Understanding How and Why Open Source Contributors Use Diagrams in the Development of Ubuntu

Koji Yatani[1], Eunyoung Chung[2], Carlos Jensen[2], and Khai N. Truong[1]

[1]Department of Computer Science
University of Toronto
Toronto, ON M5S 3G4, Canada
{koji, khai}@dgp.toronto.edu

[2]School of Electrical Engineering & Computer Science
Oregon State University
Corvalis, OR 97331-5501, USA
{chung, cjensen}@eecs.oregonstate.edu

ABSTRACT
Some of the most interesting differences between Open Source Software (OSS) development and commercial co-located software development lie in the communication and collaboration practices of these two groups of developers. One interesting practice is that of diagramming. Though well studied and important in many aspects of co-located software development (including communication and collaboration among developers), its role in OSS development has not been thoroughly studied. In this paper, we report our investigation on how and why Ubuntu contributors use diagrams in their work. Our study shows that diagrams are not actively used in many scenarios where they commonly would in co-located software development efforts. We describe differences in the use and practices of diagramming, their possible reasons, and present design considerations for potential systems aimed at better supporting diagram use in OSS development.

Author Keywords
Diagramming, visual representation, software development, open source software (OSS).

ACM Classification Keywords
H.5.3 [Group and Organization Interfaces] Computer-supported cooperative work; D.2.10 [Design] Methodologies, Representation

INTRODUCTION

Open source software (OSS) efforts use an open software development model where the source code is made freely available to everyone. OSS itself can often be redistributed and used anywhere, while commercial software often has restrictions on its use, modification and/or distribution, and development occurs within a closed community or organization. As one of our participants succinctly states below, open source is a license as well as a philosophy, something very much on the minds of both developers and users. This impacts the practices surrounding both the use of Open Source as well as its development:

> *Open source is a license. It's a legal issue if you look at it that way. It's a license to apply to the work that is freely available. If you look at it from a development process, it's a philosophy. It's a different development method. And the way of communication is something different. [P2]*

Another key feature of many OSS efforts is that they are often based on volunteerism, and rarely involve any co-located developers or teams, whereas most traditional commercial software development efforts involve paid co-located teams. This difference has forced changes to many aspects of the development process, most visible in the ways developers communicate and collaborate. For instance, OSS developers depend almost exclusively on Internet-based communication to maintain an awareness of each other [7]. Though co-located teams use the Internet as well, it is not their only means of communication and co-ordination. When moving to a purely or primarily online organizational structure, many practices need to change.

Cherubini *et al.* discussed the role of diagrams[1] (including drawings and figures) in the software development practices of co-located teams [2]. They found that the use of diagrams played an important role in their participants' practice; for example, they were used to understand the code, foster discussions, design, explain aspects of the software to others, and support documentation or presentations. Given the importance and prevalence of diagrams in co-located software development, it is important to understand how these practices have been adopted and migrated in OSS efforts, specifically with respect to:

CHI 2009, April 4–9, 2009, Boston, Massachusetts, USA.

[1] We refer to any kind of visualization containing information as a diagram. Thus, a diagram may describe code structure, interactions among modules, the organization of a team, or a development schedule. A diagram can be drawn using an analog medium (paper or whiteboard), software (*e.g.*, Visio, Dia, or PowerPoint), through Unified Markup Language (UML), or even using ASCII characters.

RQ1. How do OSS developers use diagrams?

RQ2. Why do OSS developers use diagrams?

RQ3. If they do not use diagrams, why?

RQ4. If they do not use diagrams, how do they cope?

To answer these questions, we conducted semi-structured interviews of active Ubuntu contributors from a variety of locations, backgrounds, and roles within the project. Our study reveals conflicting attitudes towards the use of diagrams. We found that diagrams served a variety of purposes, as they do in co-located software development, but that some differed. We also learned that some participants try to avoid using diagrams and instead develop practices allowing them to complete their tasks and communicate with others without the use of diagrams.

The paper is organized as follows. We first review relevant work, focusing particularly on previous studies on the software development practices of OSS projects and distributed collaboration. Next, we describe our methodological approach. We then discuss the findings from this study, describing the practices surrounding the use of diagrams in Ubuntu. We present design considerations for future systems aiming to better support diagramming in OSS development.

RELATED WORK

Co-located Collaboration and Software Development

In a study of how Integrated Product Teams physically co-located work together and their requirements for working across distances, Poltrock and Engelbeck learned that scheduled meetings and opportunistic interactions amongst developers were valuable in facilitating communication, collaboration, and coordination in team work [11]. Similar findings were presented by Sawyer *et al*. in their study of the effects of a team room (a room to support intense team work with electrical meeting facility) when they interviewed and surveyed developers in one software development laboratory [14]. In addition, they found that a team room provided a closed environment for the project, which allowed the developers to concentrate on their work intensively. Ko *et al*. explored how and what work-related information co-located developers acquire [8]. Their observations of 15 developers at work revealed that awareness of co-workers was the most frequently sought information. However, when surveyed, developers reported that awareness of co-workers was not very important.

Diagrams are important artifacts that have been shown to facilitate co-located software development. For example, Dekel studied two co-located software development scenarios in which participants were given the task of producing an object-oriented framework for developing case management applications [4]. They found three major factors to the success of these co-located meetings: flexibility of drawing space, team structure, and awareness of focus. The importance of a flexible drawing space shows that visual communication plays a critical role during discussion sessions. Additionally, Cherubini *et al*. interviewed nine software developers about how and why they use drawings [2]. They documented nine distinct uses for diagrams in software development teams; additionally, they found that developers themselves would in most cases quickly generate these diagrams (on paper or whiteboard).

Distributed Collaboration and Software Development

Some important contextual information (*e.g.*, facial expressions or body language) available in co-located collaborations is often lost in distributed collaborations. This loss of context results in different types of issues. Bellotti and Bly identified four problems in their study of collaboration practices between geographically distributed team members of a design consulting firm: Team members had difficulty locating people, maintaining awareness of the location and status of other team members, keeping synchronicity in communication, and coordinating collaborative activities [1]. These problems led these designers to prefer co-located collaboration even if it required some physical effort, such as travel to another site to meet face-to-face.

Olson and Teasley's case study of how the use of groupware in distributed collaboration affected the behaviors of members of a design team showed that social responsibility and commitment diminished or disappeared when the team members did not meet face-to-face [10]. To address coordination problems in distributed collaboration, Redmiles *et al*. advocated a new paradigm for distributed software development, called Continuous Coordination, where both formal approaches (*e.g.*, workflow management systems) and informal approaches (*e.g.*, emails and instant messaging) are used to provide scalability and flexibility in coordinating within a project [12].

OSS developers are often geographically distributed. Because they face the challenges described above, many have developed their own practices to cope with these problems. Gutwin *et al*.'s examination of group awareness within OSS projects revealed that text-based communication, such as mailing lists and chat systems, contains valuable information for maintaining group awareness [7]. Robertsa *et al*.'s examination of email communication in the Apache project found that core developers tend to form sub-groups and communicate intensively within these rather with the whole [13]. Elliot *et al.* discovered that summaries of mailing lists and Internet Relay Chat (IRC) logs supports lightweight coordination and awareness within distributed projects [5].

Although diagrams play an important role in software development, including facilitating communication and collaboration among co-located developers, their significance and use in OSS projects have not been well explored. Twidale and Nichols reported use of screenshots and ASCII art in the bug reporting process [15].

Participant	Country	Roles
P1	USA	Translation, community building, patches
P2	Netherland	Marketing
P3	France	Code development, community building
P4	USA	Team planning, testing, community building
P5	Germany	Package maintenance
P6	Hungary	Translations, backporting, bug reporting
P7	USA	Release manager, bug reporting
P8	Canada	Package maintenance
P9	Canada	Project leader, Package maintenance

Table 1. Interview participants.

METHOD

To examine how and why diagrams are used in any and all aspects of the software development process of an OSS project, we performed a series of semi-structured interviews with contributors to one particular effort—Ubuntu. In this section, we first discuss why we chose Ubuntu. Next, we discuss our participant recruitment method and describe those who took part in our study. Finally, we describe how the semi-structure interviews were conducted.

The Ubuntu Project

Because of the potential cultural differences across OSS efforts, we opted to focus on one large and diverse project. Ubuntu is a Linux distribution with a regular release schedule and active developer base. The project has two types of core contributors; members and developers. Members are those who have made any type of significant contribution to Ubuntu (including non-programming chores). Developers are members who have successfully contributed code to Ubuntu. There were about 430 active members and 110 developers as of September 2008 [20].

Ubuntu has a collection of teams that either develop software or organize efforts. As of September 2008, there were over 30 project-based development teams that focused on specific functions or applications in Ubuntu and about 180 local communities that supported localization. Additionally, there were two groups overseeing the overall project: The Technical Board was responsible for all technical decisions, such as the package policy, release feature goals, and package selection for new releases. The Community Council managed the social aspects of the project, including: Code of Conduct; team creation and appointment of team leaders; and the creation of new organizational structures and processes.

Participants

We recruited participants through four Ubuntu mailing lists. Participation was open to anyone currently working on Ubuntu or related projects in regular communication with other contributors. We used these criteria to obtain a sample of individuals who filled various roles in the community as well as people from as many parts of the world as possible. In total, we recruited nine participants from a diverse set of roles (see Table 1). All participants were male, ranging in age from late-teens through late-50s, representing Europe and North America. They had on average of two and half years of experience working with Ubuntu. We compensated participants with $30 USD (or 20 Euros).

Theme	Observed Agreement	Cohen's Kappa
Communication conventions	0.97	0.88
Reasons/Purposes for offline meetings	0.99	0.91
Sharing offline meetings	0.99	0.85
Reasons/Purposes for diagrams	0.96	0.88
Creating Diagrams	0.94	0.71
Updating Diagrams	0.98	0.81
Comparing Diagrams	0.99	0.85
Sharing Diagrams	0.97	0.68
Lack of Diagrams	0.96	0.88

Table 2. Themes identified in the study.

Procedure

Our study was divided into two phases. First, we asked participants to complete a questionnaire and provide us with information and materials for discussion. The questionnaire featured questions about participants' OSS experience, project participation, their roles in each project, and basic demographics. We also asked participants to share diagrams they had created, modified or used as part of their work on Ubuntu. In the second phase of the study, we conducted semi-structured interviews with participants. We used the materials provided by the participants to ground the discussion about their communication with others in the Ubuntu project and their diagram use in the project. Participants were asked to refer back to the diagrams they had provided, as well as any others they might have used or seen in the past. Though some participants were not native English speakers, all interviews were conducted in English. The interviews took between 40 and 60 minutes. All interviews were recorded and transcribed with consent.

To gather a breadth of perspectives, we continued to interview participants until the interview data converged. Although the small sample size used in our study is a potential concern, our data saturation rate is in line with what has been shown experimentally to be achievable for broader research [6]. Because the scope of our study is narrow, we achieved data saturation with few participants.

From the interview transcripts, we extracted approximately 200 excerpts for in-depth analysis. Two researchers conducted open-ended inductive coding on the quotes to identify nine themes pertaining to how OSS contributors communicate and use diagrams (see Table 2). The coding scheme was discussed amongst the research team and refined. A third researcher then performed the coding again for inter-rater reliability [3, 9]. Throughout this paper, we preserve and present subjects own (sometimes original) use

of language and grammar as faithfully as possible, omitting the traditional [*sic*].

Given the qualitative nature of the interviews and the modest size of the participant pool, when numbers are presented, it is to give the reader an idea of the prevalence of certain behaviors across our sample. These numbers are not intended as statistical evidence of frequency and their described practices cannot be assumed to generalize across OSS efforts. However, the actions of our participants illustrate interesting behaviors which merit further research.

RESULTS

Project Communication and Coordination

Because Ubuntu contributors regularly communicate online they have developed conventions to make communication more effective. All participants agreed that their main communication channels are email and IRC, and that communication is predominantly text-based, in line with the findings of Gutwin *et al*. [7].

> *Main source of communication is probably email with mailing lists. Other big source of communication is IRC… Especially, in Ubuntu, we have a very large, very comprehensive wiki, wiki.ubuntu.com. [P5]*

As seen in the comment of P5, Ubuntu contributors communicate over the website (*e.g.*, a wiki or a blog) as well. Launchpad [19] is an important web-based medium used by this project to share ideas about the project, bug-reports and information about the projects' members. It is also used for creating diagrams (discussed later).

P2 explained the different purposes to which IRC channels and mailing lists are put, also in line with [7].

> *Most of the time, online IRC conversation is an unofficial platform for communication because it's fairly hard to share the situation and conversation over IRC… Mailing lists, the platform to discuss and announce real problems and issues and solutions. In some problems, the IRC channel is not used because it takes too much time to read and respond to questions. We are just using mailing lists because it's far easier to keep track of the messages and we can filter out all other stuff. [P2]*

In addition to regular online communication, the project has regular bi-annual meetings, called Ubuntu Developer Summits, attended by many. There discussions center on process and project management, the next version of Ubuntu, and brainstorming new features.

> *Mainly technical issues like what new software we want to enable, how to improve process, how we develop, also how we communicate, how we handle bugs, mainly what features we want to implement in the next or subsequent release. [P5]*

> *The most important discussion that sometimes happens there is to brainstorm. That's always something good. Most of the time, you see, if it is a new concept or a new idea, it can be sent to that kind of meetings. And then we "shoot out". Everyone asks questions about these things. And that's how ideas are evolved. [P2]*

P2 also mentioned the importance of its social aspects of offline meetings.

> *Most of the time, real-life meetings are real social events and make it easier to collaborate with other developers. Because you know them in person, you can know what they look like, or that kind of stuff. [P2]*

Although most of the participants agreed that an offline meeting facilitates informal discussions and strengthens social ties, they recognized the inherent logistic difficulties.

> *This year it was in Venezuela, but I didn't attend there because it was too far and too expensive for me to go there. [P5]*

> *But since we are located all over the world even for the French community, it's quite hard. We tried to have… Let's say, we can meet most of them, but for instance, one of us is living in New York in the French community. One of my friends is living in New York. And he is not with us. So, it's not easy. [P3]*

Because only a limited number of contributors can attend an offline meeting, putting materials created or used at these events online is important. Participants take notes with a collaborative text editor (Gobby [18]), and use wiki markups to reduce publication effort.

> *We usually have, in each group of the sessions, someone who writes specifications, who writes down the results… And document is in the Ubuntu wiki and links from Launchpad or some other web servers, where can track the dependencies or the needs or that kind of stuff… We usually use the wiki markup language in the gobby document, so we can just copy and paste it to the wiki. [P5]*

Videos are also used to archive and share the meetings with their worldwide developer base.

> *So, all the sessions, I think, nearly all of the sessions, are recorded, and broadcasted by the Internet. So, there are public archives, so you can see them later on. [P5]*

The Role of Diagrams

Cherubini *et al*. identified nine uses of diagrams in co-located software development [2]: *understanding existing code*, *ad-hoc meeting*, *designing/refactoring*, *design review*, *onboarding*, *explaining to secondary stakeholders*, *explaining to customers*, *hallway art*, and *documentation*. Interestingly, only a subset of these purposes were observed in our study. In this section, we discuss how the reported uses of diagrams in co-located efforts manifest themselves in Ubuntu as described by participants in interviews.

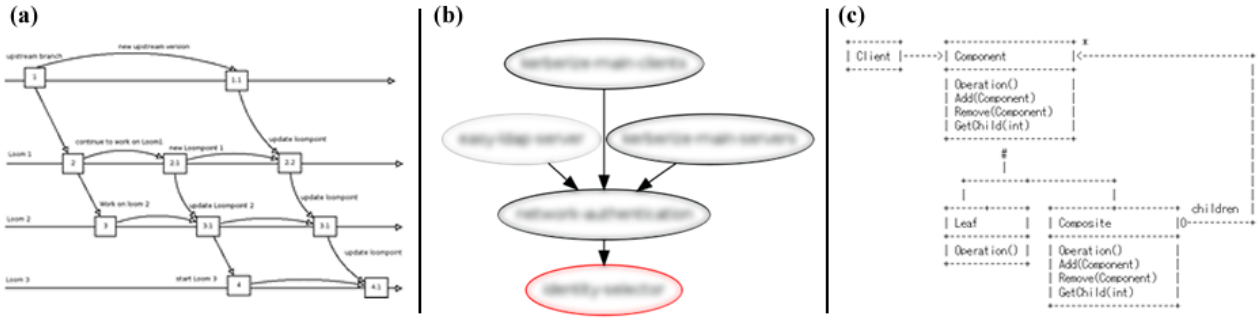

Figure 1. Diagrams created by the participants (some masked for anonymity): (a) flow chart created by P5 with Dia; (b) dependency tree automatically created by launchpad based on the declaration made by P4; and (c) ACSII art created by P2.

Understanding Structures

In contrast to the observations of Cherubini *et al.,* our participants infrequently used diagrams for understanding existing code (we will discuss how they cope without diagrams later). We did, however, find evidence of diagram use for visualization of system structures in a similar way to what Cherubini described as *understanding existing code*.

> *We use diagrams to communicate the infrastructure that we have for the server… Because we have a lot of servers like system servers, we needed to work to have one clear view of them. They have various names… To have a better view of the scheme. Because it was [explained by] some quite long paragraphs and I really think that diagrams could be much easier to understand. And I would say, it's a real image of my proposal. That's why I did some drawings for that. [P3]*

P8 commented that he used handwriting sketches when he needs to understand very difficult problems or algorithms. These sketches however tend not to be shared or maintained.

> *Generally I do not draw anything. Only time I do that is for algorithm or like something very difficult and complex to draw picture of mine. [P8]*

Ad-hoc meeting

Contributors have frequent informal discussions over email or IRC, in which diagrams can help them explain their ideas or opinions to others. P5 created two different diagrams (one of them is Figure 1 (a)) to convey his idea of package maintenance to other developers.

> *At the time when I created this diagram, I noticed a lot of discussions in various mailing lists. They argued about how was the better approach, what was the best tool, what was the best workflow. … I created the wiki page to explain my view on that matter so that in the discussion, I can point people to… "Well, I think this is the best one. This is the best approach. Look at this page." [P5]*

Designing/Refactoring

P7 shared his experience with freehand sketching to design a user interface for his project. Although he found freehand sketching easy, he found making modifications hard, and did not reuse the sketches.

> *I made hand sketch on a piece of a paper with pencils. It was interesting. It required me to do a lot of erasing, which is not really fun. I did not get whole codes when I was doing that. It probably was not the best work… I mean, doing a piece of paper, while it could work, I think, I think it's just so counterproductive just because if you have to make one change, you basically change, could change the whole thing I don't even know. You start to put it together, and oh, it's not wide enough. That's screwed up everything. [P7]*

Design review

Design reviews happen through feedback from other community members. However, reviews are primarily done via text rather than using diagrams. P7 expressed concerns with using diagrams in design reviews.

> *I got a lot of feedbacks from the community once I actually release it on subversion. If testers give feedback, I find [it's] something that I have not thought of. If we use more diagrams, then testers want to give more feedbacks on them. Then, the problem on the development cycle would be caused. If there is feedback too late, it would make repositories delayed. [P7]*

As Cherubini *et al.* point out; diagrams are sometimes used to reverse-engineer ideas. P8 had an experience where his project needed to reverse-engineer a USB module.

> *We had reverse-engineering. I did not know that how reverse engineering worked but they did. [The] USB diagram was very complex. So, we had huge information in written form. We absolutely had diagrams that the USB interface was not drawn manually but they used tools to generate that from data set somehow. [P8]*

Onboarding

OSS projects are largely volunteer-based, and must often deal with high turnover among contributors. Making information necessary for joining available to potential contributors is essential to ensuring the long-term survival

of projects. Two participants gave us examples where diagrams were created primarily to help new developers join their teams.

> *But it's fairly important to create diagrams in a project so that new incoming developers have a lower-level barrier to contribute to the project because it's easy to understand how the code works [with diagrams]. So, I like high-level documentation and high-level diagramming. [P2]*

> *Now in the architecture, we have some figures and something like that. And it was quite needed because otherwise when new comers came to this list, they were completely lost. [P3]*

Explaining to users and secondary stakeholders

Our participants did not mention the use of diagrams to explain software to users and secondary stakeholders. However, the Ubuntu project uses a project management service (Launchpad) with a feature called Blueprints. This tool allows Ubuntu contributors to post feature requests, have discussions, and post specification changes. Launchpad also automatically creates diagrams composed of different colored circles and lines, such as dependency trees. Thus, these diagrams can and likely are used to communicate with secondary stakeholders in this community.

Hallway Art

As Cherubini *et al*. point out; some diagrams are used for sharing overall system or project status awareness. We found one instance of diagram use for this purpose.

> *We have some, I would say, some dedicated diagrams for roadmap for various things we have… In the French community, we have a website, and we have a forum, and we have a wiki and something like that. So, in each, we talk and we organize some of them and we have a roadmap for each of that. [P4]*

Documentation

Because code can be shared and modified by any developer, documentation is often used to assist developers in understanding code structure quickly and accurately. Figure 1 (c) is an example diagram created by P2 for the documentation of his project.

> *I found a document about some changes on package managers for debian. And I think it was a few months ago. And I created a very basic ASCII art, maybe a few weeks ago, to define some projects for documentation of the architecture in the OpenARM project. [P2]*

P2 expressed an interesting perspective on documentation. He felt that a diagram created in the design stage should remain as a part of the project documentation. This means that diagrams may serve different roles over their lifetime.

> *Creating the design of something is also documentation. Before you start developing, you create the design, and the documentation of your final project. So that's also where you create diagrams. So, before I start typing codes, I also write diagrams. [P2]*

However, P2 also recognized that reuse of documents across purposes and audiences is difficult. He described a case where he needed to create a more sophisticated version of a low-fidelity diagram used within the team.

> *ASCII arts are often used for internal documentation in the project. It actually works. But website, which is used for marketing, for attracting new people and all kinds of stuff. …, you have to talk to the audience. [P2]*

Practices and Challenges around Diagram Use

Creating Diagrams

As seen in Figure 1, participants used different tools and techniques to create diagrams. Only two participants (P7 and P8) told us they used freehand sketches, though infrequently, a departure from the findings of [2]. Another two of our participants (P3 and P5) used Dia [16], an open-source drawing tool for software development. Figure 1 (a) is a diagram created by P5 with Dia. P3 explained that he recommended Dia to other developers in his project.

> *Honestly, two of the guys do diagrams and they also use the same tool I did. Because they asked me how I made them (diagrams created by P3), I told them, and they said, "that's great for me." So they are using it. [P3]*

Launchpad automatically creates diagrams showing dependencies on other projects or components based on declared specifications. Figure 1 (b) shows the dependency tree created by Launchpad based on declarations by P4.

> *That was generated automatically. I just told the system which project is dependent on [my system] and what is dependent on [my system]… Based on the dependency that I declared. [P4]*

Our interviews discovered that ASCII art was frequently used for simple diagrams or for internal use. Figure 1 (c) is an ASCII diagram created by P2.

> *I do not use a tool I think is overkilled. Everybody can change ASCII arts. Everybody can delete ASCII arts. Sometimes, it's not official or appealing… Most of the time in the development, especially source code development and documentation and how the internal structure code works, you use ASCII arts. [P2]*

Screenshots were also mentioned as a way to diagram. Although none of the participants had taken screenshots of their projects, P7 told us that contributors working on UI themes shared screenshots within his development project.

They come up with design themes. They have done more the main screen shots and ideas and present them in meetings. [P7]

There are different OSS tools to create diagrams. However, this flexibility can also pose a problem, as P4 explained.

You know, I mean, it's very easy to waste time for making it look pretty that could've gone into just talking about what we want to do or waste time for arguing what format to use or what tool to use. [P4]

P5, who often creates diagrams with Dia, told us that inter-operability of an image format constrains him to Dia.

What has been most annoying is the inter-operability between different programs. I mainly choose to use Dia because I can export it to SVG very easily and import it to Inkscape. With Inkscape, I can do modifications I cannot do with Dia… I have to choose Dia not because it is the best tool but because what I can get out of it, export functions… in that way, I can import it into other programs. [P5]

Sharing Diagrams

Our interviews revealed that sharing diagrams was usually done over the Internet, (*e.g.*, a post to a website, or upload to a version control system).

I just created it in PNG and attached it to the wiki. That's publishing. [P5]

We put them on the website... You can take it from sourceforge and open it. We export the stuff and put it on the website. [P3]

P2 explained that uploading diagrams to a version control system made it easier to share them with others.

We put [diagrams] on the subversion system. That's the ideal way… We are really in the best practice. The best way is to put it on subversion. Everybody can keep the track of the changes on the documentations, the website, the presentations and all kinds of stuff. And it's centralized and everybody can access. [P2]

Most OSS project mailing lists prohibit or frown upon attaching diagrams in postings. P2 described how sharing diagrams is done within community conventions.

And also I sent it via IRC… It's a convention… You don't hook up the complete message you receive. So, by using an external website, for pasting text and images and that website creates a link. And that link, you can send it to the IRC website. Then you can explain what kind of information can be found at that link. So, all other people follow the text and the conversation can keep going on in the IRC channels. [P2]

Updating Diagrams

As we describe in The Role of Diagrams section, diagrams are often generated on the fly and shared for discussion. These same diagrams, therefore, do not necessarily lend themselves well for official documentation, where updates are necessary as the documented system is modified. Some participants did not consider this a significant problem.

If I would have to update the diagram or create a similar one, I would attach the source to the diagram in the wiki. So, I don't expect any problem on updating the diagram. [P5]

Some diagrams were automatically updated; for example, diagrams in Launchpad were updated automatically when a project dependency changed.

Well, it's automatically updated whenever anyone changes the dependency of any of the stacks that are described there. So, any stack can depend on any other stack, and it automatically draws the tree, the diagram by tracing those dependencies. [P4]

Updating diagrams seems to be a relatively rare practice according to our interviews, and sometimes undesired. For instance, P5 told us that when he creates a diagram, he does not expect that he or others will ever update it. These diagrams are part of the conversational record and therefore should remain static.

This specific diagram has not been updated. To be honest, I don't intend to update this diagram because at the time when I created the diagram, it was the same time as I created this blog post, a complete wiki page. And this helped the discussion with some friends of mine. [P5]

If diagrams are used as a part of the system documentation, they need to be updated. However, we did not find an example of regularly updated diagrams for documentation except those made by automatic diagram generation tools.

Two participants discussed potential problems updating diagrams, particularly in a collaborative manner.

What would be problematic is that someone else would take the source, and would modify on the earlier version of mine. You would have a problem if two separate edits on the same file and then you want to merge these two diagrams... Merging changes in a diagram would be exceptionally hard. [P5]

P5 also pointed out that this issue has been resolved for text files, including source files. However, the same solutions have not been successfully applied to diagrams.

We have experience with handling diversions of all packages and software, and we have a pretty clever solution of how to merge or how to handle the diversions. However, they don't really apply it to the diagrams or pictures but rather to… how to say… to plain text, to plain ASCII text in the source code. [P5]

P3 allowed others to update diagrams, but recognized that this could become a problem as the size of his team grows.

> *No. I don't mind in that case because there are a few people within that [team]. But if we had a huge number [of people] or it was open [to public], it (version control) would be definitely needed. [P3]*

Another related issue is establishing standards and norms for diagrams within a team. P4 explained that the wide variety of diagram tools and individual contributor's preferences made it difficult to reach consensus.

> *There's also how to edit, you know. Different people use different tools to create diagrams. And you wanna be able to communicate to everyone no matter how they are interacting with you. Anyone would be able to modify it. And it's kind of hard to agree on what tools to use. There are many many different tools that people use for diagrams and many different formats which you can use for diagrams… That's because it's a real hassle to deal with a mass of different programs and formats and most are proprietary. [P4]*

Updating diagrams also presents significant challenges for OSS communities, where the project history may not be captured as well or thoroughly as in co-located teams, or where there may be more turnover. Determining which diagrams are current and which are outdated is often difficult. As P4 points out, comparing two versions of a diagram can be challenging, at least compared to text files, for which tools exist.

> *Well, it's harder to compare two diagrams. You know, if I want to have difference on what you said versus what he said… When you use that program, the main goal of the diagram they fight is to try to compare two the totally different [diagrams]. [P4]*

Surviving Without Diagrams

Although participants agreed that diagrams were useful in general, they were not willing to or did not feel the need to use them in all cases, or anywhere near as frequent as seen by Cherubini *et al*. [2]. P3 discussed his reason for not using diagrams in his debugging tasks, claiming that sufficient information existed in other forms and sources.

> *There is not really a need for visual communication. Most of the information is located in bug reports or something like that. It's enough. There is no need for visual one in that case. [P3]*

Large tasks in OSS projects such as Ubuntu are usually divided into small tasks. In other words, a strict modularity strategy is adopted to limit the potential interactions any developer needs to consider with other modules.

> *It might be the case for people who work on a more complicated program to share diagrams that are standard for those programs. But that's not something that I do. [P4]*

P4, who was generally passive about using diagrams, explained his strategy for dealing with and figuring out existing code without diagrams.

> *Some systems that we tried out to explain code to somebody using pictures, you know, this function calls that function. Sometimes, that kind of things is useful but not nearly as often we think. What people tend to do is to share the code itself and then everyone can use their own favorite editors or integrated development environments, interactively move between, you know, who calls this, or who calls that. That interactivity may replace the desire for some static pictures… I use Emacs to edit the code or look at the code. Defined within the Emacs to… dependency things. Tools that are programmed with Python with things like IPython to inquire or find out what functions I have or what the arguments are. [P4]*

DISCUSSION

How Ubuntu Developers Use Diagrams

As discussed earlier, participants were more likely to use software, web-based systems, ASCII art, and screenshots to create diagrams than a piece of a paper. Thus, diagrams for Ubuntu are more likely to be digital than analog. This is different from what Cherubini *et al*. observed in the practices of co-located teams, driven of course by the need to share ideas with a large online developer community.

> *This is a drawing tool that allows me to do open source collaboratively. It must be shared. It has to be shared. [P1]*

This naturally leads to changes in the cost and appeal of diagramming, and how diagrams are used.

Why Ubuntu Developers Use Diagrams

Our study showed that five of the identified purposes of diagrams described by Cherubini *et al*. [2] were common online (*ad-hoc meeting*, *designing/refactoring*, *onboarding*, *hallway art*, and *documentation*) and two additional purposes existed partially (*understanding structures,* and *design review*). However, we could not find clear evidence for two other purposes (*explaining to secondary stakeholders,* and *explaining to users*).

Although we found evidence of diagram use for understanding the structure of a system or algorithm, participants did not report a case where they used diagrams to understand existing code. One possible reason is that participants have already established their own practices for examining code. This circumvents the perceived high cost of standardizing methods and conventions for creating and editing diagrams.

Likewise, we found partial evidence of the use of diagrams for design review. Beyond a reported instance of a design diagram that was reverse-engineered, no participant worked on system or user interface design, which may be one reason we could not find stronger evidence. Another reason

may be that design review usually involves updating diagrams, which most of our participants tried to avoid. After a disappointing experience with a paper sketch, P7 has developed a practice of sharing his ideas for user interfaces without using diagrams. He uses Glade [17], a user interface designer tool for GTK+ and GNOME. He makes a draft user interface in it, shares the source code, and lets other contributors revise it. The source code is an extended markup language (XML) file, which makes the sharing and updating process easier than with an image file.

> *If they can download source codes and look at it and look at the UI, then they can make tweak themselves. Instead of making suggestions and sending those suggestions in text which what they are doing if I have given them screen shots, whereas if I give them source, they can give suggestions and they can make changes themselves. [P7]*

Explaining to secondary stakeholders was another purpose absent from our study. One possible explanation for this is the strict modularity of the project. In general, a project is divided into many small parts, and they are usually well segmented. Each component is developed or maintained by a small group of developers or a single person. Every developer can assume that all those involved understand how the components interact with each other (*i.e.*, what the input is, or what format the output has). Therefore, there is less need for explaining the details of each component.

Similarly, though no clear case of diagram use to explain to customers was seen in our study, it might be a prevalent practice among contributors working in marketing or user support. For example, P2, who works for marketing, identified a gap in the level of investment in diagrams intended for internal *vs.* external use and publication. This points to the fact that developers are divorced from the task of communicating directly with end-users. This could of course lead to situations where the user documentation becomes out of synch with the reality of the software's inner workings.

Why Ubuntu Developers Don't Use Diagrams

We found conflicting attitudes towards the use and usefulness of diagrams in our study. On one hand, participants who actively used diagrams saw their value:

> *I'm a strong believer in creating good documentation with diagrams. One thing you always have to remember is that creating a model or a diagram of something, it's simplified [and] better. [P2]*

However, participants who were ambivalent towards diagrams often felt that text communication was sufficient.

> *What I am saying is that it's not that important to me. I don't see people saying, "Oh, boy, I really… this is... I just saw a diagram. Have you seen it right now?" People just use text to describe stuff they are working on. [P4]*

This is clearly different from the findings by Cherubini *et al*. [2], where co-located software developers generally agreed on the importance of diagrams and used them actively.

The Ubuntu Project *vs.* OSS in General

The findings presented in this paper are based on interviews with nine Ubuntu contributors, and may not carry over to other OSS projects. OSS projects span a wide range of organizational models, team sizes, and application areas, which makes it difficult to generalize.

The Ubuntu project is among a small minority of large (both in terms of code and contributor base), mature, and centrally managed OSS projects. Smaller OSS teams may feel less need for diagramming for instance. Our findings point to important differences in the practice of diagramming in a large distributed OSS project from that observed in co-located software teams, something which warrants further study, study which should explore the generalizability of our observations across OSS projects.

FUTURE DESIGN CONSIDERATIONS

Sharing the "Source Code" of Diagrams

One reason some participants gave for not using diagrams more frequently was that merging different versions of a collaboratively edited diagram is difficult due to the lack of appropriate tools. These problems made some participants turn to ASCII art, which though limiting and cumbersome, can be edited by anyone and handled by current version control systems. One potential solution could be to share the "source code" for a diagram, textual codes which define a diagram, such as the XML of Glade, rather than binary files. This would allow existing version control systems to track changes. However, any tool to create a diagram from the source code must be widely available, which might impose other problems; contributors may have different preferences for tools and conventions.

Even if diagrams are distributed as editable and track-able text representations as described above, and important hurdle remains in developing the tools that would merge and represent change in a visually obvious, but non-jarring way. Merging two diagrams with slight non-overlapping edits should not result in a complete re-distribution of the elements in the diagram unless strictly necessary..

Coexistence with Established Communication Channels

For OSS, diagrams need to be shared online. However, because the main communication channels used; emails and IRC, primarily focus on text, diagrams currently have to be uploaded to websites and their locations shared through those channels. Although it might be necessary to re-examine these conventions, it is important to avoid unnecessary disruption to these communities. Therefore, the process of sharing diagrams might have to continue along a parallel backchannel (for instance, a predetermined website). Establishing standards and managing expectations (such as promoting the use of a fixed site with known set of

features for the exchange of diagrams) could improve the situation.

Other Requirements

We discovered several other requirements for diagram tools in this domain. First, the tools themselves need to be free and OSS. The method for creating and editing diagrams must be accepted by a majority of OSS developers. Free software usually means no out-of-pocket cost for OSS developers, and would offer greater licensing compatibility than a proprietary product. There is also a high acceptance barrier that must be overcome, made lower if the tool itself is open source.

Many OSS projects are globally distributed. Thanks to advances in communication technology, many OSS developers have broadband Internet connections. However, there are still developers who have limited connectivity. Therefore, tools to support the collaborative generation and sharing of diagrams should not assume a pervasive broadband connection.

CONCUSIONS AND FUTURE WORK

Diagramming is a proven and valuable tool in software development. We interviewed nine Ubuntu contributors to understand how and why they used diagrams. We show that they had conflicting attitudes towards diagramming, and that even those who use diagrams in their work do not use them consistently, or for all of the purposes seen with co-located teams. We also explored how developers have adapted to working without diagrams, and discussed the reasons why diagrams were not actively used in some cases.

This study shows that the use and practices of diagramming is influential in OSS development, but that its use is far from simple at times. Further research is needed to develop a more comprehensive understanding of OSS diagram use. Deploying and testing a system designed based on our findings is another interesting research direction.

ACKNOWLEDGEMENTS

We would like to thank Gary Bader, Rhys Causey, and Jeremy Handcock for participating in our pilot interview. We also thank Steve Easterbrook and Jorge Aranda for helpful comments on this project, and David Dearman and Justin Ho for help on our paper. We thank all the participants in our study for their help and cooperation, and the greater Ubuntu community for making their materials and practices public for study.

REFERENCES

1. Bellotti, V., and Bly, S. Walking away from the desktop computer: distributed collaboration and mobility in a product design team. In *Proc. CSCW 1996* ACM Press (1996), 209-218.
2. Cherubini, M., Venolia, G., DeLine, Rob, and Ko, A. J. Let's go to the whiteboard: How and why software developers use drawings. In *Proc. of CHI 2007*, ACM Press (2007), 557-566.
3. Cohen, J. A. Coefficient of agreement for nominal scales. *Educational and psychological measurement 20*, (1960), 37-48.
4. Dekel, U. Supporting distributed software design meetings: What can we learn from co-located meetings? In *Proc. of HSSE 2005*, ACM Press (2005), 1-7.
5. Elliot, M., Ackerman, M. S., and Scacchi W. Knowledge work artifacts: kernel cousins for free/open source software development. In *Proc. of GROUP 2007*, ACM Press (2007), 177-186.
6. Guest, G., Bunce, A., and Johnson, L. How many interviews are enough?: an experiment with data saturation and variability. *Field Methods 18*, 1, (2006), 59-82.
7. Gutwin C., Penner, R., and Schneider, K. Group awareness in distributed software development. In *Proc. of CSCW 2004*, ACM Press (2004), 72-81.
8. Ko, A., DeLine, R., and Venola, G. Information needs in collocated software development teams. In *Proc. of ICSE 2006*, ACM Press (2006), 344-353.
9. Landis, J.R., and Koch, G.G. The measurement of observer agreements for categorical data. *Biometrics 33*, 1, (1977), 159-174.
10. Olson, J. S., and Teasley, S. Groupware in the wild: Lessons learned from a year of virtual collocation. In *Proc. of CSCW 1996*, ACM Press (1996), 419-427.
11. Poltrock S. E., and Engelbeck, G. Requirements for a virtual collocation environment. *Information and Software Technology 41*, 6, (1999), 331-339.
12. Redmiles, D., Hoek, A. V. D., Al-Ani, B., Hildenbrand, T., Quirk, S., Sarma, A., Filho, R. S. S., Souza, C. D., and Trainer, E. Continuous coordination: a new paradigm to support globally distributed software development projects. *Wirtschaftsinformatik 49*, (2007), 28-38.
13. Robertsa, J., Hann, I.-H., and Slaughter, S. Communication networks in an open source software project. In *Proc. of OSS 2006*, Springer Press (2006), 297-306.
14. Sawyer S., Farber, J., and Spillers, R. Supporting the social processes of software development teams. *Information Technology and People 10*, 7, (1997), 46-62.
15. Twidale, M. B., and Nichols, D. M. Exploring usability discussions in open source development. In *Proc. of HICSS 2005*, IEEE Computer Society Press (2005), 198.3.
16. Dia, http://live.gnome.org/Dia/
17. Glade, http://glade.gnome.org/
18. Gobby, http://gobby.0x539.de/
19. Launchpad for Ubuntu, https://launchpad.net/ubuntu/
20. Wiki for Ubuntu, https://wiki.ubuntu.com/