Group Awareness in Distributed Software Development

Carl Gutwin, Reagan Penner, and Kevin Schneider
Department of Computer Science, University of Saskatchewan
57 Campus Drive, Saskatoon, SK, Canada, S7N 5A9
+1 306 966-8646
carl.gutwin, reagan.penner, kevin.schneider @ usask.ca

ABSTRACT
Open-source software development projects are almost always collaborative and distributed. Despite the difficulties imposed by distance, these projects have managed to produce large, complex, and successful systems. However, there is still little known about how open-source teams manage their collaboration. In this paper we look at one aspect of this issue: how distributed developers maintain group awareness. We interviewed developers, read project communication, and looked at project artifacts from three successful open source projects. We found that distributed developers do need to maintain awareness of one another, and that they maintain both a general awareness of the entire team and more detailed knowledge of people that they plan to work with. Although there are several sources of information, this awareness is maintained primarily through text-based communication (mailing lists and chat systems). These textual channels have several characteristics that help to support the maintenance of awareness, as long as developers are committed to reading the lists and to making their project communication public.

Categories and Subject Descriptors
H.5.3 [**Information Interfaces and Presentation**]: Group and Organization Interfaces – *Computer-supported cooperative work*

General Terms
Design, Human Factors.

Keywords
Collaborative software development, group awareness, OSS.

1. INTRODUCTION
Software development is one real-world situation where work regularly happens in a distributed fashion. Open-source software (OSS) is particular in this regard – these development projects have numerous programmers from many different parts of the world, who rarely if ever meet face-to-face. Despite the difficulties imposed by distance, many OSS projects have managed to produce large, complex, and successful software systems (such as the Linux operating system, the Apache web server, or the OpenOffice application suite).

CSCW'04, November 6–10, 2004, Chicago, Illinois, USA.

Although OSS groups have been studied in the past (e.g., [7,18]), there is still little known about exactly how they manage to overcome the barriers to coordination that are imposed by distance. For example, Mockus and colleagues ask:

> One worry of the 'chaotic' OSS style of development is that people will make uncoordinated changes, particularly to the same file or module, that interfere with one another. How does the development community avoid this? [18, p. 312]

In this paper, we look at one aspect of this issue – how distributed developers maintain group awareness. Group awareness information includes knowledge about who is on the project, where in the code they are working, what they are doing, and what their plans are. This knowledge seems vital if distributed developers are to coordinate their efforts, smoothly add code, make changes that affect other modules, and avoid rework.

We carried out a study of open source teams to determine whether developers need to stay aware of one another, what awareness information developers keep track of, and how they gather and maintain their knowledge. We interviewed fourteen developers on three different well-established OSS projects, examined email and chat archives, and analysed project artifacts such as source-code repositories, web pages, and official project documentation.

We were surprised by our results. First, we expected that projects would be set up to reduce awareness requirements, with each software module carefully partitioned and protected from others. However, we found that official partitioning is limited, and that developers can contribute to any part of the code – an organizational approach that increases awareness requirements.

Second, we found that the developers were able to maintain a good general awareness of other developers and their activities, and were able to find more detailed information about people's activities when they needed to. However, we were surprised that the main mechanisms for maintaining group awareness were simple text communication tools – developer mailing lists and text chat. Since these tools are disconnected from the project artifacts, and because they require explicit effort, we expected them to provide only incidental awareness – but in all three projects they were the main source of information.

When we looked more closely at the email and chat messages, we found that these text channels have a number of characteristics that are valuable for the provision and collection of awareness information. First, they are public, and so allow all the developers on the list to become peripheral participants in each others' conversations. By overhearing others and by seeing who is talking about what, developers can gather important group awareness information. Second, mailing lists allow people to find out who the experts are in an area, simply by initiating a discussion: because the messages go to the entire group, the 'right people' will identify themselves by joining the conversation.

These awareness mechanisms can only work if most of the discussion between developers happens on the public channels – and we found that there are strong elements of organizational culture on these projects that do just this. In particular, there is a strong culture of 'making it public' where developers are willing to answer questions, discuss their plans, report on their actions, and argue design details, all on the mailing list.

Our findings provide details of how one kind of real-world distributed group maintains awareness and manages coordination, and exposes some of the underlying mechanisms that allow developers to overcome the problems of distance. We were impressed that ordinary verbal communication could be so effective in supporting awareness and coordination – particularly when the discussions so often refer to work artifacts that are not represented in the communication system.

Although not all work settings are similar to open source development, we believe that our findings can assist analysis of awareness in other distributed work situations, and that the principles of awareness on OSS projects can benefit other types of computer-supported distributed work. Our study also suggests that groupware designers should tread carefully when inventing tools for distributed software development. Although visualization systems can provide a variety of awareness information – such as integrating information about artifacts and activities – these tools must ensure that they do not 'siphon off' public interaction from communication channels that helps the entire group to stay aware.

2. AWARENESS IN DISTRIBUTED WORK

In many group work situations, awareness of others provides information that is critical for smooth and effective collaboration. Group awareness is the understanding of who is working with you, what they are doing, and how your own actions interact with theirs [6]. Group awareness is useful for coordinating actions, managing coupling, discussing tasks, anticipating others' actions, and finding help [10]. The complexity and interdependency of software systems (e.g., [16]) suggests that group awareness should be necessary for collaborative software development.

Three mechanisms help people to maintain awareness in co-located situations: explicit communication, where people tell each other about their activities; consequential communication [25], in which watching another person work provides information as to their activities and plans; and feedthrough [5], where observation of changes to project artifacts indicates who has been doing what.

Although group awareness is taken for granted in face-to-face work, it is difficult to maintain in distributed settings. Studies of distributed work have shown that much of the communication and implicit information that is available to a co-located team does not exist for remote collaborators [13,22]. For example, Herbsleb and Grinter [13] found that lack of ad-hoc communication between software developers caused an increase in coordination problems and a decrease in collaboration between remote sites.

Of the three mechanisms stated above, awareness through explicit communication is the most flexible, and much research has been carried out on how groups communicate over distance, through newsgroups, MUDs, email, text chat, and instant messaging (e.g., [4,8,27,28]). However, since intentional communication of awareness information also requires the most additional effort, many awareness systems attempt to support the implicit mechanisms as well as communication. General approaches include providing visible embodiments of participants and visual representations of actions that allow people to watch each other work (e.g., [3]), and overview visualizations that show authorship and changes to the project artifacts (e.g., [14]). Although there are not many awareness displays specifically for software teams, these approaches have led to visualization systems such as Augur [9], Tukan [24], and editors that allow observation of others [6].

These systems are not in wide use, however, and there is still little known about what awareness techniques and mechanisms are in use in the widely-distributed development environment of an open source project. In the next section, we describe the projects that we studied and the methods that we used to investigate this issue.

3. PROJECTS AND METHODS

The three open source projects we looked at are NetBSD, Apache httpd, and Subversion. We chose these projects because they are distributed, they are at least medium-sized in terms of both the code and the development team, and they all produce a product that is widely used, indicating that they have successfully managed to coordinate development.

NetBSD (www.netbsd.org). The goal of the NetBSD project is to produce a free and redistributable UNIX-like operating system. NetBSD is secure and highly portable. NetBSD is a mature project with more than 250 developers in many countries.

Apache httpd (httpd.apache.org). The Apache httpd project builds and maintains the Apache HTTP Server, a cross-platform open source web server. The project is mature, and successful: by most accounts Apache is the most popular server on the Internet.

Subversion (subversion.tigris.org). Subversion is a free open-source version control system, intended as a replacement for the current de facto standard, CVS (Concurrent Versioning System). Subversion is a relatively young project: it has only recently released version 1.0 of the system (in February 2004).

We used three main sources of information to investigate group awareness issues: interviews, email archives, and project artifacts. Our primary source of information was interviews with fourteen developers from the three projects (nine from NetBSD, four from Subversion, and one from Apache httpd). Interviews were conducted by email over a two-month period, and were loosely structured around four topics:

- The project's organization and the developer's role and history with the project: how the project is set up, what areas the developer works on, how long they have been a developer, how they became a developer.
- How work is divided on the project: whether developers are assigned roles, whether people work primarily in one area or several, whether there are official restrictions on work area.
- Group awareness requirements: types of information developers gather about others' activities, questions about others that they must answer in order to coordinate.
- Awareness mechanisms: what information sources exist to provide group awareness information; how developers use those sources to find and maintain awareness knowledge.

Issues raised in these interviews were followed up in an analysis of project communication. We examined several months of the archived developer mailing lists looking at how information about developers and their activities was shown and referred to in the

messages and discussions. We also subscribed to and followed the developer lists for two months, to get a sense of the flow of discussions and the timing and frequency of messages.

Finally, we looked through official project artifacts, including the source code repositories, the bug databases, and the project web sites. Repositories were analysed for basic information about the projects such as their age, size, the number of developers, and the amount of partitioning that occurs (see Table 1).

The basic questions above organize our findings in the next sections. We first look at the question of whether there are real requirements for group awareness on these projects, and then turn to the ways that general and specific awareness are maintained.

4. IS GROUP AWARENESS NEEDED?

The main benefits of group awareness on a distributed software project would be in simplifying communication and improving coordination of activity. The need for awareness therefore depends on the degree to which developers must coordinate. As described above, software systems involve dependencies and linkages that require knowledge of others' activities. These dependencies can cause problems when development teams are distributed [12,13]. Does this imply that open source projects have managed to reduce these dependencies – simply based on the fact that they do manage to produce successful software?

There are two ways that these dependencies can be reduced – by reducing the number of developers, or by strongly partitioning the code. The effects of increasing the number of developers has been studied before: Brooks' Law states that "the complexity and communication costs of a project rise with the square of the number of developers" (quoted in [23]). Raymond [23] suggests that OSS projects avoid this explosion of connections by having only a small set of core developers, with a larger 'halo' of people whose activity is limited. Raymond discusses projects with one to three core developers, where awareness can likely be easily maintained through verbal communication; at the higher end, Mockus and colleagues [18] suggest that a core of ten to fifteen developers is the maximum that can be handled without the need to subdivide into separate subprojects. This number is large enough that the maintenance of awareness would not be simple.

As can be seen from Table 1, the three projects that we looked at fall at the high end of the scale. When the core group is defined as those who contribute 80% of the changes to the source code [18], we find that all the projects are large (although the core for NetBSD could be subdivided into its several major subprojects). Nevertheless, it is not the case that awareness requirements are removed just through project size.

A second way to reduce coordination and awareness requirements is through partitioning, where developers work in tightly constrained areas. Each person is essentially the owner of a particular module or set of files, and does most of the work on that code. Partitioning restricts each person's awareness requirements to their own code and its immediate dependencies.

However, we found that partitioning was not so strongly applied that it could remove the need to stay aware of the group. The most strongly partitioned of the three projects is NetBSD; as seen in Table 1, one developer was responsible for most of each file's changes, and two-thirds of files are strongly partitioned. Some practices on this project reinforce the notion of ownership – several types of NetBSD modules have a "maintainer" listed in the source tree, indicating who is responsible for that module; in addition, the project keeps lists of developers on their web site that associate people with particular areas of the code.

Table 1. Project summary statistics.

	NetBSD	httpd	Subversion
Start of project	1993	1996	2001
Current revision	1.6.2	2.0.48	1.0.1
Project size: number of files	21,455	2,788	1,038
Lines of code (millions)	7.5	1.0	0.5
Number of developers	263	69	31
Active developers[1]	127	21	28
Mean committers per file	2.8	5.2	5.4
Mean change of 1st committer	85%	65%	59%
% strongly partitioned files[2]	67%	30%	31%
# of developers who make 80% of changes to the code	18	13	24

[1] 'Active' means committing in a three-month period
[2] 'Strongly partitioned' means that the first developer makes more than 50% of the changes, and none other makes more than 10%.

Even so, developers stated that in practice, people are allowed to work wherever they see fit – reflected in the global commit access that is shared by all developers. When we asked developers whether they primarily work in a 'home' area in the code, most of them said that they had worked in a variety of areas, and were able to "hop around" following their interests. For example:

> *NetBSD developer N1:*
> I am literally responsible only for the overall welfare of the NetBSD Guide…I am, however, implicitly responsible for a wide variety of other items. … Basically, I like variety :)

One developer suggested that the volunteer nature of the project led to an expertise-based model rather than strict divisions:

> *NetBSD developer N2:*
> Responsibility is a strange concept in a collaborative volunteer project. With most things there are several people who know their stuff, so there's no clear concept of responsibility.
> The exception is of course where someone's name is down against something. For example, I put my name against the <xyz> package as its maintainer, and so I'm responsible for it. When I commit my new port, I will be responsible for that.

On Apache httpd and Subversion, in contrast, official partitioning was almost nonexistent – on these projects, "all committers are responsible for all parts of the code" – and in fact the traditions of both projects argue explicitly against partitioning. Part of the reason is likely that these projects are much smaller than NetBSD, but they have also found that ownership can cause as many coordination problems as it solves:

> *Apache httpd developer A1:*
> For httpd, the paradigm is that all committers are responsible for all parts of the code. This is to lessen the impact of 'owned' modules. … a few modules were 'owned' by a particular individual, [but] when that person left, the module rotted. …those modules are detested by [the] general httpd community because [they were] never cleanly integrated and there was 'ownership' regarding that module that was never clearly relinquished. In general, we really try to avoid 'clear' ownership. It's been bad before when that's happened.

The summary statistics for httpd and Subversion (Table 1) reflect this attitude: the largest fraction that any developer contributes to any single file is about two thirds, and less than a third of the files are strongly associated with a single developer.

The lack of clear partitioning reinforces the findings of Mockus and colleagues [18], who suggest that it is not simply the structure of a project that enables developers to coordinate their actions:

> [Lack of clear ownership] strongly suggests some other mechanism for coordinating contributions. It seems that rather than any single individual writing all the code for a given module, those in the core group have a sufficient level of mutual trust that they contribute code to various modules as needed. [It is] more a matter of recognition of expertise than one of strictly enforced ability to make commits to partitions of the code base. [18, p.325]

This way of organizing projects – through areas of expertise rather than through explicit partitioning – does not remove awareness requirements; it actually increases them. When developers can work anywhere, they need to know who else is active in the area, and who the experts are, so group awareness becomes a critical component in successful coordination.

When we asked developers what kinds of information about others that they tracked, they mentioned two types. First, developers maintain a broad awareness of who are the main people working on their project, and what their areas of expertise are. Second, when a developer wishes to do work in a particular area, they must gain more detailed knowledge about who are the people with experience in that part of the code. We next look in more detail at each of these types of awareness, and at how developers maintain (or find) each type of knowledge.

5. GENERAL AWARENESS: Who's Who?

The developers that we interviewed all maintain an overall, broad awareness of who is who and who does what on their project. We were surprised to learn, however, that the primary mechanisms for maintaining this awareness are the text-based communication tools that are common to every project (mailing lists and text chat). The examples below give some indication of how the developers maintained general awareness:

> *NetBSD developer N3:*
> My tracking of NetBSD things has been almost entirely in the form of watching mailing lists. As for who's who, that rapidly became apparent when I started following the lists and seeing who did what. I can't follow a mailing list for long without picking up some sense of who people are, and in those days, I followed source-changes (which received mail about every commit), which made me aware of people who had low on-list profiles but were busy.
>
> *NetBSD developer N4:*
> [For information that I] "just know," I'd say it comes from reading the mailing lists and noting who often talks/commits code in specific areas. As well on icb getting to know folks and who does what areas.
>
> *NetBSD developer N5:*
> I had been following most of the mailing lists, but if I remember correctly I did not pay a lot of attention to the cvs logs. This certainly made me miss a lot of details, but I was able to keep the overall picture up to date.
>
> *NetBSD developer N2:*
> I watch email lists, but I don't watch cvs commits…I keep an eye on as much as I can out of what interests me. This is wider than my work area, but limited by the amount of time I have available. I read the kernel and userland lists for example, but I had to unsubscribe from the vax list a long time ago simply because I didn't have enough time...
>
> *NetBSD developer N1:*
> Since I tend to hop around I really try to keep a 10,000 ft. view of what is going on across the entire system. I watch the commits (but I do not read all of them) and of course developer discussions. Both of those activities are enough to give me a sense of what is going on.

These comments show three main ways that developers maintain general awareness: reading developer mailing lists, reading real-time chat, and watching commits to the code repository. In the next sections we look these information sources and the ways that developers use them to stay aware.

5.1 General awareness through mailing lists

The developer mailing list is the primary communication channel for an OSS project, and is also the primary mechanism for maintaining awareness. As N3 told us, "to a first approximation, all such knowledge [of the group] came my way on the mailing lists." Although discussion forums have been studied previously (e.g., [28,29]), they have not been considered in situations where people use them to coordinate work and keep track of others.

A mailing list, however, is hardly what one would think of as an effective awareness mechanism for distributed software teams: it is disconnected from the work artifacts (i.e. the code), there is no real requirement that people read or post to it, and it requires considerable time and effort for both readers and authors. In particular, it requires that people explicitly state awareness information. Although this sometimes happens in copresent activity (such as with the 'outlouds' discussed by Heath and colleagues [11]), the threshold of effort that people are willing to overcome for this kind of informing is usually very low.

Despite these apparent drawbacks, the mailing lists clearly are used, and used successfully, to gather and provide a reasonable awareness of who is on the project and what their activities are. When we investigated this further with the developers, we found two contributing factors: that people do take the time to explicitly inform others, and that considerable implicit information can be had just by 'overhearing' conversations on the list.

First, it is clear that people do post messages that are intended to inform. The examples in Figure 1 are typical: they are usually short, to the point, and state what someone has done or is going to do. In most cases, these kinds of messages generate no further discussion on the list.

To:dev@subversion.tigris.org From: <Subversion developer S1> Subject: [Patch] links page - fixed link to cvs2svn page - added trac - added submaster <*no follow-up messages*>
To:dev@subversion.tigris.org From: <Subversion developer S2> Subject: svn.collab.net upgrade complete See subject header. <*no follow-up messages*>

Figure 1. Two examples of informational mailing-list messages

What do people gather, in terms of group awareness, from messages like these? The intent of the author is not to provide group awareness, but to provide information about the activity that has been done or is about to happen. However, readers can also see that the author is active, is working in a certain area of the project, and is (or has been) engaged in a particular task. Even if a reader does not much care that a web page has been updated, it may still be valuable to know that the author is active and is doing work that has something to do with the web site. Here we see that people's expertise starts to show through, based on the activities that are reported on the list.

The second type of message involves discussion rather than dissemination. In these messages (which form the large majority of traffic on the lists), one developer posts a bug report, a design question, a proposal for a new feature, or a critique of current code, and a conversation ensues with other interested parties (see Figure 2). There are always numerous conversations going on at the same time – people do not wait for one topic to close before starting a new one, and since the list is a single public channel, many different discussions are interleaved in one place.

To:dev@subversion.tigris.org
From: <Subversion developer S4>
Subject: API/Interface design questions

Several of the functions which I have been working on to support the property keyword expansion have had altered signatures. Specifically, these two:

svn_subst_keywords_differ()
svn_subst_copy_and_translate()

have had a pool term added. So … I was hoping for some guidance on appropriate names for the new functions (appending "_pool" seems kind of lame). <…>

<14 follow-up messages involving six developers>

Figure 2. A message that opens a discussion

Any particular reader of the list will have different degrees of involvement with the different conversations, and will likely be peripheral to most of them. Being a peripheral participant in a conversation, however, does not mean that the discussions are irrelevant; as with the informational messages above, 'overhearing' these conversations still provides valuable awareness information [19]. Most importantly, peripheral participants find out who is talking about what: who asks a question or proposes an idea, and then, just as importantly, who answers the question or comments on the idea. Thus, they get information about who is working (or interested) in a particular area of the project, and who the 'players' are for that area.

The mechanism of overhearing indicates one way that mailing lists get around the question of effort raised earlier. When messages are discussions rather than announcements, awareness information comes along as an unintended byproduct of an activity that was occurring anyways. The discussion is about the bug, or the design, or the best way to implement a new feature, and people simply participate in the discussion; but as a consequence, group awareness information is implicitly conveyed to others reading the list. This phenomenon is comparable to the kinds of overhearing observed with voice communication, in ship navigation teams [15] and pilot-controller conversations [21].

Overhearing can only work as an effective awareness mechanism if a high proportion of project activity gets discussed on the list. If developers go off by themselves and build new features or make changes without discussing them on the list first, there is little evidence that will tell others what that person is doing. Does this practice – of having a public discussion about work in progress or yet to begin – happen on OSS projects? On the projects we studied, it definitely does; and in fact, there is a strong culture of 'keeping it public,' as the following comments attest:

Apache developer A1:
Almost *all* communication is done via the mailing lists. In the words of one of our developers <A3>, "If it doesn't happen on list, it doesn't happen."

NetBSD developer N6:
If something major's changed, this is usually discussed first on the mailing lists - proposal, flaming, improvement.

NetBSD developer N7:
The various mailing-lists are the most important source of information for me. If something happens it starts (most of the time) there.

NetBSD developer N3:
Of course, there are discussions taking place off-list (even in meatspace, in areas where there are enough developers concentrated to make that possible), but my impression is that the bulk of the discussion about anything even mildly major is carried out on the relevant mailing list(s), where, yes, other interested parties may lurk and pick up information without being very visible.

There are some struggles to maintain this practice (particularly when there are multiple forums for communication), but for the most part developers do follow the general guideline. In fact, in some situations people are reprimanded for not being public enough about their activities:

NetBSD developer N3:
I have seen people get slapped down, either publicly or semi-publicly (such as on developers-only (non-public) lists), for committing changes without suitable discussion first.

The public nature of the mailing lists, and the possibility of overhearing conversations, is reminiscent of a co-present work setting – of several people sitting at different desks in an office and asking questions, stating comments, and making remarks to the entire room (as in securities trading rooms [11]). In fact this idea of putting developers into the same room is a common strategy in commercial software development (e.g., [26]), done so that developers can communicate with each other more quickly and can keep track of each others' activities with less effort.

Previous work has suggested that this awareness happens best when teams are co-present. Teasley and colleagues state that "the gains from being *at hand* drop off significantly when people are first out of sight, and then most severely when they are more than 30 meters apart" [26, p. 345]. The example of the mailing lists suggests that at least some of the qualities of being 'at hand' for other developers can be transferred to distributed and asynchronous collaboration. This is achieved not through advanced technology, but rather by a reliance on (and a commitment to) ordinary verbal communication.

5.2 General awareness through text chat

NetBSD and Subversion (and most other OSS projects) provide a facility for real-time text chat in addition to the developer mailing lists. On Subversion, an IRC (Internet Relay Chat) channel is used (#svn); on NetBSD, the project runs its own private chat server using the ICB system (Internet CB radio). Both IRC and ICB allow any number of people to be present in the session, and all conversation is seen by all participants. Most end-user clients also show a list of the people who are currently logged in to the server.

A number of developers, particularly on NetBSD, stated that text chat was also an important means for maintaining awareness, and NetBSD people follow the chat system fairly closely. The ideas introduced above for mailing lists also apply to the chat system: although there is considerably more off-topic talk than on the mailing list, there is also technical discussion, and so lurkers can pick up awareness information by overhearing conversations.

NetBSD developer N6:
The NetBSD project hosts a chat server where developers hang out, and this is very often used as a medium for short-term discussion between two or few other developers. Questions may be "is anyone working on XXX?" or "anyone

got a clue on YYY?". Answers may either be direct or pointers on where/whom to ask/look.
To toss a few numbers, some smart person estimated the number of _active_ NetBSD members to about 200 (where active == did some CVS commit in the last weeks/months - don't ask me how long). Of these 200, right now 86 are logged into the chat server.

Chat conversations are much more informal than other project communication, and are sometimes unrelated to the project – as N4 says, "a lot of the time it's just stream of consciousness babbling, but often if the person you want is there, [then] technical conversation happens also." The real-time and informal nature of the channel also appears to provide an opportunity for the "unplanned contacts [that] are surprisingly important in keeping projects coordinated" [13]. The comment below shows exactly this kind of informal but useful exchange. Comments such as "did you see what C was up to today?" provide exactly the kind of 'water-cooler awareness' that has been described for co-located projects.

NetBSD developer N2:
In many cases I "just know" [about what people are doing], mostly from mailing lists and icb, but also from general conversations and chat about stuff that happens in the project. "Did you see what C was up to today? He seems to have launched an all-out assault on the ACPI code", or "Wow, another couple of network card drivers from J".

One potential drawback of a chat server for awareness is that it could split the public conversation between two forums. Since text chat is not archived, any developers who are not reading a particular conversation will lose access to that information. If a technical discussion happens on the chat server, that information is 'siphoned off' [29] from the mailing list. (Similar problems can occur with issue and bug tracking systems).

The projects that use chat servers have found a reasonable compromise, but one that does require some effort. Their workaround is to summarize relevant parts of the discussion from ICB or IRC back into the mailing list, as suggested below. This becomes part of the culture of 'making it public,' but is something that has to be remembered and reinforced.

NetBSD developer N5:
The contents there [on ICB] vary from mailing lists topics a lot, sometimes there are short bursts of in-depth technical brain storming sessions, or mutual debugging help by several people looking at the same problem. It is, however, considered bad style to have technical discussions there, since it is no public service and not documented later. So many ideas get born there, but then discussed in public on one of the technical (public) lists.

Subversion developer S3:
…the more experienced developers actively encourage the pertinent parts of IRC conversations to be copied into issue descriptions and/or to the mailing lists. Many times people will start having discussion-style exchanges in (say) the issue tracker, and someone will come by and ask them to do it on the mailing list instead. In general, there is a consciousness that different types of discussions belong in different types of forums -- and that it's okay to point that out, even in the middle of a conversation.

This problem, however, may be one reason why the Apache httpd project does not use a chat system:

Apache httpd developer A1:
…at one point, irc.openprojects.net #apr used to be a hotbed of preliminary discussion for APR-related topics. It's since died off due to time constraints for most of us and the fact that we were leaving out people in these 'real-time' conversations that we shouldn't have been. So, that's pretty much idle now. It was an interesting experiment, but it failed.

5.3 General awareness through commit logs

Commit logs are the records of changes made to the source code. These records are kept in the project's version control system (CVS for NetBSD and httpd, or SVN for Subversion). The record for each change includes information such as the person committing the change, the files affected, the number of changes, and the 'diff' (difference) between the old and new versions. Changes are automatically sent to a mailing list that developers can subscribe to. The commit log is the only awareness source that is based on actual manipulations of the project artifacts.

Many developers subscribe to these lists. They can see what type of changes are being made, and by whom, just by reading the subject lines of the mail messages. Most developers keep an eye on the commits to stay up to date on what is happening on the project, and to watch for changes that may affect their own work or plans. Seeing changes go by, however, also provides group awareness information: both the fact that people are active, and also what modules people are currently working on. As N1 states:

Very often, CVS commit logs (which are mailed to me) can allude to who works in what areas. For instance, if I had a question about plugging a new nic driver into the pci framework, I might ask someone who I have seen commit drivers to it before.

The main drawback that developers mentioned for this method of maintaining awareness is that commits can be numerous (e.g., more than 100 per day has been seen on NetBSD). Therefore, this list can be time-consuming and tedious to read.

6. SPECIFIC AWARENESS: Who to talk to?

The mechanisms described above give people a broad understanding of who people are and where they work. However, more detailed information about people's expertise and activities is often required as well. In particular, when a developer wants to do work, fix a bug, or propose a change in an area new to them, they need to know who the relevant people are with experience on that part of the project. Although this problem involves information seeking rather than ongoing maintenance of knowledge, it is still a group awareness issue.

Two main approaches were reported by the developers that we talked to. First, people use a variety of information sources available on the project to find out who are the experts in an area. Second, people post to the mailing list and just assume that the appropriate people will join the discussion. Below we describe the information sources, and then look more closely at the mailing list and at its built-in mechanism for finding experts.

6.1 Finding the right people

Developers on NetBSD discussed several ways that they looked for information when planning work in a new area. As N1 says, "basically, before I set out I try to identify who else might know a lot about the area I intend to muck around in." The example below is a good indication that NetBSD has several information sources: some that are based on the partitioning and responsibility assignment seen on this project, and some that are based on general awareness and social relationships.

NetBSD developer N4:
Generally for [finding people] it comes down to:

1. Last person to commit the file
2. Looking over the file(s) [in CVS] seeing who's most active
3. Sending mail to the various tech-* mailing lists
4. Asking on icb
5. Getting a hold of some specific folks depending on area of expertise based on either pre-existing knowledge (just "been around a while so I know who does this") or possibly checking www.netbsd.org/People/developers.html and seeing if someone is listed on the area you're looking for. Generally folks answering #3 may also point you at someone specific.

We identified five main information sources from the interviews:

- *'Maintainer' field*. Several files or modules in NetBSD (particularly user programs in the pkgsrc group) have a specific maintainer listed in the source tree. As N6 said, "every pkg in pkgsrc has a maintainer noted in the Makefile, and if you touch something not your own, you ask."
- *Repository logs*. With each commit, the version control system records both the changes made and who made them; these records can be inspected to find out who has been working recently or frequently on a particular file. The value of finding a recent committer is that "the last person to touch something is usually a good starting point, as it's more likely to be fresh in their memory." (N2)
- *Issue and bug trackers*. Most OSS projects use a tracking system for bugs, feature requests, and other issues (e.g., subversion.tigris.org/project_issues.html). Reports (retrieved through a query interface) state whether the issue has been assigned to a developer; some systems also allow queries that show all the issues assigned to a particular person.
- *Asking other developers*. Several people mentioned that they can get information about who is working in an area by asking a more senior developer, or an acquaintance who is closer to that area of work. NetBSD has a 'sponsorship' program for new developers in which an established member of the project assists a new person, answering questions and overseeing commits. These mentors often act as contacts well beyond the official sponsorship period. As N7 said, "for specific problems I ask my sponsors first, because they live in the same timezone and speak the same language."
- *Project documentation*. NetBSD keeps several web pages that indicate responsibility for different areas. These lists are fairly general, and several developers stated that they aren't sure if they are up to date (e.g., "The NetBSD projects server is intended to help tracking who's working on what…but in practice that's not really maintained" – N6).

In contrast to NetBSD, developers on Apache httpd and Subversion were much more likely to simply post to a mailing list rather than investigate other sources (and these projects do not use maintainer lists). As we discuss next, there are characteristics of the list itself that make this approach an effective awareness tool that allows implicit investigation of expertise.

6.2 Asking the list

If the outcome of the information-seeking activities described above is clear, then a private email exchange can often be carried out (at least for small things that do not need to be made public). However, we found that developers check these sources as much to be prepared for a public discussion as they do to find a specific person for private conversation. The other sources therefore provide a set of expectations as to who their audience is, but not a complete list. A1 states:

> …usually, in my head, I have a few people in mind that *should* respond to it when I post a question I don't know the answer to. For example, if I have a question on whether something works on AIX, I'll post to the list expecting <A2> or one of the other IBM guys to respond. However, others are welcome to chime in.

Therefore, people post questions or proposals to the list without knowing exactly who the correct audience should be. However, the structure of the mailing list and the way that it is used on OSS projects combine to form a valuable characteristic in finding this audience. The list provides a robust and simple way for people to find out with whom they should discuss an issue: they just ask their question or start their discussion, and the appropriate people will self-identify themselves by replying. As N2 says:

> if I really don't know who to talk to, can't get in touch with them easily, or can't find a quick resolution, there's always a mailing list to ask in. Kernel stuff in tech-kern, userland stuff in tech-userland, arm stuff in port-arm and so on. *The right person or people will almost always answer such a query, and I will often get input from other people who have experience in that area too*" (italics added).

The list can therefore act as a proxy for every individual subscribed to it. Developers can count on the fact that the message will be read and answered by appropriate people. It is evident from the lists that people write their messages as if they are going to the correct audience, even if the writer does not know exactly who they are. This can be seen even in the language and forms of address that people use: for example, initial posts are written as if the appropriate audience is there and listening – people say "do you think X?" rather than "is anyone out there who thinks X?"

<To:tech-pkg@NetBSD.org>
I would like to add some new IMAKE_* variables to
defs.${OPSYS}.mk and bsd.pkg.mk for automatic manpage
handling in the XFree86 packages from pkgsrc.
Please have a look at this patch:
<code omitted>
What do you think? is it ok to commit?

Figure 3. Initial list message talking to an intended audience.

The idea of list-as-proxy makes the process of finding people extremely robust, and provides an 'awareness default.' That is, at a first approximation, the correct person to ask (no matter what the question) is "the list;" once into the discussion, when the relevant parties have identified themselves, awareness can be refined and made more specific with names of particular people.

The robustness of being able to ask a specific question to an entire group has been seen before in text chat [8] and multiparty voice links [2]. These previous studies have suggested other advantages for this way of communicating; in particular, people are not socially required to respond (as they are with telephone conversations or private email), and so the top experts do not become flooded with questions [17]. This flexibility, however, raises the question of whether the 'right people' really do answer a question, as opposed to the wrong people (those who give incorrect answers), or as opposed to nobody at all? This has been observed as a problem in other settings (e.g., the Zephyr help system [1], where response quality was low at times).

Although there were a few reports of messages going unanswered, developers clearly felt that they could almost always count on a response from appropriate people. Three differences between developer lists and other types of public forums may contribute to

the higher success rate of these projects. First, developers are committed to reading the list – so there is a very good chance that the relevant experts will see at least the initial post of a discussion. Second, even if the senior developers are too busy to answer all questions themselves, they watch who does respond and check the answers, often weighing in themselves if they feel that something has been stated incorrectly or if a question in their area goes unanswered. Third, there is social pressure to be correct: incorrect or poor answers are noticed, and for sake of reputation, developers will usually not answer questions in areas where they have little experience.

7. DOES AWARENESS WORK?

Does the awareness that developers get from these sources and mechanisms actually lead to successful coordination of effort? Although we did not explicitly study coordination metrics, we asked developers whether they could recall situations where work was duplicated or where people worked at cross purposes.

Only four developers could think of such a situation, suggesting that problems from poor awareness are relatively rare. As N1 said, "believe it or not, *not* trampling, causing overlap etc. is much easier than many people believe." One developer (N6) had not seen an instance in his entire time on the project:

> *NetBSD developer N6:*
> In the >5 years I'm a NetBSD developer, I cannot recall a situation where people have invested time twice redundantly. So far, people have always managed to work together on related things, sometimes in tightly coupled teams (the latest virtual memory rototiling, threads implementation and compiler changes/integration come to mind).

Those developers who did recall problems also stated that the effects on the collaborative effort were usually quite small:

> *NetBSD developer N4:*
> I can recall doing work and then seeing someone commit something which essentially duplicated what I was doing. If it's small it doesn't really matter, just move on. For larger items the common method is to branch the tree, do the work and then merge back when done. For cases like that you can set the ground rules for who works on the branch and how work is done so less conflicts happen.

> *Apache http developer A1:*
> I know it's happened; but I can't think of any specific cases. Typically, it isn't that major; or it's because we're pressed for time ([security vulnerabilities] are usually the case here).
> However, we might post two (or more!) different ways of addressing the same problem if we see a tradeoff but don't know which way to go. I know that we've done this when we wanted to do performance analysis/tuning. Should we optimize for speed or space? We'll post both and then a discussion ensues about what we should do.

These experiences suggest that the awareness that developers maintain is for the most part sufficient for their purposes – but for situations were there are problems, there also seems to be resiliency built in to OSS projects. One of the points raised by A1 above may be important here – he mentions being "pressed for time" as a factor in coordination problems. Open-source projects usually have much greater flexibility in schedule than do commercial development projects, and with more open timelines, it is much more likely that coordination errors can be detected and repaired while they are still minor.

As a second look at whether their current awareness mechanisms are sufficient, we asked the developers whether they had looked at using any other tools that provide information about people's activities on a project. The first tool we asked about was 'cvs edit' in the CVS revision system. This command helps people keep track of the specific files others are currently working on; we thought that it might be used to get a finer-grained awareness of activities than what was available through the traditional sources.

However, of the ten developers who used CVS, none used 'cvs edit.' It appears that the verbal communication on lists and chat is sufficient – as suggested by N6:

> reason we don't use 'cvs edit'... dunno, I've never seen anyone use it. I guess just because some feature is there, one doesn't have to use it. As it's said in the CVS manual, "cvs is no substitute for communication", and I guess we implement better ways for 'cvs edit'. :)

We also discussed visualization tools with a few of the developers who were familiar with them. Although we have only a small amount of data from these discussions, it seems likely that some developers will not see a great benefit from these systems. As suggested by A1, one place where these kinds of systems could be valuable is in helping new developers come up to speed.

> *Apache http developer A1:*
> I'm not sure that these historical graphs and data analysis (such as LXR or Bonsai) matter all that much - I already have a mental picture. It's a cool toy to me. Nice, but not helpful. Perhaps it could replace my infrequent use of ViewCVS when I'm trying to remember who did what.
> However, I *do* see potential value for new participants who don't yet have this picture. So, while I might not be interested in it, people who are trying to join might find this extremely useful as they try to build that mental picture.
> I'll also state that the httpd community has a strong aversion to 'social network charts' - people have posted the 'top 50 posters to dev@httpd' on the mailing list and get roundly criticized. [A former chair of httpd said] something along the lines of, "This type of information is disruptive. All of the committers are on equal footing." And, that's pretty much true…

What, then, would developers like to have in this area? Some problems were reported with the current tools, although these were relatively minor: some developers wanted better project lists (of who works in what area) that could be automatically kept up to date, and two people wanted a better way to decide (based on areas of work) which developer should be asked to handle particular bugs as they are reported. The other main awareness system that several developers would like to see improved was the access to email archives: as N4 said,

> …being able to effectively search the 10+ year email archive effectively is the largest missing hole we have. The tools for our search engine today just suck there and often it returns the kitchen sink or nothing for a given query.

Overall, however, there were few major complaints about being able to stay aware, find people when needed, communicate effectively, and coordinate plans and actions.

8. DISCUSSION

Our findings show how one kind of real-world distributed group maintains group awareness. Although this is only a small part of the overall story of how OSS teams overcome the problems of distance, we have been able to expose some of the information sources and mechanisms that allow these projects to stay coordinated (see Table 2). In this section, we consider ways that our results can be used in the broader CSCW context: we look at whether the specific awareness mechanisms seen in the study

could be used in other distributed work settings, whether there are underlying principles that can be applied more widely, and whether new tools could assist awareness on open-source projects.

Table 2. Summary of capabilities and requirements for the awareness mechanisms seen on the three projects.

Dynamic information sources
Developer mailing lists. Overhearing is a primary mechanism; wide readership allows authors to reach 'the right people.' Requires additional communication effort, a strong culture of making things public, and a critical mass of readers.
Text chat. Provides for ad-hoc communication and overhearing of informal and work-related discussions. Risk of removing communication from mailing lists; however, summaries can be posted back to the list.
Commits. Indicate people's activity levels and area of work. Can be time-consuming and tedious to read.
On-demand awareness queries
'Maintainer' field. Explicit indication of who to talk to about changes. Effort is required to keep the information up to date; the project may not agree with code ownership.
Code repository. Allows inspection of activity based on changes to project artifacts. Text-based displays means that some information such as frequency of activity is difficult to see.
Issue and bug trackers. Provide information about assignments, and show focused communication about each issue. Require explicit effort, and may remove communication from other lists.
Asking senior developers. Allows use of social networks to find other people. Requires explicit communication and an organizational culture that allows and promotes contact.
Project documentation. Provides direct information about activities and areas of work. Must be kept up to date.

Are the specific awareness mechanisms transportable?

Although simple text communication works well in these projects, and although text tools like MUDs have been successful in other work environments [4,8], it is not clear that email lists and chat systems are the answer for other distributed teams. Developers on open source projects are often 'the best of the best' in terms of technical skill and ability to get things done, and so it is possible that text-based awareness is feasible for them simply because they are very capable individuals. Also, open-source developers to some degree self-select for success in this environment: if a developer is not able to maintain adequate awareness and is not able to coordinate activity successfully, then because participation is voluntary, there is a good chance that they will not stay with the project. Finally, it appears that one of the primary motivations in open source communities is reputation among one's peers (rather than things like money or altruism) [18]. Although this is unlikely to be an explicit rating or ranking (see A1's comment above about social network charts), reputation is undoubtedly one reason why some of the more effortful parts of maintaining group awareness – reading the lists, writing good-quality responses, helping others – continue to be done by the majority of the community, and one reason why the communication forums sustain critical mass [29].

There are, of course, many people outside of open source who are technically proficient, capable, and highly motivated; it will be interesting to see whether text-based awareness can work in other distributed groups. This is an area that we plan to pursue in future.

Are there underlying principles that can be used more widely?

Even if the specifics of these projects cannot be used widely, there are a few general principles that can have broader applicability in supporting distributed awareness.

First is the importance of verbal communication, and the value of different forms and venues for discussion. For the most part, our findings reinforce previous results; however, it is worth noting the value of providing support for both 'formal' discussions (on the mailing list) and informal, ad-hoc talk on the chat system. It is also useful to know that written conversation can in some settings take the place of audio communication (a result that differs from other conclusions, e.g., [13]).

Second is the significance of overhearing as an awareness tool. Although the usefulness of this behaviour has been recognized in studies of audio channels, studies of textual communication have sometimes characterized these 'lurkers' negatively, as free riders. In many circumstances, however, they may be simply acting as peripheral participants, gathering general awareness that helps them to keep in touch with the community and the project.

Third is the value of broadcast communication. As seen in co-located situations, the ability to speak to an expected audience rather than to a specific one had several advantages in finding the right people and allowing people to decide for themselves whether to respond. This principle and the one above suggest that designers should consider whether communication facilities should be public (like a chat server) rather than point-to-point (like instant messaging or private email).

Can the open-source setting be better supported?

There were some indications of difficulties that were discussed in the interviews, even if these did not prevent people from maintaining awareness. For example, comments above mention the effort involved in reading the lists (particularly commit logs), the difficulty of managing one conversation in two communication channels (mailing list and chat), and the problems of looking for information in the mail archives. We are interested in whether developers' existing awareness support could be augmented without fundamentally changing it. We have several possibilities that we are currently discussing with developers:

- Mailing lists are time-consuming; we are looking at whether new representations of messages and threads can help to support group awareness with less effort.
- CVS commits are sometimes ignored due to time constraints; it may be possible to show dynamic awareness information from the CVS repository in a form that allows for easier browsing, filtering, and inspection.
- The splitting of communication between mail, chat, and issue tracker suggests potential for tools that link related conversational streams. This could allow conversations to be seen in the context of work artifacts (as is done in the issue tracker), without losing the public nature of the discussion.
- The idea of making things public could be extended to other types of interactions. Although this is already the basis for 'edit wear and read wear' approaches such as Augur [9], the idea could be extended to interactions with awareness information sources. For example, it could be valuable to

visualize the frequency with which people look at different information in the CVS repository.

- Search tools could be designed with awareness queries in mind. Archives are valuable resources for group awareness, particularly for new developers, but it is not known how people look for awareness information in these kinds of databases. Mining the archives should be done with caution, however, given the likely reluctance to have certain types of social relationships made explicit.

9. CONCLUSION

Open-source software development projects are examples of collaborative, distributed work where people are able to maintain awareness of each other and of others' activities. In this study we looked at requirements and mechanisms for group awareness on three open source projects. We found that distributed developers maintain both a general awareness of the entire team and more detailed knowledge of people that they plan to work with. The primary means for maintaining awareness were mailing lists and chat tools; we were struck by the capabilities of text-based communication for supporting awareness, and by the importance of the organizational culture in promoting the kinds of behaviour that make good group awareness possible through these tools.

This study is one of the first to consider how awareness works in the real world. One thing that is clear from the study, in addition what we discuss above, is that awareness is both complex and subtle. There are many leads in our data that we were unable to address here. These issues – such as the ways that non-English-speaking developers use the lists, how occasional face-to-face gatherings assist group awareness, how reputation really affects mailing-list practices, what kinds of miscommunications arise in list-based discussion, or the ways that project size affect awareness mechanisms – will be investigated as we look more closely at different parts of the data.

10. ACKNOWLEDGMENTS

Many thanks to all the developers on NetBSD, Apache, and Subversion who talked about their experiences with us, particularly Greg Oster who got the ball rolling. Thanks also to Barry Brown for valuable comments on an earlier draft.

11. REFERENCES

[1] Ackerman, M., and Palen, L., The Zephyr Help Instance: Promoting Ongoing Activity in a CSCW System, *Proc. ACM CHI 1996*, 268-275.

[2] Ackerman, M., Hindus, D., Mainwaring, S., and Starr, B., Hanging on the 'Wire: A Field Study of an Audio-Only Media Space, *ACM ToCHI*, 1997, 4,1, 39-66.

[3] Benford, S., Bowers, J., Fahlen, L., Greenhalgh, C., and Snowdon, D., User Embodiment in Collaborative Virtual Environments, *Proc. ACM CHI'95*, 242-249.

[4] Churchill, E., and Bly, S, It's all in the words: Supporting Work Activities with Lightweight Tools. *Proc. ACM GROUP 1999*, xx-yy.

[5] Dix, A., Finlay, J., Abowd, G., and Beale, R., Human-Computer Interaction, Prentice Hall, 1993.

[6] Dourish, P., and Bellotti, V., Awareness and Coordination in Shared Workspaces, *Proc. ACM CSCW 1992*, 107-114.

[7] Elliott, M., and Scacchi, W., Free software developers as an occupational community: resolving conflicts and fostering collaboration, *Proc. ACM GROUP 2003*, 21-30.

[8] Fitzpatrick, G., Kaplan, S. Mansfield, T., Arnold D., and Segall, B., Supporting Public Availability and Accessibility with Elvin, *JCSCW*, 11(3), 447-474.

[9] Froehlich, J. and Dourish, P., Unifying Artifacts and Activities in a Visual Tool for Distributed Software Development Teams. *Proc. ICSE 2004*, 2004, 387-396.

[10] Gutwin, C. and Greenberg, S. A Descriptive Framework of Workspace Awareness for Real-Time Groupware. *JCSCW*, Issue 3-4, 2002, 411-446.

[11] Heath, C., Jirotka, M., Luff, P, and Hindmarsh, J, Unpacking Collaboration: the Interactional Organisation of Trading in a City Dealing Room, *JCSCW*, 1995, 3(2), 147-165.

[12] Herbsleb, J., Mockus, A., Finholt, T., and Grinter, R., Distance, Dependencies, and Delay in a Global Collaboration, Proc. ACM CSCW 2000, 319-328.

[13] Herbsleb, J. and Grinter, R. Architectures, Coordination, and Distance: Conway's Law and Beyond. *IEEE Software*, Sept/Oct 1999, 63-70.

[14] Hill, W.C., Hollan, J.D., McCandless, J., and Wroblewski, D. Edit wear and read wear. *Proc. ACM CHI 1992*, 3-9.

[15] Hutchins, E., The Technology of Team Navigation, in Intellectual Teamwork, J. Galegher, R. Kraut, and C. Egido (eds.), Erlbaum, 1990, 191-220.

[16] Kraut, R., and Streeter, L., Coordination in software development. *CACM,* 1995, 69-81.

[17] McDonald, D., and Ackerman, M., Just Talk to Me: A Field Study of Expertise Location Finding and Sustaining Relationships, *Proc. ACM CSCW 1998*, 315-324.

[18] Mockus, A., Fielding, R., and Herbsleb, J. Two Case Studies of Open Source Software Development: Apache and Mozilla, *ACM ToSEM*, 11, 3, 2002, 309-346.

[19] Monk, A., and Watts, L., Peripheral Participants in Mediated Communication, *Proc. ACM CHI 1998*, v.2, 285-286.

[20] Nonnecke, B., and Preece, J., Lurker Demographics: Counting the Silent, *Proc. ACM CHI 2000*, 73-80.

[21] Norman, D., *Things That Make Us Smart*, Addison, 1993.

[22] Olson, J., and Teasley, S., Groupware in the Wild: Lessons Learned from a Year of Virtual Collocation, *Proc. ACM CSCW 1996*, 419-427.

[23] Raymond, E., The Cathedral and the Bazaar, O'Reilly, 2001.

[24] Schummer, T., Lost and found in software space. *Proc 34th HICSS,* 2001.

[25] Segal, L., Designing Team Workstations: The Choreography of Teamwork, in *Local Applications of the Ecological Approach to Human-Machine Systems*, P. Hancock, J. Flach, J. Caird and K. Vicente ed., Erlbaum, 1995, 392-415.

[26] Teasley, S., Covi, L., Krishnan, M., and Olson, J., How does Radical Collocation Help a Team Succeed?, *Proc. ACM CSCW 2000*, 339-346.

[27] Whittaker, S., Frohlich, D., and Daly-Jones, O., Informal Workplace Communication: What is It Like and How Might We Support It?, *Proc. ACM CHI 1994*, 131-137.

[28] Whittaker, S., Talking to Strangers: An Evaluation of the Factors Affecting Electronic Collaboration Groupware Usage *Proc. ACM CSCW 1996*, 409-418.

[29] Whittaker, S., Terveen, L., Hill, W., and Cherny, L., The Dynamics of Mass Interaction, *Proc. CSCW 1998*, 257-264.