Knowledge Work Artifacts: Kernel Cousins for Free/Open Source Software Development

Margaret Elliott
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, California
melliott@ics.uci.edu

Mark S. Ackerman
School of Information and Department of Electrical Engineering and Computer Science.
University of Michigan
Ann Arbor, Michigan
ackerm@umich.edu

Walt Scacchi
Institute for Software Research
Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, California
wscacchi@ics.uci.edu

ABSTRACT
Most empirical studies of peer production have focused on the final products of these efforts (such as software in Free/Open Source projects), but there are also many other knowledge artifacts that improve the effectiveness of the project. This paper presents a study of an intermediate work product, or informalism, used in a Free/Open Source Software project, GNUe. A digest-like artifact called the Kernel Cousin (KC) was used extensively in the project. These KCs allowed critical coordination and memory, but at the cost of considerable effort. The paper presents two examples of the KCs' use in the project as well as an analysis of their benefits and costs.

Categories and Subject Descriptors
H5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous, H.5.3 Group and Organization Interfaces

General Terms: Human factors

Author Keywords
Knowledge management, knowledge artifacts, free/open software systems, computer-supported cooperative work, CSCW, software engineering, online discussions.

1. INTRODUCTION
Yochai Benkler in his highly influential paper "Coase's Penguin" [4] argued for a new kind of production, which he termed commons-based peer production. He persuasively argued that many people could together create knowledge products, enabling new forms of production.

We need to know more about the kinds of new practices that lead to knowledge production in information-intensive, peer-production work. A great deal of energy has been devoted to detailing and studying the finished product of this work. For example, there are numerous studies of wikis, blogs, and more. However, peer-production work also entails forms of knowledge artifacts that appear to be transitory or even unfinished, yet are necessary to the work and lead to higher effectiveness.

These interstitial artifacts, or "informalisms," [27] are often in places that require lightweight coordination and awareness to sustain distributed work practices. These would include the coordination of work in virtual organizations [30], [21], the orientation of new members into organizations, and the social maintenance of the organization at a relatively low cost. Of particular interest, these artifacts may also serve as intermediate forms of organizational memory [2].

This paper presents a study of such a type of artifact in a virtual organization, the GNUenterprise.org (GNUe) project. GNUe's goal was to construct an enterprise resource planning (ERP) system that was free/open source software (F/OSS). The informalisms are called Kernel Cousins (KCs), a kind of community digest for summarizing discussions that are hyperlinked to source messages of important communication exchanges in a F/OSS project. The GNUe project was one of about 10 F/OSS projects worldwide that adopted and publicly posted KCs as a means for summarizing weekly discussions created and evolved by F/OSS developers who communicate through online discussions. The GNUe project utilized KCs for over 2 1/2 years, producing slightly over one hundred KC summary digests in that period.

This paper begins with a brief literature review, and then discusses the nature of F/OSS production and organization. The KCs are then introduced, along with a description of the GNUe project. This is followed with two examples of KC use. The paper then discusses the KCs' utility in the GNUe project, highlighting their coordination and memory function.

2. KNOWLEDGE ARTIFACTS AND F/OSS
In this study, we are interested in new ways of knowledge production and use, particularly in new or changing forms of social organization. As has often been stated (e.g., [22]), the two - knowledge production and use as well as social organization - often co-evolve and change together.

As such there are two bodies of research, found across many research literatures, that inform our investigation. The first involves knowledge production and use, and the second considers the particular form of social organization under examination here, F/OSS projects. (One might consider F/OSS projects as a case of a peer-based production, see Benkler [4].) We will discuss each body of research in turn below.

GROUP'07, November 4–7, 2007, Sanibel Island, Florida, USA.

The question of knowledge production and use in organizations has gone through many cycles of study and investigation. In its most recent incarnation, the question has fallen under the rubric of Knowledge Management (KM). Early KM studies emphasized technically-centered studies and often overly rationalistic or enthusiastic studies. Indeed, much of KM was a response to the introduction of new technologies (e.g., the Web and networking) to organizations. Later, KM came to examine the social and organizational issues underlying knowledge in organizations. After initial technical adoption, blind enthusiasm gave way to an appreciation of the social issues in adopting and using these new technologies (notably, Intranets). (Davenport and Prusak [6] is a useful introductory statement of the social issues.) Some KM work, such as Wenger [32] and Ackerman and Halverson [1] [2], helped foster an appraisal of knowledge management as socially situated, inherently embedded in a set of social contexts that provided the information with its value and use and also provided a social backdrop for how knowledge processes came to be and were maintained. Recent work has almost despaired for the difficulties of creating knowledge processes within conflicted or ambivalent social structures.

Throughout, there have been a handful of detailed, field-based studies that saw a more complex picture. These studies painted a picture of social considerations driving, constraining, and occasionally enabling technical adoption and use. These studies often examined specific artifacts (or families of artifacts) and their co-evolving organizational practices. These studies included Orlikowski's study of Lotus Notes and the need for a fit between prospective uses and organizational reward systems [25], [24], Palen's investigation of shared calendars and the very diverse practices centered in very small differences in systems and deployments [26], and Dourish *et al.'s* examination of two calendar/information systems and, again, differing possibilities for action based in relatively small differences in systems [8]. Of particular interest here is Halverson, Erickson, and Ackerman [16], where members of a large corporation constructed FAQs for a number of reasons, including personal, practical, and organizational considerations. This study showed that very micro-level co-evolution existed between artifacts (or rather, artifactual forms) and organizational, group, and personal practices. (Also see [3].)

All of these studies and systems, however, were created in fairly mainstream, static organizations - mainly large corporations with collocated workers working within a common management regime. These studies showed how practices and artifacts became intertwined, or failed to do so. However, what would be most interesting is to understand how new organizational forms might require new or additional practices and artifacts. In turn, we might also expect artifacts and practices to engender new organizational forms.

Of particular interest currently are F/OSS communities. As mentioned, these communities are instances of a more broad phenomenon, micro-contributory communities [4]. These communities are discretionary systems where people are able to not only regulate the amount of their contribution, they can also contribute very small amounts and still be productive members. Furthermore, the efforts of organizing these micro-contributions can be made minimal.

The next section discusses F/OSS communities as micro-contributory communities that construct software systems.

3. HOW IS F/OSS DIFFERENT?

This section discusses a brief historical account of the formation of the free software movement. It also distinguishes between free software and open source software and refers to both as free/open source software (F/OSS) as is the convention with most F/OSS researchers.

F/OSS development represents a relatively new approach to the development of complex software systems [14]. F/OSS development generally relies on a global distributed community of software developers and users who seek faster, better, and cheaper alternatives to closed proprietary systems. In most F/OSS projects, the resulting software system and its associated Web-based documents or development artifacts are globally accessible and publicly available at little or no direct cost. The terms and conditions of “copyleft” end-user licenses associated with F/OSS typically assert the following kinds of digital civil rights or "freedoms" to anyone who seeks to employ or use the software [7] [33]:

- Freedom to run the program for any purpose;
- Freedom to study how the program works and adapt it to their needs;
- Freedom to redistribute copies of the software at will;
- Freedom to improve the F/OSS program and to distribute the altered version;
- Required distribution of the originating license that specifies the freedoms and rights concerning the preceding properties.

These rights and freedoms stand in marked contrast to those offered with the selection, customization, and deployment of commercial software.

F/OSS development projects are iteratively developed, incrementally released, reviewed and refined by F/OSS developers working as peers in an ongoing agile manner. These methods ensure acceptable levels of quality, coherence, and security of system-wide software via continuous distributed peer review, testing and profiling. F/OSS efforts are hosted within decentralized communities of peers ([19] [27] [12] [28] [10] [31]) that are interconnected via Web sites and F/OSS repositories. Community oriented F/OSS development has given rise to new kinds of requirements for community building, community software, and community information sharing systems (Web site and interlinked communication channels for email, forums, and chat).

One of the key issues of importance to the free software community is its ability to manage software development without a top manager monitoring activity and passing judgement on the quality and timeliness of the work [27] [12] [28]. There is no lead organization or prime contractor that brings the alliance of individuals and sponsoring firms as a network virtual organization. It is more of an emergent organizational form where participants have in a sense discovered each other, and have brought together their individual competencies and contributions in a way whereby they can be integrated or made to interoperate [5] [20]. Most F/OSS projects are managed informally by a small group of core maintainers with one or two co-maintainers who make sure things are running smoothly. Since there is no schedule or timeline per

se, software building, testing, and releases happen sporadically and not on a strict schedule.

Membership flucutates everyday [29] from lurkers to unpaid regulars to full-time employees. In addition, the work is accomplished in a setting where many people communicate anonymously and *never* meet face-to-face to organize or manage the software contributions offered to the free software project. The core maintainers attempt to meet at conferences once or twice a year, but the bulk of the work is accomplished without direct face-to-face communication with random contributors. Thus, the participants often self-organize in a manner more like a meritocracy [15] [18] [12].

A core maintainer explains the typical method of managing the GNUe software assignments as:

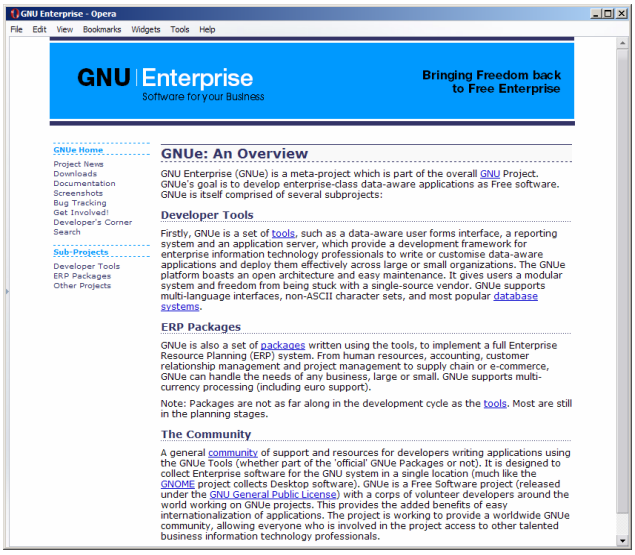
> The number one rule in free software is 'never do timelines or roadmaps'. This is a problem in open source projects. We could use a better roadmap, not having one hinders us. The features we add come about by need during consulting implementations. We may need some kind of roadmap in the future as we expand with more people. (Derek, face-to-face interview, August 2002)

4. RESEARCH SITE

The research site is a free software development community that identifies itself as GNU Enterprise (GNUe) (http://www.gnuenterprise.org). GNUe is a meta-project of the GNU Project (http://www.gnu.org). GNUe is designed to collect Enterprise software in one location on the web. The system design for GNUe consists of three items:

1. A set of tools that provide a development framework for enterprise information technology professionals to create or customize applications and share them across organizations;
2. A set of packages written using the set of tools to implement a full Enterprise Resource Planning (ERP) system; and
3. A general community of support and resources for developers writing applications using GNUe Tools. The GNUe website advertises it as a "Free Software project with a corps of volunteer developers around the world working on GNUe projects. This provides the added benefits of easy internationalization of applications. The project is working to provide a worldwide GNUe community, allowing everyone who is involved in the project access to talented business information technology professionals."

GNUe is an international virtual organization for software development [5] [21] based in the U.S. and Europe. This organization is centered about the GNUe Web portal and global Internet infrastructure that enables remote access and collaboration. Developing the GNUe software occurs through the portal, which serves as a global information sharing workplace and collaborative software development environment. Its paid participants are sponsored by one or more of twelve companies spread across the U.S., New Zealand, South America, and Europe. These companies provide salaried personnel, computing resources, and infrastructure that support this organization. However, most project participants support their participation through other means. In addition, there are also dozens of unpaid volunteers who make occasional contributions to the development, review, deployment, and ongoing support of this organization, and its software products and services. Finally, there are untold numbers



of "free riders" who simply download, browse, use, evaluate, deploy, or modify the GNUe software with little/no effort to contribute back to the GNUe community [23].

As of end of data collection in 2005, GNUe contributors consisted of 6 core developers (2 of these were listed as co-maintainers who head the project); 19 active contributors; and 19 inactive contributors. The core developers are responsible for major portions of the software development process including decisions about what software additions and modifications to include in software releases.

GNUe is a community-oriented project, as are many F/OSS development efforts [27] [31]. The project started in earnest in 2000 as the result of the merger of two smaller projects both seeking to develop a free software solution for business applications. (More information and the history of the GNUe project can be found on the GNUe Web site.) The target audience for the GNUe software modules is envisioned as primarily small businesses that are underserved by the industry leaders in ERP software, perhaps due to the high cost or high prices that can be commanded for commercial ERP system installations [28]. Many of these target companies might also be in smaller countries that lack a major IT industry presence.

Developers contributing to the ongoing evolution of the GNUe software in general provide their own personal computing resources. This is especially true for unpaid volunteer contributors, but also true of salaried participants who are paid to work on the GNUe software, particularly for their work at home. There is no standard or common personal computer configuration that is defined as the development platform, other than the requirement that a computer can run either Microsoft Windows or GNU/Linux operating systems, and that it can access the Internet or Web as needed. Thus, all GNUe community members must provide their own way into the project, via personal resource subsidies [28].

5. RESEARCH METHODS

This study is a part of a larger ongoing research project that involves a comparison of F/OSS development techniques in various F/OSS communities ([11] [10] [13] [27] [12] [28]). The

results presented in this paper are the result of an ongoing 4-year ethnography [17] of the free software movement and GNUe. The initial research questions that informed the qualitative analysis were:

1) How do people working in virtual organizations organize themselves such that work is completed?
2) What social processes facilitate open source software development?
3) What techniques are used in open source software development that differ from typical software development?

The sources of data for the study include: IRC logs from the GNUe research site; threaded email discussion messages archives; other Web-based artifacts associated with the Free Software Foundation and OSI, and the GNU project such as Kernel Cousins (summary digests of the IRC and mailing lists described in this paper); and books and articles on F/OSS. During the initial phase of research, we interpreted books and documents as well as Web site descriptions of free and open source software processes. We discovered strong cultural overtones in the readings and began searching for a site to conduct a grounded theory study of how motivations and cultural beliefs influenced the social processes of free software development. We selected the GNUe site because it represented an exemplar of an active F/OSS project and it provided archived daily IRC logs and threaded emails in addition to detailed documentation of the existing ERP software. The first author spent over 200 hours studying and perusing IRC archives and mailing list samples during the open, axial coding and analysis phases of the study.

During the open coding phase of our analysis, the first case study presented in [11] was selected as representative of the strong influence of cultural beliefs on GNUe software development practices. We discovered that for some GNUe participants, the strong belief in the development and use of free software was an idealistic motivation for joining and perpetuating the community. During this period, we also discovered the KCs Web site and began coding and studying the GNUe KCs as a cultural artifact and as an integral part of their software development practices. The KCs also provided us with categorized summaries of discussions on the IRC and email threads. These indices helped us find the examples presented in this paper.

During the second phase of the study, we performed axial coding on various samples of how the strong belief in the development and use of free software influenced software development choices and practices. Using grounded theory, we discovered that their organizational culture was instrumental in maintaining a free software community in which people volunteer hours of effort to produce free software [9]. In addition, we explored the influence of the Free Software Movement on the ideology of GNUe and its work practices. This research also includes data from email and face-to-face interviews with GNUe core maintainers and contributors, and observations at Open Source conferences. For example, we exchanged email question-and-answer sessions with GNUe core maintainers and conducted a lengthy interview with one of the GNUe core maintainers at an Open Source conference in 2002.

In the later stages of the research, we chose to study the KCs in depth as an artifact that contributes to the coordination of software development in a virtual distributed community. The main GNUe KC writer, Peter Sullivan, also answered several email questionnaires giving us detailed explanations of the importance of the KCs and of his role as a KC writer.

6. KERNEL COUSINS – WHAT ARE THEY?

Specifically, this study examines Kernel Cousins (KCs), a type of knowledge artifact used as an informalism – a seemingly transitory artifact that is actually quite important in creating effective peer production --in the GNUe project. As one might expect from F/OSS communities, they were adapted for the GNUe project from the efforts of other F/OSS projects. The KC Web site (www.kernel-traffic.org) contains a set of web-based newsletters from various free software projects. The newsletters serve as a summary of the weekly activity on the software development computer-supported communication (CMC) tools: IRC and mailing list archives. Started in 1999 by Zack Brown, a Linux Kernel developer and editor of Linux Today, the Kernel Traffic newsletter summarizes the activity on the main Linux kernel software development mailing list. As stated by Zack Brown on the current KC Web site, "On it, Linus Torvalds, Alan Cox and a lot of other amazing programmers from around the world share patches, argue about implementation details, discuss the news of the day, and generally make history."

By September 1999 the Kernel Traffic web site had expanded to include newsletters from other free software projects: Wine, Hurd, Gimp, and Debian. To distinguish these newsletters from the original Linux Kernel one, Zack Brown named these Kernel Cousins, and established a standard format for all KCs hosted on the www.kernel-traffic.org site. From 1999 to the present, KCs have mainly been initiated and maintained by volunteers. When a KC volunteer author is no longer available and a substitute KC writer cannot be found, then the KC goes to "sleep," listed on the newsletter site as a "sleeping cousin". However, the archives of previous publications persist and are still publicly available for browsing, search, or download. Once someone volunteers to continue writing a KC for a specific project again, that sleeping cousin becomes active again.

As of 2005, there were only two active KCs: 1) the Wine project – a project to create a Microsoft Windows emulator running on Unix platforms, and 2) git – a recently formed project to design and develop a new revision control system for use in the Linux Kernel project. Zack Brown wrote the summaries for Linux (which consists of as many as 3000 messages per week), until he stepped down in November 2005.

Table 1 shows the active and sleeping KCs that were listed on the www.kernel-traffic web site and their status as of late 2006.

Project	Status	#KCs	Start Date	End Date	# Contributors Quoted in KCs
Linux	Sleeping	335	1/99	11/05	1929
Wine	Sleeping	315	6/99	7/06	690
git	Sleeping	1	5/05		48
GNUe	Sleeping	125	10/01	9/06	257
KDE	Sleeping	76	3/01	4/04	420
GIMP	Sleeping	44	6/99	5/01	124
Debian	Sleeping	28			
Samba	Sleeping	40	11/99	2/01	298
SLUG Pearls	Sleeping	7	6/00	6/00	58

Table 1 – Statistics on Existing/Sleeping Cousins

6.1 KC Technical Specifications

The newsletters are written in XML using a makefile and XSLT recipe files along with a few varied scripts made available for download from the FSF website. Using the KC makefile and scripts enforces a uniformity to the newsletter formats enabling standardized conventions such as a list of all contributors quoted in the newsletter with the number of times they were quoted. All KCs have color-coded text to represent authors, exact quotes, contributors' names who have been quoted, and hyperlinks pointing back to the originating source of IRC and mailing lists which the KC summarizes. See Figures 1 and 2 for examples. Unfortunately, we cannot reproduce the use of color and hyperlinks here, we instead substitute **boldface** type.

The makefiles are set up with various indices which are described below. We show in the GNUe KC section how these indices help people search through the KCs for summaries related to specific technical topics. Each KC is divided into numbered sections with titles and where appropriate, the sections link to a list of related sections from other KCs. In this way, KC readers can select a topic and read a historical account of all threads related to that topic: Topics, Contributors, Index of TOC for each KC, KC Archives, and Authors. The GNUe community used these links to group sections of the KCs into topics related to their main modules: Application Server, Forms, Reports, etc.

7. GNUe KCs

The KCs were used by GNUe developers during a period of November 2001 to September 2006. One can consider the KC newsletters as "reflective" documentation since they are subjective essays written by volunteers as a weekly summary of IRC logs and mailing list threads. There is no direct supervision as to the content, number of threads to include, or quality of communication to include in the weekly KC newsletter. Thus, the KC entries become a cultural artifact reflecting the writer's personal preferences and style of writing. As a cultural artifact, the KCs serve to reify the sensibility of contributing to free software projects and to immortalize each contributor's name in a digital artifact. In addition, when active, people reading the KCs would not have to waste inordinate amounts of time reading IRCs.

During the period from 2000 to 2002, we interviewed one of the GNUe co-maintainer and core developer, Derek Neighbors, several times. We asked him general questions about software development and management of free software developers. Generally, he was enthusiastic about the GNUe project, and was eager to correspond with us. When asked about software development management without knowing the developers on a personal level, he responded with

> Actually IRC works well for us as a surrogate "office", people are regulars for the most part in their 'hours'. So you learn to expect person X to be in the office around xxxx (irc) and such. It is logged [recorded] unlike an office so other contributors can catch up if they miss work. ;).

The KC served the purpose of summarizing the discussions in this surrogate office so that people would not have to waste inordinate amounts of time reading IRCs. In fact, most of the work in GNUe takes place on the IRC instead of email or phone calls:

> Oddly the core really prefers IRC. We do exchange email a bit, but generally only for those that require it or ask for it, as email seems to slow us down. It's too traditional...Many free software folks think IRC is a waste of time as there is 'goofing off', but honestly I can say its what builds a community. I think a community is necessary to survive...I put our longevity [3 years at the time of the interview] solely to the fact that we have a community.

Therefore, recording the GNUe activities in a KC instantiates GNUe developers forever and provides a useful chronicle of software development activity. As well, they are "mainly for folks who can not devote time to reading IRC logs or can't access IRC real time, but want a high level view of what is going on.... It's the only way many can see what's happening" (Derek).

The main author of the KCs, Peter Sullivan, also provided us with detailed information and personal observations regarding experience with the KC in the GNUe project. While writing the KCs, he did receive positive feedback on their usefulness to GNUe developers. For the core developers who were always on the IRC, the main advantage was to catch up after being out of town. For others, the KCs served to inform more "fringe" users of activities without having to read the full logs. Peter suggested that now that KCs are gone, he noticed that people use email more and IRC less of the time. He believes this either the culture of the GNUe project is changing without the KCs to more of a mailing list work environment, or the new type of GNUe contributors prefer mailing list activity to IRC.

He indicated that he had to step down as author of GNUe KCs due to time constraints caused by a new job (for which he was paid, unlike the volunteer work for GNUe). Writing KCs could take anywhere from 5 minutes to confirm no activity to about 2-3 hours a day for periods of "heavy" IRC and mailing list activity. Interestingly, even though Peter was at will to write as little or as much as he wished, he chose to try and cover all activities. Peter was not a GNUe developer; however, he was able to write the KCs as though intimately involved with GNUe's Python code. He

4. Using non-free tools for GNUe Documentation
14 Nov 2001 - 17 Nov 2001 (10 posts) Archive Link: "Can We Please Not Use LyX!?!?"
People: Daniel Baumann, Michael Brown, Jason Cater, Neil Tiffin, Chad Walstrom, Derek Neighbors

Further to Issue #3, Section #12, Daniel Baumann questioned the use of LyX for GNUe Documentation. He realised that people were having problems with installing docbook, but said **"I can build html versions of stuff on my box if this is what we have to do."** He added **"I really shouldn't have to be harping on this issue for a GNU project, but some ppl like to take convience over freedom and this should not be tolerated. "** Michael Brown suggested **"why not juile LyX with the QT2 toolkit? According to http://www.devel.lyx.org/guii.php3 , this is pretty close to being complete."** Daniel said he had tried the QT port before, and was not impressed. Jason Cater said **"The upcoming release was originally planned for this past weekend.."** They had used LyX because they had had problems trying to get docbook to install. Neil Tiffin said **"I would much rather have docs in any format than no docs at all."**

Daniel reiterated his position, and said " I feel that this issue needs to be ironed out and I apologize for the prior language, but I am very frustrated and I feel alienated" . Derek Neighbors said "I think we have always had the stance of 'docbook' is the 'preferred' format of documentation, however we will take documentation in ANY format. I know I have personally accepted text files, texinfo and word documents and converted them to docbook. This motto is because we know its hard to find documentors so we wont term them away REGARDLESS of their tool." Daniel proposed "that we use text for now and I will volunteer to do docbook or we just switch to something that works better on all systems like say textinfo. What do you guys think?"

There were also full and frank discussions of this issue IRC on 14th, 15th and 16th November.

Figure 1: The Kernel Cousin about using F/OSS tools in the GNUe project

claims this familiarity came from "inferring things from the context of the discussion, or doing some limited research on the web, or even just asking!"

Next we present the two examples of how KCs in order to ground our analysis of them.

7.1 GNUenterprise KC Examples

Below, we present two examples of GNUe software development work that are summarized in sections of a KC. The first case involves a discussion of whether or not a free software community should use non-free software tools to develop GNUe documentation. The second case is more of a technical nature and includes a newcomer who engages the GNUe developers with his skill in fixing GNUe bugs rather quickly. This case illustrates the dynamics of micro-contributions since the newcomer makes a significant contribution in a short period of time without ever meeting the core maintainers face to face. There were 169 people who were quoted once or twice during the three year period, from 2001-2004, 64 who were quoted 3 to 26 times, and 4 maintainers who were quoted 1835 times in total. The bulk of the work appears to have been performed by the top four maintainers. However, dozens of other people, like mc380 in our second example, donate smaller amounts of individual time and effort than the core maintainers yet make significant contributions as a whole. KC Example 1 - Saves time for developers

Figure 1 shows the KC for the first example. It illustrates how the KC documents a three-day debate which occurred on both mailing lists and IRCs in which contributors debated the issue of using non-free tools to develop GNUe documentation. In this example, Chillywilly, a frequent contributor, balks at the need to install a non-free tool on his computer in order to edit documentation associated with a current release. Even though his colleagues attempt to dissuade him from his concerns by suggesting that he can use any editor – free or non-free – to read the documentation in HTML or other formats, Chillywilly refuses to back down from his stance based on a strong belief in free software. This debate lasts three days. Since the three IRC archives also include interactions among contributors on several other issues as well, the KC gives interested readers a clear update on an issue that would otherwise have required reading lengthy IRC archives.

The KC has been excerpted for space reasons (see Figure 1 below). Note that there are there is a way to link back to other discussions; for example, the first line where there is a hyperlink back to KC Issue #3.

Next we show a very small portion of the IRC log for the November 14, the first day of the debate. Chillywilly announced on the IRC that a fellow collaborator, jamest, had made documents with lyx and questions the appropriateness of using lyx since it requires the installation of non-free software. The following IRC excerpts have been changed slightly for readability:

```
Action:  chillywilly trout whips jamest for
  making lyx docs
Action:  jcater troutslaps chillywilly for
  troutslapping jamest for making easy to do
  docs
<Chillywilly> lyx requires non-free software
<Maniac>  lyx rules
<Chillywilly>  should that be acceptable for a
  GNU project?
<Maniac>  chillywilly: did he type it on a
  non-free computer?
<Mr_You>  heh
<Chillywilly> Maniac: you make no *** sense
<Maniac>  :)
<jcater>  Chillywilly: basically, given the
  time frame we are in, it's either LyX
  documentation with this release, or no
```

```
  documentation for a while (until we can get
some other stinking system in place)
<jcater>  pick one :)
<Chillywilly> use docbook then
```

This discussion continued into November 15 and evolved into a discussion of the problems with docbook as well.

```
<Maniac> chillywilly: so GNU projects cannot
  use non-GNU software in any portion of their
  project?
<Chillywilly> no, they shouldn't use non-free
  software
<Chillywilly> libxforms would reqwuire me to
  add non-free section to sources.list
<Chillywilly> thus I will not do it and cannot
  read the damn docs
```

A lengthy and heated discussion ensued on the IRC. Actually, the document was also available in html and text format, so Chillywilly could easily have read the documentation. Other developers with a more moderate view about the sole use of free software criticized his argument regarding lyx. Even though they agreed that Chillywilly was being unreasonable, several participants agreed with his philosophy. Chillywilly continued to argue, however, that the installation of lyx did not match his philosophical orientation toward free software development. Chillywilly ended this conversation with an exclamation that the lyx is "evil" software.

```
<jamest> however the people that are willing to
  put the effort into the user_guide and
  tech_ref (jcater and myself)
<jcater> **** it all... read the source code!
<jamest> are sick of fighting docbook
 [...]
<Chillywilly> I casn read that wihout being
  installing evil software
<Chillywilly> s/being//
```

The conversation continued with a lengthy discussion of technical issues unrelated to the documentation problem. Meanwhile several people sent emails about the documentation fight to the distribution list. This distribution list, as mentioned, is much more public. One of the other GNUe developer's emails included:

```
I would like to personally apologize to the
discussion list for the childish email you
recently received. It stemmed from a
conversation in IRC that quickly got out of
hand.  It was never our intention to alienate
users by using a non-standard documentation
format such as LyX.
```

The conversation continued with and without Chillywilly regarding lyx usage. The next day, Chillywilly broached the subject of jcater's response to Chillywilly's email regarding lyx.

```
Action: chillywilly sees jcat **** and moaning
  about my mail.
<Chillywilly> jcater
<ajmitch> no, it doesn't look like much of a
  **** & moan to me
<jcater> huh?
<Chillywilly> irc logs shows otherwise
```

As might be expected, Chillywilly stuck to his diehard view without any interest in switching to non-free software.

Finally, Chillywilly was convinced to drop the issue for the present. Mr_You appealed to Chillywilly to not let his philosophy impede progress, and jcater suggested that constant bickering looks bad for the GNUe project. Chillywilly still insisted that needing to install non-free software is a huge impediment to developers, yet finally he dropped the issue.

```
 <Mr_You> as time goes on, we can move to
  another solution ...
<Mr_You> I realize it goes against your
  philosophy.. but philosophy shouldn't get in
  the way of progress if it is a temporary
  issue
<Mr_You> its just a minor temporary issue in
  the huge scheme of things
<Mr_You> I don't think it threatens our
  integrity
<Chillywilly> why not just do it in text and
  then mark it up later then everyone csn read
  development docs without the B.S.
<Mr_You> go for it chilly.. you have your valid
  reasons
<Chillywilly> Mr_You: the easy way to do that
  would be to run lyx and copy and paste I will
  not install it again until I can run it
  easily with a Free GUI
<Mr_You> you just haven't been successfull in
  convincing others at this time, I have no
  doubt you may be able to in the future, as
  everyone agrees with you ideally but
  technically its a minor issue
<Chillywilly> it is not *minor*...it makes us
  look bad
<jcater> chillywilly: emails like what you sent
  make us look bad
```

As mentioned, GNUe readers would have had to read through roughly 34 pages of the IRC logs to fully read the discussion summarized in approximately 4 paragraphs of a KC. Of course, there would have been many extraneous comments and other conversations in that IRC log, which would have made using it even more difficult. One of the standard problems of IRC is rereading a log file to catch up.

What may have been difficult to see in the printed version, the KC includes links to the relevant portions of the IRC conversation, so interested parties can review those relevant conversation fragments.

7.1.1 Example 2 – GNUe KC index

Figure 2 shows the KC for the second example. It illustrates how the KC can serve as an index and summary of complex technical discussions. This particular discussion thread becomes linked to a larger set of mailing list threads called the Application Server. In this example, a newcomer, mcb30 or Michael Brown, joined the IRC and requested CVS (archival) access after pointing out bugs which he had fixed during his GNUe installation. He was a consultant who wanted to use the GNUe software to help him run his small business in England. This example reflects the sporadic software development that results in substantial code fixes and design modifications from an infrequent contributor. Mcb30 was quickly accepted by frequent contributors especially because he posted significant bug fixes very rapidly. However, his name is listed in the KC contributor list only seven times so he did not contribute to the project on a long time basis.

The story of the KC here is somewhat complicated; we simplify it here for publication purposes. On November 16, mcb30 got on the IRC and asked for information regarding the use of GNUe. He

had a conversation on the IRC with reinhard about GNUe and bug reporting:

```
<mcb30> Is anyone here awake and listening?
<reinhard> yes
<mcb30> Excellent.  I'm trying to get a CVS
  copy of GNUe up and running for the
  first(ish) time - do you mind if I ask for a
  few hints?
<reinhard> shoot away :)
<reinhard> btw what exactly are you trying to
  run?
[...]
<mcb30> OK - what I want to do is get
  *something* running so I can get a feel for
  what there is, what state of development it's
  in etc. - I'd like to contribute but I need
  to know what already exists first!
<reinhard> ok cool
<reinhard> let me give you a quick overview
<mcb30> I have finally (about 5 minutes ago)
  managed to get "setup.py devel" to work
  properly - there are 2 bugs in it
<mcb30> I've got a patch file - who should I
  send it to?  jcater?
<reinhard> jcater or jamest
<mcb30> ok, will do, thanks
```

Over the course of the next three days, mcb30 made changes to the software and tested it. Each day he sent messages to the mailing list and IRC while working offline on the GNUe software bug fixes. Some of what he did is summarized in the KC for November 16 through November 19.

Throughout four days of detailed discussions on the IRC and mailing lists, mcb30 contributed code and design ideas related to the module. Finally, after testing these code modifications, he suggested to the core contributors on the mailing list that he should have “commit” access so that he could submit the code for the next release. He was told that he needed to sign a copyright form with the Free Software Foundation (FSF). Accordingly, mcb30 asked for copyright assignment requirements on the IRC and then he was directed to the mailing list. The IRC exchange, with all of its noise, includes the following:

```
<mcb30> in the midst of all this, can someone
  tell me how I go about doing the necessary
  steps to get CVS access (copyright assignment
  or whatever
<codewind> well first you need the software i
  presume thats in order :)))))))))
<jcLunch> mcb30: send me an email to
  jcater@gnue.org and I will reply with the
  first steps of copyright assignment
<dneighbo> mcb30 you can also send to
  info@gnue.org so have your email and we can
  send you the documents to get started
<dneighbo> dont let it hold up you doing work
<dneighbo> we just cant put into the cvs tree
  until we get things squared away
[...]
<mcb30> jcater: will send mail to you and
  info@gnue.org
```

This continued on the mailing lists. Mcb30 said in an email:

```
[...] I can't commit the code until I get CVS
write access (I'm waiting on the FSF for the
copyright assignment forms), but thought you
might like to know that it works. ...  I've
done a few significant modifications in
methods/, most of which was pulling some code
out of {python,glibmodule}_methods.c and into
methods.c.  I think I've actually ended up
reducing the total number of lines.  Dynamic
loading of methods is what I'd like to attack
next, but I'm not going to start coding
anything else until I've got the work I've
already done checked in.

How long does this copyright assignment thing
take?

Michael
```

A core developer answered saying that he thought the copyright assignment process took a week, but might be speeded up. This was corrected by one of the core developers of the project:

```
[...] Send via patch file to info@gnue.org.
Generally we dont give CVS access immediately
even after assignment is done.  We used to,
but the problem is sometimes someone will do
assignment make a patch or two then
disappear.  ...The FSF Clerk [...] in Boston,
MA. [...] will draw you up 'physical' papers
and snail mail them to your address.  Where
you sign them and return them via snail mail
and they are put on file.  So generally it
takes at least 2 weeks.
```

Mcb30 responded:

```
OK, but you do realise that this sort of
thing creates a *huge* barrier to attracting
new developers?  If it wasn't for the fact
```

10. FSF copyright assignment and CVS access
16 Nov 2001 - 17 Nov 2001 (6 posts) Archive Link: "**[gnue-geas] Method loading**"
People: **Michael Brown**, **Reinhard Müller**, **Derek Neighbors**

Michael Brown confirmed "I have GEAS working with a mixture of python and glibmodule methods loaded simultaneously." He asked how long FSF copyright assignment would take. Reinhard Müller said "it usualy takes more than a week altogether" , but he would ask Derek Neighbors to expediate. He would also like to list Michael's company as a project sponser. Derek Neighbors asked Michael to e-mail the patches - "Generally we dont give CVS access immediately even after assignment is done." Michael said he thought "this sort of thing creates a huge barrier to attracting new developers" . Derek also said copyright assignment was normally done by snail mail, but "If we wanted to put a huge rush and you have a fax it could be arranged" . Michael supplied his fax number.

Figure 2: The Kernel Cousin about being able to submit code and having CVS access in GNUe

```
that I've already written a chunk of code and
become involved, this type of disincentive
would probably have been sufficient for me to
think "too much hassle to join in, may as
well just come back in six months to see if
they've got any further writing it yet".
```

The KC summarizes this entire exchange (which we have summarized in our account as well).

After the copyright discussion on the mailing list, mcb30 continued to show up sporadically in IRC, but did not continue his efforts on GNUe. Nonetheless, the KC continues to serve as a summarization of how to obtain commit access and the copyright issues involved in that access.

8. DISCUSSION

In the above two examples, one can see that the KCs can lead to the rapid assimilation and condensation of sprawling conversations. This had three major benefits for the virtual organization. First, it provided peripheral attention and participation to members of GNUe. People did not have to pay attention to details in many discussion messages, as the important conversations would be summarized - allowing members to note that they existed as well as the gist of the discussion. It became even more useful after vacations or other periods away from the project. If necessary or important, members could follow the relevant links to recover the entire conversation.

Second, people who were not central to the effort could maintain a low-level interest in the project. As mentioned, one of the central uses of the KCs was to inform managers (of interested companies or of the participants themselves) and others who needed to monitor but not fully engage the project. KCs enabled standard organizational practices to interoperate with the GNUe work practices. Third, the KCs reinforced important decisions. They were written down, and therefore they became remarkable, as well as indexible, navigable, and searchable. One could easily retrace them, making it possible to quickly go through the important design decisions on an ongoing basis. As such, they served as an important, informal design rationale or design memory.

While the KCs were a form of digest, this informalism, then, enabled critical peripheral awareness and peripheral participation, either directly (through memory or background attending) or indirectly (through managers' being able to casually monitor). In a highly distributed, volunteer project, this is critical. KCs enabled more people to attend and engage as they could, thereby enabling the participation of a larger group of interested people.

KCs were not perfect, however. One of the issues that we found most interesting was KCs' dependency on a small number of people, and in GNUe on only one person. Attempts to turn the KC production over to others or to distribute it among a small group were not successful, revealing several important issues. The KCs were time-intensive, and they required someone who was technically strong. One had to understand deeply the technical issues involved. In most F/OSS projects, people with that capability are cutting code, as this is more socially and vocationally central. Therefore, the KCs required someone who was technically adept, had time, and was relatively unconcerned about the social centrality of code production. This is rare, and hard to cultivate. This may also be why the KCs, while serving as a form of design rationale, were not further distilled into formal documentation.

As well, it is quite possible that KCs as well are important at a specific time in a F/OSS (or other) project. A strong possibility is that KCs are best used when the project is being intensively developed. There is great coordination and communication need, and it is hard to stay caught up. Phone conferences are one way that handles the need for constant coordination, but KCs provide a more durable and easily retrieved/searched memory. KCs are also lighter-weight than phone conferences for casual participants, as they do not require the synchronous participation of phone conversation; they allow background attending and peripheral awareness. When the project is more stable or slower-paced, KCs may be less valuable. We cannot know this with certainty, however, without further study.

9. CONCLUSIONS

The use of KCs in the GNUe project illustrates use of seemingly transitory knowledge artifacts as informalisms to enhance coordination, help social and project maintenance, and maintain a shared understanding and identity within a complex, distributed project. GNUe is a classic F/OSS project that maintained an identity centered in its code production. However, we could expect that other virtual organizations would require similar artifacts, able to focus the organizational participants on the important activities and their histories in a timely and efficient manner. These artifacts might have little value or use after these activities end. Nonetheless, their importance in organizational effectiveness, and especially in peer production, should not be underestimated. Indeed, the GNUe KCs were useful and important in the ongoing work of the project in a manner that reiterated the value of collaboration through peer production.

Obviously, if the KCs were important, why were they not continued? Although we made a significant effort to determine this, the answer remains murky. We believe that a fair amount of the answer lies in the unique characteristics of the main author of the KCs, Peter Sullivan in GNUe, or Zack Brown in Linux Kernel. Peter (and Zack) was technically skilled, was able to summarize well, was willing to write, personable, had enough time, and was able to engage in deeply technical discussions of source code functionality or structure without engaging in source code development. These are rare skills in F/OSS projects where participants primarily contribute to the project on their own time. (One could imagine that they would be less rare in organizational projects; the central vocational anchor in F/OSS projects is technical expertise.) Nonetheless, we could envision several ameliorations. Training, in places like Information or Informatics schools, may enable these conjoined skills to be more readily available. Additionally, one could imagine that if one of the major issues with KCs or other similar forms of knowledge artifacts is sheer time, technical augmentations to facilitate distillation [2] might ease the problem. Production of the KCs required laborious cutting and pasting, as well as linking, by hand. Clearly, this is a place where relatively straightforward tools would help. One could even imagine distributed tools that allowed project participants to mark important sections of IRC or email discussions, expediting KC authoring.

10. ACKNOWLEDGMENTS

The research described in this report is supported by grants #0083075, #0205679, #0205724, #0350754, #0534771, and 0325347 from the U.S. National Science Foundation. No endorsement implied. Les Gasser at UIUC; John Noll at Santa

Clara University; and, Chris Jensen and others at the UCI Institute for Software Research are collaborators on the research described here.

11. REFERENCES

[1] Ackerman, M. S. and Halverson, C. Considering an Organization's Memory. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'98)* (1998), 39-48.

[2] Ackerman, M. S. and Halverson, C. Organizational Memory: Processes, Boundary Objects, and Trajectories. *Computer Supported Cooperative Work: The Journal of Collaborative Computing*, 13, 2 (2004), 155-189.

[3] Ackerman, M. S., Halverson, C. A., Erickson, T. and Kellogg, W. A. (ed.) *Resources, Co-Evolution, and Artifacts: Theory in CSCW*. Springer, New York, 2007.

[4] Benkler, Y. Coase's Penguin, or Linux and the Nature of the Firm. *Yale Law Journal*, 112(2002), 369+.

[5] Crowston, K. and Scozzi, B. Open Source Software Projects as Virtual Organizations. *IEE Proceedings--Software*, 149, 1 (2002), 3-17.

[6] Davenport, T. H. and Prusak, L. *Working Knowledge: How Organizations Manage What They Know*. Harvard Business School Press, Boston, 1998.

[7] DiBona, C., Ockman, S. and Stone, M. *Open Sources: Voices from the Open Source Revolution*. O'Reilly Media, Sebastopol, CA, 1999.

[8] Dourish, P., Bellotti, V., Mackay, W. and Ma, C.-Y. Information and context: lessons from the study of two shared information systems. In *Proceedings of the ACM Conference on Organizational Computing* (1993), 42-51.

[9] Elliott, M. *The Virtual Organizational Culture of a Free Software Development Community*. Paper presented at the 3rd Workshop on Open Source Software, Portland, Oregon, 2003.

[10] Elliott, M. and Scacchi, W. Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration. In *Proceedings of the ACM Conference on Supporting Group Work (Group 2003)* (2003), 21-30.

[11] Elliott, M. and Scacchi, W. *Free Software: A Case Study of Software Development in a Virtual Organizational Culture*. Technical Report No. UCI-ISR-03-6, Institute for Software Research, University of California, Irvine, 2003.

[12] Elliott, M. and Scacchi, W. Free Software Development: Cooperation and Conflict in a Virtual Organizational Culture. In S. Koch (ed.) *Free/Open Source Software Development*. Idea Publishing, New York, 2005, 152-172.

[13] Elliott, M. and Scacchi, W. Mobilization of Software Developers: The Free Software Movement. *Information Technology and People*, in press.

[14] Feller, J. and Fitzgerald, B. *Understanding Open Source Software Development*. Addison-Wesley, New York, 2002.

[15] Fielding, R. T. Shared Leadership in the Apache Project. *Communications of the ACM*, 42, 4 (1999), 42-43.

[16] Halverson, C. A., Erickson, T. and Ackerman, M. S. Organizational issues: Behind the help desk: Evolution of a knowledge management system in a large organization. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work* (2004), 304-313.

[17] Hine, C. *Virtual Ethnography*. Sage Publications, Newbury Park, CA, 2000.

[18] Jensen, C. and Scacchi, W. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study. In *Proceedings of the International Conference on Software Engineering (ICSE)* (2007).

[19] Kogut, B. and Metiu, A. Open Source Software Development and Distributed Innovation. *Oxford Review of Economic Policy*, 17, 2 (2001), 248-264.

[20] Ljungberg, J. Open Source Movements as a Model for Organizing. *European Journal of Information Systems*, 9, 4 (2000), 208-216.

[21] Noll, J. and Scacchi, W. Supporting Software Development in Virtual Enterprises. *J. Digital Information*, 1, 4 (1999).

[22] O'Day, V. L., Bobrow, D. G. and Shirley, M. The Social-Technical Design Circle. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'96)* (1996), 160-169.

[23] Olson, M. *The Logic of Collective Action*. Harvard University Press, Cambridge, 1971.

[24] Orlikowski, W. J. The Duality of Technology: Rethinking the Concept of Technology in Organizations. *Organization Science*, 3, 3 (1992), 398-427.

[25] Orlikowski, W. J. Learning from Notes: Organizational Issues in Groupware Implementation. In *Proceedings of the Computer Supported Cooperative Work* (1992), 362-369.

[26] Palen, L. Social, individual and technological issues for groupware calendar systems. In *Proceedings of the ACM Conference on Human Factors in Computing Systems* (1999), 17-24.

[27] Scacchi, W. Understanding the Requirements for Developing Open Source Software Systems. *IEE Proceedings--Software*, 149, 1 (2002), 24-39.

[28] Scacchi, W. *Understanding the Development of Free E-Commerce/E-Business Software: A Resource-Based View*. In S. K. Sowe, I. Stamelos and I. Samoladas (ed.) *Emerging Free/Open Source Software Practices*. IDEA Group Publishing, Hershey, PA, 2007, 170-190.

[29] Scacchi, W. and Jensen, C. Guiding the Discovery of Open Source Software Processes with a Reference Model. In *Proceedings of the Third IFIP International Conference on Open Source Systems* (2007), 265-270.

[30] Schmidt, K. and Simone, C. Coordination Mechanisms: Towards a Conceptual Foundation of CSCW Systems Design. *Computer Supported Cooperative Work Journal*, 5, 2/3 (1996), 155-200.

[31] Sharma, S., Sugumaran, V. and Rajagopalan, B. A Framework for Creating Hybrid Open-Source Software Communities. *Information Systems Journal*, 12, 1 (2002), 7-25.

[32] Wenger, E. *Communities of practice: learning, meaning, and identity*. Cambridge University Press, New York, 1998.

[33] Williams, S. *Free as in Freedom: Richard Stallman's Crusade for Free Software*. O'Reilly Books, Sebastopol, CA, 2002.