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

Carsten Østerlund* & Kevin Crowston

* Corresponding author

Syracuse University School of Information Studies
Hinds Hall, Syracuse NY 13244–4100
tel: +1 315 443–8773 fax: +1 315 443–6886
costerlu@syr.edu, crowston@syr.edu
Abstract

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 collaboration is technology supported, meaning that the work is done through shared documents of various kinds. We develop a framework articulating the characteristics of documents supporting collaborators with asymmetric access to knowledge versus those with symmetric access to 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 (1) purpose, (2) context of use, (3) content and form, and (4) provenance in greater detail than documents supporting symmetric collaborators. We explore these hypotheses through content analysis of documents and instructions 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. As participants gradually gain access to knowledge, the study suggests, prescriptions about the content of documents become less important compared to prescriptions about the context, provenance, and process of work. The study suggests new directions for research on communications in virtual collaborations, as well as advice for those supporting such collaborations.

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 [Olson & Olson] (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, 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 make a contribution. 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
deployed deeply involved, many only peripherally so?

In such situations, one is often advised to, “know your audience and write appropriately.”
But (1) Who is the audience? and (2) How does one write appropriately? In this paper, we
conceptualize different answers to the first question by studying two types of relations among
writers and their audience: relations characterized by symmetric access to knowledge vs.
asymmetric access to knowledge. By symmetric and asymmetric we mean simply that writers
and readers may have access to the same (symmetric) or different (asymmetric) sets of
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
background knowledge and what strategies online collaborators can apply to “write
appropriately” to these different audiences. More specifically, we address the following broad
research question:

What features are expected of documents that link people with asymmetric access
to background knowledge compared to documents used among people with
symmetric access to 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 (Ciaran, 2016; Huvila, 2011, 2016; Levy, 2016; Shankar et al., 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 (particular 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 background knowledge.

**Genre Theory**

Document genre has been defined as typified communicative action invoked in response to a recurrent situation (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 symmetric access to knowledge among members of a community. When it comes to writing journal reviews, for instance, senior scholars familiar with the review genre of their field are likely to know the expectations implied. Conversely, new graduate students who do not share that community’s background knowledge 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 asymmetric access to knowledge, there must be explicit statements about the genre, namely a document’s 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, *boundary object*, to explain how such heterogeneous communities maintain productive communication. We posit that documents shared among people with asymmetric access to 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. They arise 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 while 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 do not precisely describe the details of any one locality, thing, or activity, remaining somewhat vague and abstract. It is this quality which 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. Since their purpose is to span differences in perspective, people with symmetric access to 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 strong 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. Novice review writers, for instance, might find a basic outline helpful that specify required elements. Some journals build such standardized forms into their reviewer 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 symmetric access to knowledge have different requirements for comparability and visibility from people with asymmetric access to knowledge. If people intimately know the situation and its practices, little documentation is needed to compare the content of different document. 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 acknowledgement 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. 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). 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 in some detail:

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

**Hypothesis 2:** A document shared among people with 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 which 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 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. Historical documents offer an extreme case of a highly asymmetric relationship between what a document provides and the users’ background knowledge. Accordingly, archivists have long been concerned with preserving background knowledge to contextualize the use and meaning of historical documents. In particular, archivists record (1) the origin or source of a document, and (2) information regarding the origins, custody, and ownership of an item or collection\(^1\).

People holding significant background knowledge about a community, for instance, a PI on a research project, may simply need to know the author, title, and date to 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 into the larger work process. They likely gain little from the minimal audit trail common to most systems. Therefore, we suggest that documents used by people with asymmetric access to knowledge require more details about the provenance of their communication and will explicitly state their history to allow better contextualization of documents’ use. This leads us to our final hypothesis:

\(^1\) Definition from http://www2.archivists.org/glossary/terms/p/provenance
**Hypothesis 4:** A document shared among people with 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.

**Design of the Research**

To test our hypotheses and characterize documents linking people with various degrees of asymmetric access to background knowledge, we chose a structured content analytic methodology comparing what is required of documents across collaborative situations. To determine the normative expectations for documents, we examined both a sample of documents and sample of instructions for creating those kinds of documents. In doing so, we have drawn on the traditions of many genre, boundary object, and provenance studies that use content analysis, building on an interpretive foundation, as a main methodological approach (Orlikowski, et al, 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 et al., 2010). While 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 different degrees of asymmetric access to background 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 background knowledge. Key to our interest is that most FLOSS projects are developed by primarily virtual teams comprising professionals and users (von Hippel, 2001) that coordinate their activity primarily through computer-mediated communication tools (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 (e.g., Gacek & Arief, 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. They are the only participants with the right to add source code to the shared code repository.

![FLOSS projects’ onion structure, participants and documents.](image)

**Figure 1.** FLOSS projects’ onion structure, participants and documents.

Surrounding the core are perhaps ten times as many co-developers. These individuals contribute sporadically, reviewing or modifying code or contributing bug fixes; they need a much lower level of interaction. They thus share somewhat less background knowledge than the core developers do.
Surrounding the developers are the *active users*: individuals who use the latest releases and contribute bug reports or feature requests (but not code). Since they are not involved in development, we hypothesize that active users share even less background knowledge with developers. 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 have different degrees of asymmetric knowledge, 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, since 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 hypothesized would have little shared background 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. These 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 that we hypothesized would have considerable shared background knowledge.
Most FLOSS projects use a source code control system (SCCS) to maintain the code for a project (e.g., Subversion or 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 symmetric access to knowledge.

**Changeset 9506b88b3 in mythtv**

<table>
<thead>
<tr>
<th>Timestamp:</th>
<th>10/09/17 17:15:48 (6 days ago)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author:</td>
<td>David Hampton &lt;mythtv@...&gt;</td>
</tr>
<tr>
<td>Branches:</td>
<td>master</td>
</tr>
<tr>
<td>Children:</td>
<td>9e3db8cd46</td>
</tr>
<tr>
<td>Parents:</td>
<td>594b16057 (diff), 6ee77bc064 (diff)</td>
</tr>
</tbody>
</table>

*Note: this is a merge changeset, the changes displayed below correspond to the merge itself.*

Use the (diff) links above to see all the changes relative to each parent.

**git-author:**

David Hampton <mythtv@...> (10/09/17 17:15:48)

**git-committer:**

David Hampton <mythtv@...> (10/09/17 17:15:48)

**Message:**

Cleanup gcc and clang warnings generated by -Wextra switch.

This commit fixes a large majority of the warnings generated by using the -Wextra switch to gcc and clang. It does not fix any warning related to objects that use QObject as their base class. Enabling the -Wextra flag by default will occur in a later commit.

Trac: fixes #12996
Github: fixes #134

Figure 3. Example source code control system check in message. The MythTV project (from https://code.mythtv.org/trac/changeset/9506b88b3bc2c3717b32102ece45794fa71f791b/mythtv).

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

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.

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

**Figure 5.** Instructions for submitting a patch. Boost (from http://www.boost.org/doc/libs/1_39_0/tools/build/v2/hacking.txt).
Figure 6. Example instructions for SCCS commit messages (from http://httpd.apache.org/dev/guidelines.html).

We chose examples of 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 like SourceForge that support multiple projects) as a basis for sampling, but there are many forges, and many interesting projects use their own infrastructure instead. 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.

Table 1 shows the projects examined organized by the dimensions of the sampling. Table 2 presents additional information about each of the projects.

<table>
<thead>
<tr>
<th>Users are programmers (symmetric)</th>
<th>Big</th>
<th>Small</th>
</tr>
</thead>
<tbody>
<tr>
<td>Users are admins</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Users are not developed (asymmetric)</td>
<td>VirtualBox</td>
<td></td>
</tr>
</tbody>
</table>

These projects were purposively selected to achieve variation in size, formality of organization, and target audience. To improve comparability, we selected several projects from the general domains of web services, software development, and multimedia.

Table 2

<table>
<thead>
<tr>
<th>FLOSS Projects Examined</th>
<th>Size¹</th>
<th>Formality</th>
<th>Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. WebKit (browser engine)</td>
<td>13M</td>
<td>Sponsored by Apple</td>
<td><a href="http://www.webkit.org/">http://www.webkit.org/</a></td>
</tr>
<tr>
<td>2. gcc (compiler)</td>
<td>6.5M</td>
<td>Free Software Foundation</td>
<td><a href="http://www.gnu.org/software/gcc/">http://www.gnu.org/software/gcc/</a></td>
</tr>
<tr>
<td>3. ncurses (programming)</td>
<td>240K</td>
<td>Free Software Foundation</td>
<td><a href="http://www.gnu.org/software/ncurses/">http://www.gnu.org/software/ncurses/</a></td>
</tr>
<tr>
<td></td>
<td>Library/Project</td>
<td>Lines of Code</td>
<td>Development Team</td>
</tr>
<tr>
<td>---</td>
<td>---------------------------------------</td>
<td>---------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>4.</td>
<td>Boost libraries</td>
<td>23M</td>
<td>Community developed</td>
</tr>
<tr>
<td>5.</td>
<td>FFMPEG (digital video library and tool)</td>
<td>1.1M</td>
<td>Community developed</td>
</tr>
<tr>
<td>6.</td>
<td>cURL (command line web tool)</td>
<td>270K</td>
<td>Individual-led community</td>
</tr>
<tr>
<td>7.</td>
<td>wget (command line web tool)</td>
<td>49K</td>
<td>Free Software Foundation</td>
</tr>
<tr>
<td>8.</td>
<td>Apache httpd (web server)</td>
<td>700K</td>
<td>Apache Software Foundation project</td>
</tr>
<tr>
<td>9.</td>
<td>phpMyAdmin (web-based database admin tool)</td>
<td>400K</td>
<td>Software Freedom Conservancy project</td>
</tr>
<tr>
<td>10.</td>
<td>VirtualBox (PC emulator)</td>
<td>7M</td>
<td>Sponsored by Oracle</td>
</tr>
<tr>
<td>11.</td>
<td>OpenOffice (office suite)</td>
<td>23M</td>
<td>Apache Software Foundation project</td>
</tr>
<tr>
<td>13.</td>
<td>MythTV (digital TV recorder)</td>
<td>2.4M</td>
<td>Community developed</td>
</tr>
</tbody>
</table>

1 lines of code from https://www.openhub.net/; count for Apache Open Office is from http://bit.ly/KIB_linescode

From the 14 project websites, 266 guideline documents were collected for analysis—142 bug reports, 91 source code patches, 24 that 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 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 done by two coders using the NVivo program. Initial coding disagreements were discussed to consensus (100% agreement); unresolved issues were discussed at regular meetings with the authors to arrive at an agreed set of codes. 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 co-developers. Specifically, we compared the instructions given for creating and using bug reports, patches and to commit messages.
First, for the wider gap between 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 8 projects (6 large and 2 small) explicitly addressed core developers with instructions on committing patches. Several projects had no specific instructions for the communication around committing code. Often, projects included documentation on security-related issues rather than the mechanics of day-to-day code commits.

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

Second, the narrower space between bug report instructions involving active user and patches submitted by co-developers did not exhibit the same differences as the wider gap between bug reports and commit message instructions. We found 91 documents associated with source code patches compared to the 142 guidelines for bug reports. Of these 24 covered both bugs and patches. Our analysis of patch-related documents revealed that a majority of them addressed newcomers to the FLOSS project. Typical patch-related documents specify the communication process involved in patch creation and submission as a way to help newcomers become 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. Below, we compare the instructions for bug reports, patches and SCCS commit messages and their consistency with each of our four hypotheses (See Table 3).

![Bug Tracker](image)

**Figure 7.** Example instructions for FFmpeg bug reports (from http://www.ffmpeg.org/bugreports.html).

**Hypothesis 1**

We find quite 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, when one manages to find instruction pages for using the SCCS (e.g., Figure 6), these rarely state the purpose of the commit messages; rather, it seems to be assumed that the creator understands the role of commit messages.

Hypothesis 2

Consistent with this hypothesis, bug report (55) and patch (55) instructions appear more explicit about the expected context of use compare to commit message (9) (see Table 3). Looking at the sub-elements of this hypothesis bring out some interesting similarities and differences. None of the projects specify the timing of communication for either of the three types of documents.

**Boundaries** receive most attention in the bug report (15) and patch (11) instructions while only one project bother articulate the scope of communication for commit messages (1). 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 does not specify such boundaries.

**Ideal types** are rarely given for bug reports (1) while they seem a lot more significant for patches (10) and interestingly also commit messages (4). The same hold true for specifying the relevant participants with relatively few references for bug reports (9) compared to patches (17) and commit messages (3). 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 stands out as significantly more relevant for bug reports (30) compared to patches (17) and commit messages (1). 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.

<table>
<thead>
<tr>
<th>Table 3</th>
<th>Bug Reports vs. Patches vs. Commit messages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Bug Reports</td>
</tr>
<tr>
<td>Hypothesis 1</td>
<td></td>
</tr>
<tr>
<td>Purpose</td>
<td>11</td>
</tr>
<tr>
<td>Hypothesis 2</td>
<td></td>
</tr>
<tr>
<td>Boundaries</td>
<td>15</td>
</tr>
<tr>
<td>Ideal Types</td>
<td>1</td>
</tr>
<tr>
<td>Participants</td>
<td>9</td>
</tr>
<tr>
<td>Place</td>
<td>30</td>
</tr>
<tr>
<td>Time</td>
<td>0</td>
</tr>
<tr>
<td>Hypothesis 3</td>
<td></td>
</tr>
<tr>
<td>Visibility</td>
<td>12</td>
</tr>
<tr>
<td>Standardized Form</td>
<td>12</td>
</tr>
<tr>
<td>Format</td>
<td>13</td>
</tr>
<tr>
<td>Content</td>
<td>58</td>
</tr>
<tr>
<td>Hypothesis 4</td>
<td></td>
</tr>
<tr>
<td>Provenance</td>
<td>5</td>
</tr>
<tr>
<td>New</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>67</td>
</tr>
<tr>
<td>Total</td>
<td>233</td>
</tr>
</tbody>
</table>

Hypothesis 3

As predicted by the hypothesis more explicit statement about the form and content was required for bug reports (92) compared to patches (37) and commit messages (25). Drilling down into the sub-hypothesis 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). The same was the case for standardized forms build into the documentation. Bug reports (12) had the most followed by commit message forms (5) and patches (2). 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, since 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). The majority of patches and SCCS commit messages are just plain text and can be long or short. Some projects do suggest fields to include, such as the relevant bug report that the patch fixes, but these are rarely required. Furthermore, exactly how the patch should be described is left to the developer.

Content instructions constitute the most prevalent category across all three document types: bug reports (58), patches (26) and commit messages (8).

Hypothesis 4

Partially consisted with the hypothesis bug reports (5) and patches (13) called for explicit statement of the provenance of the communication compared to commit messages (2). Yet, it should be noted that we found three times as many instructions for patches than bug reports. Nevertheless, some projects do specify the importance of keeping a bug report’s genealogy, as illustrated in an example from VirtualBox (see Figure 8). In regard to patches, the Boost project documentation details the benefits of keeping track of the origin, ownership, and location of
patches (see Figure 5). The relatively few provenance instructions for bug reports might be explained by the system infrastructure used by most FLOSS projects requiring authors to register in the system, allowing others to track the authorship of their documents, developers cc’ed, and important dependencies. 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, 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.

In summary, our hypotheses are largely supported by comparing the instructions for bug reports (used by individuals with least access to project knowledge), patched (used by individuals typically with more project knowledge) to those of commit messages (used by individuals with the deepest project knowledge). The further one get away from the core of the project the more instructions are provided for creating documentation.

**Changing and Commenting Tickets**

Once a ticket has been entered into Trac, you can at any time change the information by annotating the bug. This means changes and comments to the ticket are logged as a part of the ticket itself.

When viewing a ticket, the history of changes will appear below the main ticket area.

*In the Trac project, we use ticket comments to discuss issues and tasks. This makes understanding the motivation behind a design- or implementation choice easier, when returning to it later.*

**Figure 8.** Discussion of recorded history of changes in VirtualBox bug report instructions

(from https://www.virtualbox.org/wiki/TracTickets)

**Process**

Our final finding comes from the open coding of interesting regularities in communication not predicted in our initial hypotheses: a significant number of documents
explicated the process of bug-report (67), patch-related communication (79) and commit messages (7). 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 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.

![How to Submit a Patch](image)

**Figure 9.** Example of patch process instruction. Firefox (from [https://developer.mozilla.org/en-US/docs/Developer_Guide/How_to_Submit_a_Patch](https://developer.mozilla.org/en-US/docs/Developer_Guide/How_to_Submit_a_Patch)).
Content and Form vs. Context of Communication

Comparing documents related to bug reports versus patches, we noticed a difference in how frequently documents explicate the context of communication (Hypothesis 2) compared to its 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 this 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 their report in any detail. 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 co-developers 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 one moves toward the center of the FLOSS community. Once one becomes a core developer, we
hypothesize, one knows the ropes and do not need to be reminded about the purpose, provenance and process. However, it can be helpful to be reminded about the context of communication and in particular the content and form expectations. only needs to explicate communicative expectations in unusual cases.

[[These results call for further empirical exploration. Rich practice-oriented studies may offer significant benefits. Such a perspective would allow us to further explore what aspects of communication require particular attention when serving diverse users. ]] [[ndern?]]

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

**Discussion**

By showing how documents change with varying levels of shared background 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: (1) illuminate different points on a continuum, (2) enrich one another, and (3) 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 high and low access to background knowledge. To date, genre studies have tended to focus on groups with symmetric access to genre expectations. By definition, the concept of genre pulls one’s attention towards communicative consensus, 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 highly asymmetric situations. Somewhere between these two extremes, the notion of boundary object pinpoints situations defined by asymmetric access to background 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).

**Enrich one another**

While the three frameworks offer great analytical power in and of themselves, our findings suggest that further deconstructing the dichotomy between high and low access to background 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 knowledge to a setting which changes over time. Individuals learn, as do communities, interacting with each other over time.

The literature on boundary objects likewise gains from cross-pollination with genre and provenance studies. According to Star (2010), the majority of boundary object studies emphasize interpretive flexibility, i.e., the same boundary object can mean different things to different 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. By overemphasizing interpretive flexibility, however, one easily loses track of how boundary objects arise due to what Star calls “information needs” or in our terms, due to asymmetric access to background knowledge about information and work requirements. Shifting focus from interpretive flexibility to information and work needs and their different material and organizational instantiations allows us to provide 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 overemphasis on interpretive flexibility. Instead we can consider what it takes to overcome the “information needs” of different group, e.g., FLOSS project users, co-developers and core developers.
Contribute to a unified framework

The concept of provenance has gained prominence in computer science to help describe the flow of information across application and file. 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 (e.g., 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, 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 (Østerlund & Carlile, 2005), and “in response to recurrent situations” (Orlikowski & Yates, 1994, p. 544). 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 (Ciaran, 2016; Huvila, 2011, 2016; Levy, 2016; Shankar et al., 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 levels of background knowledge to the situation. 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 background knowledge characterizing the context.

A unified framework may also allow us to track the origin, development, and sometimes death of typified communicative actions. For instance, how does 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. Asymmetric access to knowledge and asymmetric power go hand in hand. The distribution of project relevant background knowledge affects organizational members’ ability to perform. Tracking what parts of communication are explicated in regard to particular relations (e.g., 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 (e.g., 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 symmetric access to background 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 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. First, the paper develops a framework based on three previously separate bodies of literature that characterize documents serving collaborators with asymmetric access to 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 to documents supporting symmetric groups. Second, our work suggests that practitioners of online communities would benefit from explicitly considering (1) how much access to relevant knowledge various participants hold, and (2) 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 background knowledge opened us analytically to productive cross-pollination between genre, boundary object, and provenance studies. While our study compared extremes (i.e., bug reports and commit messages), we note that it is not simply a matter of switching between high and low access to background 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 wide knowledge bag among participants (bug reports), explicating the purpose, content, form, and place of communication becomes particularly pertinent. As we narrow the gap a notch (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 co-location to gain knowledge about work and learning but must establish co-presence 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 as co-presence. Instead of focusing on co-location, co-presence 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 indicates (2014), 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 background knowledge needed to establish co-presence 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 differing access to background knowledge. As researchers describe the typified communicative actions invoked in response to recurrent situations, they can explore what information needs drive different sub-audiences, depending on their access to relevant background 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. In particular, the extensive use of standardized forms for bug reports may provide some interesting insights. In healthcare, for instance, one finds a push for more standardized record keeping and information sharing. If it is mainly groups harboring a wide knowledge gap who benefit from using standardized forms, one may assume that resistance to standardized systems will come from group with relative narrow knowledge gap in their use of healthcare information systems. Using a standardized form that requires high regularity in semantics and objects and great detail may seem like a waste of time for someone with considerable background knowledge 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


