of the origin, ownership, and location of patches (see Figure 5). As illustrated in Figure 6, the Apache project does specify how core developers can help track who has been involved in the development of committed code.

Summary

In summary, our hypotheses are largely supported by comparing the instructions for bug reports (used by individuals with least access to project knowledge), patches (used by individuals typically with more project knowledge) to those of commit messages (used by individuals with the deepest project knowledge). These findings also support our initial assumption that the groups examined differ in access to shared knowledge. The further one gets away from the core of the project, the more instructions are provided for creating documentation.

Process

Our final finding comes from the open coding, in which we observed an interesting regularity in communication not predicted in our initial hypotheses: a large number of documents explicated the *process* of bug-report (67), patch-related communication (79), and commit messages (7). The distribution was statistically significantly different, χ^2 (2, N = 153) = 58, $p < .001$. It is worth noticing that all the ideal types found for patches (10) depicted the communication process related to patch creation and submission. For example, Figure 9 paints the process of submitting a patch to the Firefox project. We notice how this ideal type for the process provides a simple visual guide spelling out the steps involved in preparing, creating, testing, reviewing, and finally having a core developer commit a new patch.

Content and form versus context of communication.

Comparing instruction documents related to bug reports versus patches, we noticed a difference in how frequently they explicate the context of communication (Hypothesis 2) compared with content and form (Hypothesis 3). Table 3 shows that more documents explicate the *context* of use associated with source code communication, with the exception of specifications of boundaries and of where communication takes place, which is highly prevalent among bug report documents (see Figure 7). The reverse is true for Hypothesis 3. Documents targeting active users submitting bug reports seemed to explicate *content* and *form* (Hypothesis 3) more than patch-related documents. Figure 10 highlights these differences, comparing the percentages of documents relating to each hypothesis out of the total number of either bug reports or patches.

A possible explanation might be that most projects use a bug-tracking system. To use such a system effectively, users need to be told its location and the content and form of the information that they should provide. The system automatically records provenance-relevant information, so bug reporters need not understand what will happen to

DOI: 10.1002/asi

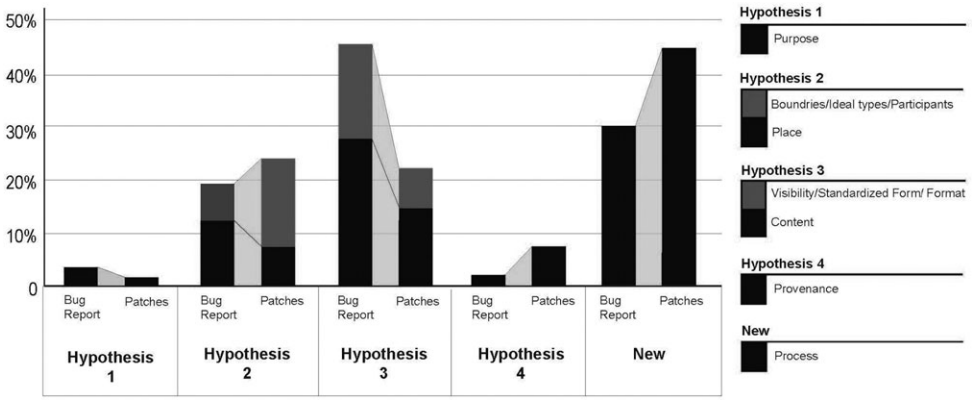


FIG. 10. Comparing the percentages of documents relating to each hypothesis out of the total number of either bug report or patch instructions.

their report in any detail. On the other hand, submitting a patch is more involved and unpredictable and requires a better understanding of the communication's context (Hypothesis 2). To effectively engage in this type of communication, developers must understand where it takes place, who is involved, the boundaries of that work, and ideal representations of the communication process.

As participants move from the periphery as active users to codevelopers submitting source code patches, the knowledge they require about communication practices changes. Knowing where to go, what to communicate about, and in what format signify newcomers' first steps. Understanding the context of communication and its provenance is the next step as they move toward the center of the FLOSS community. Once they become core developers, we hypothesize, they have learned the ropes and so do not need to be reminded about the purpose, provenance, and process. It can be helpful to be reminded about the context of communication and in particular the content and form expectations, but there is a need to explicate communicative expectations only in unusual cases. These results call for further empirical exploration. Rich practice-oriented studies may offer important benefits. Such a perspective would allow us to further explore what aspects of communication require particular attention when serving diverse users.

Discussion

By showing how expectations about documents change across settings where actors have access to varying levels of asymmetric or symmetric knowledge, our findings offer new insights into how the genre, boundary object, and provenance frameworks may inform one another. We explore this relationship in three ways. Genre, boundary objects, and provenance can (a) illuminate different points on a continuum, (b) enrich one another, and (c) contribute to a unified framework. We will address each possibility in turn.

Illuminating Different Points on a Continuum

The literatures on genre, boundary objects, and provenance have each helped articulate how one communicates effectively at different points along the continuum between access to asymmetric and symmetric knowledge. To date, genre studies have tended to focus on groups with access to symmetric knowledge because, by definition, the concept of genre pulls one's attention toward communicative consensus, reflected in typified communicative actions that organizational members invoke in response to recurrent situations (Swales, 1990). At the other end of the continuum, the literature on provenance articulates the importance of tracking documents' histories to facilitate knowledge sharing in situations with highly asymmetric knowledge. Somewhere between these two extremes, the notion of boundary object addresses situations defined by access to asymmetric knowledge. As Star (2010) argues, her work on boundary objects, standards, and infrastructure developed out of an explicit "desire to analyze the nature of cooperative work in the absence of consensus" and shared understanding (Star, 2010, p. 604). Thus, our analysis suggests that these views are complementary, each shedding light on different parts of the continuum of settings.

Enrich One Another

Although the three frameworks offer great analytical power in and of themselves, our findings suggest that further deconstructing the dichotomy between access to symmetric versus asymmetric knowledge can enable these theoretical perspectives to enrich each other. After all, boundaries come in many hues, some stark, others fuzzy, with many in between. Participants bring varying degrees of background to a setting that changes over time. Individuals learn, as do communities, interacting with each other over time.

The literature on boundary objects may therefore gain from cross-pollination with genre and provenance studies. According to Star (2010), the majority of boundary-object studies emphasize interpretive flexibility, that is, the same boundary object can mean different things to different

DOI: 10.1002/asi

groups. For instance, a road map may highlight a series of animal habitats to a zoologist, but to a vacationer it points the way to a campground. However, overemphasis on interpretive flexibility can lead to overlooking how boundary objects arise due to what Star calls "information needs" or, in our terms, due to access to asymmetric knowledge about information and work requirements. Shifting focus from interpretive flexibility to information and work needs and their different material and organizational instantiations provides a more nuanced analysis of not only what people make of boundary objects they frequently use but also what aspects of these recurrent communicative actions are more or less important to them. Genre theory may help articulate these information needs by specifying the context, content, and purpose of recurrent communicative actions. The notion of provenance further allows us to explore the history of such artifacts and how they explicate changes to custody, ownership, content, and form. In short, enriching the notion of boundary object with insights from genre and provenance theory allows us to move beyond an emphasis on interpretive flexibility, adding as well what it takes to overcome the "information needs" of different group, for example, FLOSS project users, codevelopers, and core developers.

Contribute to a Unified Framework

A unified framework combining the three perspectives raises a fundamentally different set of issues. The concept of provenance has gained prominence in computer science to help describe the flow of information across applications and files. The literature on genres and boundary objects may strengthen the descriptive power of this endeavor by highlighting how the movements of such information manifest themselves in different types of objects through people's communication practices (for example, standardized forms and ideal objects). As information constantly gets recycled, reworked, and repackaged, the context, purpose, content, and form may change. Genre theory offers a ready toolbox to describe such changes.

For instance, in our initial hypothesis development, we had not expected that FLOSS participants would explicate the communication process itself. However, process emerged as the most frequently explicated expectation among people with asymmetric knowledge. In retrospect, this explicating process makes sense theoretically. Both contemporary genre and boundary object literatures build on a practice-theory foundation that stipulates that social structures and phenomena only exist as they get produced and reproduced in people's everyday social practices (de Certeau, 1984; Østerlund & Carlile, 2005), and "in response to recurrent situations" (Orlikowski & Yates, 1994). Consistent with both perspectives, it is understandable that FLOSS core developers take time to explicate the sequential process of FLOSS communication activities to help ensure mutual understanding.

Bringing boundary objects, genres, and provenance under the same conceptual roof allows us to consider them in a unified framework, which speaks to the broader literature on documents, boundary objects, and provenance (Trace, 2016; Huvila, 2011, 2016; Levy, 2016; Shankar et al., 2016; Trace, 2016). Taking our point of departure in a community's typified communicative actions invoked in response to recurrent situations (i.e., genres), we observe that participants may bring different stocks of knowledge to the situations. Participants need and seek different things from these recurrent communicative actions. As this study suggests, articulating content and format expectations might be more defining for some recurrent communicative practices than for others, depending on the distribution of knowledge characterizing the context.

A unified framework also allows us to track the origin, development, and sometimes death of typified communicative actions. For instance, how do certain communication practices become standardized? Most of our FLOSS documents are by now part of project infrastructures and standardized regarding what part of the communication can be poorly structured and what parts must be well structured. A detailed comparison of new and established FLOSS projects may further reveal how boundary objects develop into more standardized genre expectations and established repositories with a carefully nurtured provenance.

The historical lens brings debates about power, prevalent in the boundary object literature, into genres and provenance studies. Access to asymmetric knowledge and asymmetric power go hand in hand. The distribution of project-relevant knowledge affects organizational members' ability to perform. Tracking what parts of communication are explicated with regard to particular relations (for example, active users and core developers) may suggest where the organization has experienced conflicts. If communication among organizational members occurs without problems, then there is no need for fixes and thus little incentive to explicate communicative expectations through documents (as seems to be the case among core FLOSS developers). On the other hand, if problems recur in the communication between organizational members (for example, between active users and core developers) the organization may have a need and incentive to explicate expectations. Levina and Orlikowski's (2009) study of power and genres support such assumptions. They found that newcomers are more likely to introduce new communicative practices and thus challenge existing ones. In the FLOSS context, one can imagine active users posting bug reports in all shapes and formats to support their own performances and interests. This variety of reports may have made life difficult for developers, motivating them to explicate communication expectations around bug reports.

Conclusion

Let us return to our original concern: How do common documents serve diverse users? How does one write appropriately when no consensus or access to symmetric knowledge defines the communicative context? For instance, virtual collaborations bring together people with various access to and understanding of the work at hand, yet their

DOI: 10.1002/asi

shared documents must serve these diverse users, many of whom are literally not on the same page.

The present research contributes to both scholarship and practice around this question. The article develops a framework based on three previously separate bodies of literature that characterize documents serving collaborators with access to asymmetric knowledge versus documents supporting those with symmetric knowledge. Drawing on document-centric approaches, we hypothesize that documents supporting asymmetric groups are likely to be more prescriptive and explicate their own use compared with documents supporting symmetric groups.

Our work suggests that practitioners of online communities would benefit from explicitly considering (a) how much access to relevant knowledge various participants hold, and (b) how prescriptive and explicit documents must be to support those various groups. Systematic knowledge of such document variations becomes essential to support heterogeneous online communities.

Theoretically, the research extends the current literature on genre, boundary objects, and provenance by suggesting ways in which they may contribute to one another. Consideration of the varying degrees of access to relevant knowledge opened us analytically to productive cross-pollination between genre, boundary object, and provenance studies. Although our study compared extremes (i.e., bug reports and commit messages), we note that it is not simply a matter of switching between access to asymmetric or symmetric knowledge and with it high and low explication of communication. Interesting dynamics reveal themselves when we examine the space between these extremes. In situations with a highly diverse knowledge among participants (for example, our case of bug reports), explicating the purpose, content, form, and place of communication becomes particularly pertinent. As we narrow the gap a notch (for example, patches), we see a shift toward explicating the context and provenance of communication. What remains constant is the need to explicate the process, the work practices out of which the communication emerges.

Methodologically, online-community studies benefit. In distributed settings, researchers cannot rely on colocation to gain knowledge about work and learning but must establish copresence by engaging in the ongoing communication with a keen eye to the materiality and history of those interactions. Trace data, such as the documents we examined, constitute the stuff of many contemporary online community studies (Geiger & Ribes, 2011). But facing the heaps of data left behind by digital collaborators is daunting. It can be difficult to develop what Beaulieu (2010) labels copresence. Instead of focusing on colocation, copresence can be established through various interaction modes such as joining feeds, participating in discussion boards, and diving into archival materials. Many of these traces are thin data in and of themselves. However, as Ribes (2014) indicates, participants in these online communities must rely on these same traces to make sense of their distributed and "thick activities" (Ribes, 2014, p. 2).

Better understanding of the types of knowledge needed to establish copresence in different online situations will assist researchers navigating distributed collaborations to make informed interpretations and thick descriptions. The framework provides an approach to understanding the practices of heterogeneous audiences with access to differing knowledge. As researchers describe the typified communicative actions invoked in response to recurrent situations, they can explore what information needs drive different audiences, depending on their access to relevant knowledge, and how those organize around specific objects and manifestations of the origins, custody, and ownership of those documents. Researchers can be sensitive to what parts of the interactions the documents explicate, whether it is the content, context, process, or provenance.

A unified genre, boundary object, and provenance framework will further how we theorize online interaction and digital collaborations. Mapping the document universe in, for example, online-learning communities will extend our understanding of the learning trajectories newcomers take as they enter digital collaborations. Access to work and activity awareness often involves documentation. Understanding how such documents are tailored (or not) to facilitate newcomers' legitimate peripheral participation will deepen our understanding of such learning processes. We suggest that newcomers first struggle to master the content and format of certain work practices and only later develop a sense of the work context, its participants, places, and temporal structure. The work process and its provenance are central activities from the beginning, but gain in importance as more forms of participation open up to them.

Finally, the research contributes to system design for online communities and technology-supported collaborations more broadly. The extensive use of standardized forms for bug reports offers some interesting insights that should be critically tested in other organizational contexts dealing with different types of knowledge. In healthcare, for instance, where the central artifact is the human body, not code, one finds a push for more standardized record keeping and information sharing. If it is mainly groups facing a wide knowledge gap who benefit from using standardized forms, resistance to such standardized systems will likely come, one could hypothesize, from groups facing a relatively narrow knowledge gap. Using a standardized form that requires high regularity in semantics and objects and great detail is likely to seem a waste of time for someone with considerable background in the specific area. A detailed understanding of what characterizes documents that support collaborators with different degrees of heterogeneous knowledge could help create systems that tailor content to specific user groups.

References

Bazerman, C. (1995). Systems of genres and the enactment of social intentions. In A. Freedman & P. Medway (Eds.), Genre and the new rhetoric (pp. 79–101). London: Taylor and Francis.

Beaulieu, A. (2010). Research note: From co-location to co-presence: Shifts in the use of ethnography for the study of knowledge. Social Studies of Science, 40(3), 453–470.

Bowker, G.C., & Star, S.L. (1999). Sorting things out: Classification and its consequences. Cambridge, MA: MIT Press.

Cannon-Bowers, J.A., & Salas, E. (1993). Shared mental models in expert decision making. In N.J. Castellan (Ed.), Individual and group decision making (pp. 221–246). Hillsdale, NJ: Lawrence Erlbaum Associates.

Clark, H.H., & Brennan, S.E. (1991). Grounding in communication. In L. B. Resnick, J.M. Levine, & S.D. Teasley (Eds.), Perspectives on socially shared cognition (pp. 127–149). Washington, DC: American Psychological Association.

Cox, A. (1998). Cathedrals, Bazaars and the Town Council. Retrieved from http://slashdot.org/features/98/10/13/1423253.shtml.

Cramton, C.D. (2001). The mutual knowledge problem and its consequences for dispersed collaboration. Organization Science, 12(3), 346–371.

Crowston, K., & Kwaśnik, B.H. (2003). Can document-genre metadata improve information access to large digital collections? Library Trends, 52(2), 345–361.

Crowston, K., Wei, K., Li, Q., & Howison, J. (2006). Core and periphery in Free/Libre and Open Source software team communications. In Proceedings of Hawai'i International Conference on System System (HICSS-39), Kaua'i, Hawai'i. https://doi.org/10.1109/HICSS.2006.101.

de Certeau, M. (1984). The practice of everyday life (S. Rendall, trans.). Berkeley, CA: University of California Press.

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

Geiger, R.S. & Ribes, D. (2011). Trace ethnography: Following coordination through documentary practices. In Proceedings of Hawaii International Conference on System Sciences (HICSS), pp. 1–10. IEEE.

Gilliland-Swetland, A. (2005). Electronic records management. Annual Review of Information Science and Technology, 39(1), 219–253.

Howison, J., & Crowston, K. (2014). Collaboration through open superposition: A theory of the open source way. MIS Quarterly, 38(1), 29–50.

Hruschka, D.J., Schwartz, D., St. John, D.C., Picone-Decaro, E., Jenkins, R.A., & Carey, J.W. (2004). Reliability in coding open-ended data: Lessons learned from HIV behavioral research. Field Methods, 16(3), 307–331.

Huvila, I. (2011). The politics of boundary objects: hegemonic interventions and the making of a document. Journal of the American Society for information Science and Technology, 62(12), 2528–2539.

Huvila, I. (2016). Awkwardness of becoming a boundary object: Mangle and materialities of reports, documentation data, and the archaeological work. The Information Society, 32(4), 280–297.

Krippendorff, K. (2004). Content analysis: An introduction to its methodology. Newbury Park, CA: Sage.

Levina, N., & Orlikowski, W.J. (2009). Understanding shifting power relations within and across organizations: A critical genre analysis. Academy of Management Journal, 52(4), 672–703.

Levy, D.M. (2016). Scrolling forward: Making sense of documents in the digital age (2nd ed.). New York, NY: Arcade.

Lonsdale, H., Jensen, C., Wynn, E., & Dedual, N.J. (2010). Cutting and pasting up: "Documents" and provenance in a complex work environment. Hawai'i International Conference on System Science (HICSS-43), Kauai, HI.

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.

Moon, J., & Sproull, L. (2000). Essence of distributed work: The case of the Linux kernel. First Monday, 5(11). https://doi.org/10.5210/fm.v5i11.801

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

Orlikowski, W.J., & Yates, J. (1994). Genre repertoire: The structuring of communicative practices in organizations. Administrative Science Quarterly, 33, 541–574.

Orlikowski, W.J., Yates, J., Okamua, K., & Fujimoto, M. (1995). Shaping electronic communication: The metastructuring of technology in the context of use. Organization Science, 6(4), 423–444.

Østerlund, C., & Carlile, P. (2005). Relations in practice: Sorting through practice theories on knowledge sharing in complex organizations. The Information Society, 21(2), 91–107.

Ram, S., & Liu, J. (2012). A semantic foundation for provenance management. Journal on Data Semantics, 1(1), 11–17.

Raymond, E. (1998). The cathedral and the bazaar. First Monday, 3(2). https://doi.org/10.5210/fm.v3i2.578

Raymond, E.S. & Moen, R. (2006). How to ask questions the smart way. Retrieved from http://catb.org/~esr/faqs/smart-questions.html.

Ribes, D. (2014). Ethnography of scaling, or, how to a fit a national research infrastructure in the room. In Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (pp. s158–s170). New York: ACM.

Rossi, M.A. (2004). Decoding the free/open source puzzle: A survey of theoretical and empirical contributions. In J. Bitzer & P. Schroder (Eds.), The economics of open source software development: Analyzing motivation, organization, innovation and competition in the open source software revolution (pp. 15–55). Amsterdam, The Netherlands: Elsevier Press.

Scozzi, B., Crowston, K., Eseryel, U.Y., & Li, Q. (2008). Shared mental models among open source software developers. In Proceedings of Hawai'i International Conference on System System (HICSS-41), Big Island, Hawai'i.

Shankar, K., Hakken, D., & Østerlund, C. (2016). Rethinking documents. In U. Felt, R. Fouché, C.A. Miller, & L. Smith-Doerr (Eds.), Handbook of science and technology studies (4th ed.; pp. 59-85). Cambridge: MIT Press.

Star, S.L. (1989). The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving. In L. Gasser & M.N. Huhns (Eds.), Distributed artificial intelligence (Vol. 2, pp. 37–54). San Mateo, CA: Morgan Kaufmann.

Star, S.L. (2010). This is not a boundary object: Reflections on the origin of a concept. Science, Technology & Human Values, 35(5), 601–617.

Steinmacher, I., Graciotto Silva, M.A., Gerosa, M.A., & Redmiles, D.F. (2015). A systematic literature review on the barriers faced by newcomers to open source software projects. Information and Software Technology, 59, 67–85.

Swales, J.M. (1990). Genre analysis: English in academic and research settings. New York: Cambridge University Press.

Trace, C.B. (2016). Ethnomethodology: Foundational insights on the nature and meaning of documents in everyday life. Journal of Documentation, 72(1), 47–64.

von Hippel, E.A. (2001). Innovation by user communities: Learning from open-source software. Sloan Management Review, 42(4), 82–86.

von Hippel, E.A., & von Krogh, G. (2003). Open source software and the "private-collective" innovation model: Issues for organization science. Organization Science, 14(2), 209–213.

Wayner, P. (2000). Free for all. New York: HarperCollins.

White, M.D., & Marsh, E.E. (2006). Content analysis: A flexible methodology. Library Trends, 55(1), 22–45.