Beyond Pretty Pictures: Examining the Benefits of Code Visualization for Open Source Newcomers

Yunrim Park, Carlos Jensen
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97331
+1-541-737-2555
{parkyu, cjensen}@eecs.oregonstate.edu

Abstract

Joining an Open Source project is not easy. Newcomers often experience a steep learning curve dealing with technical complexity, lack of domain knowledge, and the amount of project information available for starters. This paper looks at the information needs of newcomers and the potential benefits of information visualization in supporting newcomers through a controlled experiment. Our results show that current OSS environments and development tools are lacking in support for the information needs of newcomers, and that existing visualization tools and techniques can help. We also discuss the potential problems and pitfalls associated with the inappropriate use of code visualization tools.

1. Introduction

A continuous influx of newcomers and their active engagement with development activities are crucial to the success of Open Source Software (OSS) projects. While many avid OSS users with a technical background express an interest in contributing to OSS projects as a developer, only a small proportion of them actually take steps to turn into active developers; the vast majority stays at the periphery of the community [19, 30, 34, 35]. One of the possible reasons for inactivity and withdrawal of membership might be that joining an OSS project implies a steep learning curve and requires a significant, and unpredictable, commitment of time and effort.

Participants need to learn about technical as well as the social conventions of the project, and acquire knowledge and skills necessary to make contributions. Because of the voluntary nature and loose structure of OSS teams, people joining an OSS project are usually expected to learn about the project on their own using publicly available resources, for instance, mailing lists, discussion boards, wikis, IRC, source code repositories, and issue tracking systems [37]. Some studies have concluded that the development environments and resources provided by OSS projects are sufficient to support the work of contributors [16, 20, 28, 44]. Sufficient however does not mean optimal, and the needs of those looking to join a project are different from those who are already engaged in the project. Circumstantial evidence for this can be seen in the low rate of newcomer success [34, 35].

This paper examines the potential benefits of tuning environments to meet the needs of developers joining a project. To this end, we conducted an empirical evaluation of current OSS development environments and tools, and how they support newcomers' learning process, and explored the potential benefits of existing information visualization techniques in the learning process. Known qualities of visualizations, for example, facilitating exploration and discovery in large datasets, may simplify the initial learning process and create a better environment for learners [39].

The rest of the paper is organized as follows: First, we review the challenges to joining OSS projects and the information needs of newcomers. Section 3 describes our controlled experiment, and we present the results in Section 4 before concluding with a discussion of lessons learned and future research.

2. Related Work

2.1. The process of joining an OSS project

Those joining an OSS project typically go through a series of steps before getting to the writing of code; learn about the software as an outsider, learn about the community by observing communication and ongoing activities, join discussion, report bugs, and finally fix bugs, and commit code. There is still little research on the joining and role transition process of OSS projects.

Perhaps the most in-depth and comprehensive studies of the process to date are [42] and [14].

Newcomers typically spend a significant period learning about the project before participating in any activities. Learning in this period is independent and characterized by "lurking" behaviors: Newcomers lurk on mailing lists, read previous discussions, and review public information to learn about people, ongoing activities, history and current status of the community. Lurking appears to be an essential part of the joining process, as newcomers acquire the social and technical knowledge expected of participants [31, 36, 42, 43]. After acquiring some knowledge about the project and its norms, newcomers start interacting with other members by joining discussions and take part in peripheral activities such as reporting or fixing bugs.

Observations of newcomers show that successful joiners do not typically start by introducing themselves or expressing their interest in contributing. Instead, they participate in ongoing technical discussions about existing problems [14, 42]. Their initial messages are not limited to suggesting technical solutions for problems but often include code, suggesting a high degree of technical knowledge and skills is required Successful joiners who make considerable contributions are promoted to active or core developers, which often includes the privilege of committing code to the repository [25]. Ducheneaut summarizes the process [14]:

"Through an initial period of observation, newcomers can assimilate the norms and values of the community and analyze the activity of the experts. To evolve any further, they have to start building an identity for themselves and become more visible to the core members. [...] learning involves the construction of identities and is itself an evolving form of membership".

Successfully joining an OSS project is a long process premised on the acquisition of substantial knowledge and skills. Newcomers should understand underlying technologies and the inner-workings of the code and familiarize themselves with the community, which requires significant time and effort.

Studies of the information needs associated with performing software development and/or maintenance tasks show that "awareness about artifacts and coworkers" is frequently needed by programmers [20, 27]. Members of a software development team need to maintain awareness of ongoing activities (e.g. "what is currently being discussed in mailing lists", "what changes are being made in the code", "what bugs have been reported") and other members (e.g. "who is working on what", "who do I need to communicate or collaborate with"), status of a project (e.g. "how much progress has been made since last release",), goals (e.g. "what is the big picture plan for the project"), and so on. They also need to understand the structure of the existing code base and the relationship between different elements [18, 38].

Sim, and Holt [17] equated joining a software project with moving to a foreign country for work – it requires "learning about the job, the local customs, and sometimes a new language." Newcomers, especially novice programmers, can experience a steep learning curve and need to fill a significant knowledge gap. In traditional software development teams these problems are mitigated through mentorship by existing members [16, 27]. Unfortunately, mentorship is rarely available in OSS projects, partly due to physical distance between members, and high turnover rate of contributors [4, 23]. The most common advice to newcomers is to learn through observation of public resources such as mailing lists, IRC, bug trackers, and source repositories [20, 42].

2.2. Development environments and information visualization

OSS development teams are collaboration-intensive online communities, whose goals cannot be achieved without a high-level of information sharing and task coordination. To overcome the problems associated with a lack of face-to-face interaction [23], contributors to OSS projects rely on a number of communication channels, including email, discussion boards, wikis, and IRC to maintain awareness and share ideas. Source code repositories, issue tracking systems, and archives are also frequently used to keep track of changes and design decisions, as well as to coordinate and share [37].

Although repositories and archives contain comprehensive information about projects, finding specific and useful information can be challenging and often time-consuming even for experienced members. The tools commonly are mostly text-based, and the amount of information generated on a daily basis can be very large. This information is usually a byproduct of collaboration and does not necessarily get organized for future reference [4].

A number of studies show that "lean media" such as email and IRC provide an effective way to share knowledge and maintain awareness, supporting close collaboration among collaborators in geographically distributed locations [16, 20, 28, 44]. Another group of studies, however, point to the limitations or challenges of lean media commonly used by OSS development teams. To support more efficient collaboration and information sharing, researchers have experimented with different tools that form group memory from development artifacts and provide information relevant to current tasks [5, 10], that connect information scattered across multiple repositories [1, 10], that sort information (semi-) automatically [2], or that support collaborative information building/sharing through annotations or tags [7, 33, 41].

Visualization provides an efficient way of presenting a large amount of data efficiently and facilitates sensemaking and information discovery by leveraging human visual and cognitive systems [39]. Effective use of visual representations can provide insight and uncover patterns that are not readily apparent. These benefits have caught the interest of research communities and a number of tools have been developed as reported in several surveys [3, 26, 40]. Those tools serve different purposes and address different needs, including assisting understanding of code structure and dependencies [15, 32], supporting work coordination of change requests and tracking changes in source code [9, 21], providing information on socio-technical dependencies [12, 13], and visualizing member activities to maintain awareness and collaboration [6, 8]. We believe that visualizations can assist learning and lower the up-front cost for joining a project as suggested in [20, 21, 22].

To date we see little mainstream adoption of such tools (one notable exception being Launchpads' Blueprints service (https://blueprints.launchpad.net/)). We believe one of the reasons for the low adoption is the lack of empirical evidence showing the benefits of introducing visualization tools to existing development environments [3, 40].

3. Methodology

A 3x2 between-subjects design was used, the independent variables being environment and gender. The experiment was conducted in a laboratory where participants performed four tasks that involve finding information about an OSS project. For the study, we used a real OSS project called "GanttProject" (http://ganttproject.biz/), which was selected from SourceForge.net based on the following criteria: (1) implemented in Java, (2) used CVS as a code repository, (3) had at least 10 core contributors, (4) had activity (90% or higher activity percentile at SourceForge.net), (5) reasonable size (10,000+ LOC), (6) development maturity (4-beta or higher stage at SourceForge.net), (7) age (2 years+). These criteria were set to ensure that the project was compatible with the visualization tools and had enough activity and artifacts to make the tasks challenging. Table 1 shows the tools provided to each experimental group.

SourceForge provides a set of standard tools and infrastructure common to most OSS projects, which served as a "default" toolset for the experiment and were available to all groups. Group A (our control) only had access to these resources. The experimental groups, B and C each had two additional visualization tools (see Table 1). Each set of tools served the same purposes but differed in feature sets (Table 2 and 3). The tools used by Group B were integrated into Eclipse, while Group C used standalone applications.

Table 1. Control vs. Experimental groups

Group	Default Resources	Visualization Tools	
		Change History	Code Structure
A	• Website & tools in SourceForge.net • GanttProject's own website • Eclipse IDE	N/A	
B		Version Tree	Creole
C		Augur	SA4J

- Version Tree (http://versiontree.sourceforge.net/)
- Creole [32] (http://www.thechiselgroup.org/creole)
- Augur [12]
- SA4J (http://www.alphaworks.ibm.com/tech/sa4j)

The selection of visualization tools involved a review of numerous tools designed to provide information about collaborators and change history as well as understanding code structure. Among many aspects of tools, ease-of-use was considered the most important criterion for selection to minimize the possible learning curve for subjects. Finally, we selected four visualization tools as shown in Table 1: Version Tree and Creole for Group B, and Augur and Structural Analysis for Java (SA4J) for Group C.

Table 2. Change history: Version Tree vs. Augur

	Version Tree	Augur
App. type	Eclipse plug-in	Standalone
Repository	CVS	CVS, Subversion
Features	Revision history in tree view. Supports code browsing and easy switching between revisions	Revision history in line-oriented view, line charts, bar charts. Information about author-activity, time-activity, other stats.
Visualization units	Individual files	Wide range from line-level to an entire project
Browse code	Yes	No

We grouped these tools along two axes: by type of application and by visualization content. Version Tree and Augur both visualize collaboration and change histories whereas Creole and SA4J visualize the structure of code and relationship between elements. The reason for having two types of applications (plug-ins vs. standalones) in our study was twofold, first, to investigate how integrated versus standalone tools affect performance, and second, to try to identify the features most useful to newcomers. Plug-ins are typically less complex than full applications, meaning less learning. Table 2 provides a brief comparison of Version Tree vs. Augur, and Table 3 Creole vs. SA4J.

We reviewed previous studies on the information needs in collaborative software development and selected the four most relevant [18, 20, 27, 38]. These studies examined collaborative software development and/or maintenance processes and identified information critical to performing these tasks.

Table 3. Code structure: Creole vs. SA4J

	Creole	**SA4J**
App. type	Eclipse plug-in	Standalone
Prog. lang.	Java	Java
Features	Code structure and relationship between elements in different views (nested, class interface hierarchy, call graphs, fan in/out, package dependencies)	Code structure and relationships between elements, provides analytic information (anti-pattern detection, stability analysis, package analysis, change propagation, quick filter, etc.)
Dependency information	Package dependencies (nodes and arrows), nested view, call graphs	Graphical: Dependency browsing, What if view Textual: Local/Global dependencies, Details

We grouped the types of information identified in these four sources into six categories: (1) Main contributors and their expertise, (2) History of changes, (3) Structure of code and relationship between different elements, (4) Impact of a change, (5) Control flow and execution behavior, and (6) Source code details. Using these categories, we defined tasks that would mimic real-life situations a novice with limited time (1 hour) could be expected to do. We derived four tasks for participants to do. The following are short descriptions of the tasks subjects were asked to complete:

Task 1. Find information about active contributors and change history at the project-level ("Project-Level")

1. Who are the most active code contributors? Name three.
2. Where in the code are they working?
3. How large is the code base of the project?
4. Find information about the changes to the CVS repository over the last 12 months.

Task 2. Find collaborators and change history at a package-level ("Package-Level")

1. Find the contributors who know about the package. Name three of them.
2. When was the most recent change made in the package and who made the change?
3. How has the package been changed since it was first created in terms of size?
4. What should you need to know to make a change in the package and to submit it?

Task 3. Understand code structure and relationship between elements within a package ("Within Package")

1. How many classes does "dependency" have? And how many interfaces?
2. Find the classes that are heavily interacting with other classes in the package "dependency"? Name two of them.
3. Suppose you want to change the class "SearchKey". Which classes of the package "dependency" will be directly affected by this change?
4. Which of them is the most dependent on "SearchKey"? Describe the dependency.

Task 4. Understand relationship and dependencies between two packages ("Between Packages")

1. How are the two packages related?
2. Which class of the package "task" is the most dependent on the package "dependency"?
3. Which class of the package "dependency" is the most dependent on the package "task"?
4. How and where would you find the information that might be useful in understanding the existing code?

Version Tree and Augur visualize the history of changes to code base, core contributors, and the number of contributions. These tools were potentially helpful for Tasks 1 and 2. Creole and SA4J visualize code structure and relationships between different elements. These supported Task 3 and 4.

27 students (18 males and 9 females) were recruited from the School of Electrical Engineering and Computer Science at Oregon State University. All had at least 2 years of experience with Java programming. Other than the intentionally high number of females, subjects were similar to the prototypical OSS developer [11, 24]. Participants were randomly assigned into the three groups, which were gender-balanced. Prior to the experiment, they received a quick introduced to the environment, provided a brief description of the OSS project, as well as a refresher on the Eclipse IDE. Those assigned to an experimental condition were given a short tutorial on the extra tools, explaining the purpose of the tools and demonstrating their key features. Participants were given time to familiarize themselves with the environment and tools.

Both qualitative and quantitative techniques were used to analyze performance. Qualitative analysis included questionnaires administered at the end of each task, comments made during and/or after experiment, tool usage and strategies identified from transcripts of sessions. For our quantitative analysis, two researchers graded the answer sheets collected at the end of each task The inter-rater reliability showed a high positive correlation (Pearson's correlation coefficient r=0.8464). Task completion (time) and correctness (scores) were analyzed using the Kruskal-Wallis non-parametric test because there was no assurance of normality in the distribution.

4. Results

The first two tasks focused on project status and history whereas the last two focused on understanding code, specifically dependencies. Thus, the tools used in the two sets of tasks differed. Subjects in Group A mainly used CVS Browse and the source code, which accounted for about 70% of their time on task in Tasks 1 & 2. Subjects in Group C predominantly used Augur (~80% of the time). Subjects in Group B consulted source code more than others and did not show a preference for one tool. Although Version Tree could help them find information about change history, subjects did not find the graphical representation that helpful. In fact, most subjects were turned off by its inability to visualize aggregate information about changes at the package or higher level. Choice of tools for Tasks 3 & 4 was more straightforward. Each group used one tool predominantly as shown in Table 4.

Table 4. Tool usage during Task 3 and Task 4 (Subjects across groups predominantly relied on one tool/resource.)

	Group A	Group B	Group C
Tools/Resources (Time spent: 30mins)	Source code (91.24%)	Creole (87.84%)	SA4J (96.39%)

Without additional support for code comprehension and analysis, finding dependencies for subjects in Group A meant heavy use of code inspection. Most subjects scanned dozens of files one by one to find pieces of code that seemed related to those in question. Some subjects used the built-in search function of Eclipse and looked for specific class names. Finding such information was easier for subjects in Group B & C who used Creole and SA4J. These tools analyze code structure and dependencies between elements on the fly and present them in various forms, reducing the users' cognitive load.

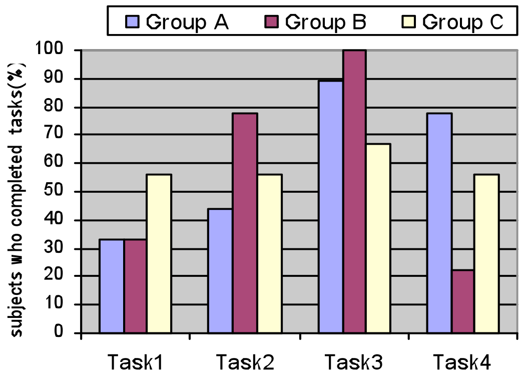

Figure 1. Number of participants completing each task

In a test examining the tradeoffs associated with learning a new tool, Figure 1 shows the number of participants who completed the entire task in the time allotted. Group C performed better in the first two tasks than Group A, but did worse in the last two tasks. Group B performed better than Group A in Task 3, but did significantly worse in Task 4 because of the tools' ("Creole") representation of different types of dependencies. While dependencies between elements within the same package (Task 3) were easy to perceive from the view generated by the tool, dependencies between two different packages (Task 4) were not as intuitive and required interaction with the view. This hurdle contributed to the different completeness between two tasks.

When examining the benefits associated with using code visualization tools, Figure 2 shows the mean task scores of each group (correctness). Group C scored higher than Group A throughout. Group B performed better than Group A in Task 3 and Task 4. Statistical analysis of task correctness (i.e. task scores) between groups (Kruskal-Wallis rank sum: two-sided, df=2) showed significant differences in the first three tasks; (1) Task 1: $\chi 2=8.7496$, $p<0.013$, (2) Task 2: $\chi 2=6.3274$, $p<0.043$, (3) Task 3: $\chi 2=7.9851$, $p<0.019$.

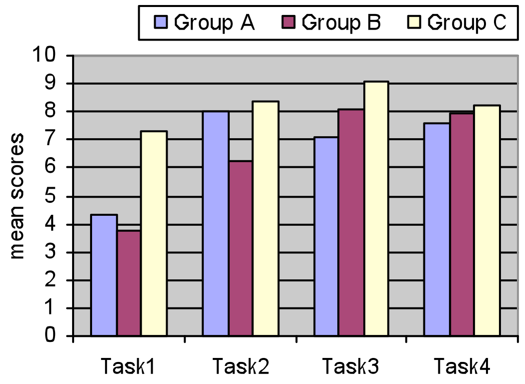

Figure 2. Mean score of each task

High mean score and task completion of Group C in Tasks 1 & 2 suggest that using Augur led to both increased efficiency and correctness. The dominant tool for Tasks 3 & 4 ("SA4J") on the other hand, improved quality but did not improve efficiency due to the complexity of the tool itself. Creole helped Group B with Task 3 in both completeness and correctness. Contrary to our expectations, we did not see clear differences between plug-ins and stand-alone apps.

As expected, our results show that visualization tools made it easier for participants to find information by presenting large amounts of data in an accessible form. Although some of the tools confused participants, resulting in lower completion in Tasks 3 & 4, this was not surprising given subjects' short exposure to the tools. Participants corroborated that they did not feel the tools were difficult to use and preferred using them over the default.

Differences in performance between the control group and the experimental groups came mostly from those questions requiring extensive search and analysis of large datasets. Table 5 provides examples that illustrate the difficulties associated with finding basic information using the default toolset.

Answering Question 1 was straightforward for subjects in Group C because Augur highlighted packages, files, and lines of code a specific author had recently touched. Finding this information from the CVS and bug tracker system was challenging as subjects had to examine a large number of change logs and bug reports to find names that appeared frequently.

Table 5. Mean scores of individual questions that showed significant statistical difference

(10 max, Kruskal-Wallis rank sum: two-sided, df=2)

Question	Mean Score & Statistical Difference		
	Group A	Group B	Group C
1. Where in the code are active code contributors working?	4.28	3.07	7.94
	$\chi^2=7.4225, p<0.025$		
2. How has the package changed in terms of size (LOC)?	2.38	1.88	8.21
	$\chi^2=7.2109, p<0.02$		
3. Which class is dependent on "X"? Describe the dependency	5.50	8.89	10.00
	$\chi^2=5.4135, p<0.067$		

Question 2 addressed the challenge of understanding the evolution of a large code base. While finding changes to a specific file or a particular revision number was straightforward for all groups, obtaining high-level information (e.g. changes made to a package or the entire project, changes committed over several months or years) was tedious and time-consuming for Groups A & B because tools like the CVS Browse page and Version Tree only provide information about individual changes to individual files. Augur, on the other hand, visualized the evolution of code on various levels (e.g. individual file, package, and entire project).

The last question required understanding of source code. Although the search built in to Eclipse helped to a certain extent, subjects in Group A had to switch from one file to another and read through the code to find dependencies, which was made even more difficult by poor documentation and lack of comments in the source code. Creole and SA4J were useful for answering these questions as they analyzed code structure and determined the degree of dependencies.

5. Discussion

Our controlled experiment revealed that the tools and resources available to current open source developers fail to extract information embedded in development artifacts. Difficulty obtaining such information may have negative impact on newcomers' motivation and efficiency. Our investigation of the impact of information visualization in this context suggests that providing visual information to newcomers may reduce the challenges they face when learning about a new project.

Comparisons of the mean scores show that subjects in the experimental groups performed generally better than those in the control group in terms of correctness, and that visualization tools provided the means for obtaining quality information. When it comes to task completion, Augur and Creole were particularly helpful. They provided more efficient ways to handle large amounts of data and understand dependencies in source code, reducing the learning curve and information overload experienced without these tools.

Visualization tools also received positive reviews. Many subjects mentioned that providing visual information such as the class diagrams or dependency views offered by Creole and SA4J would help new developers understand the structure of existing code and find problems to work on. "I could look at the source code. But then if I had a bigger problem, my solution, finding a solution would directly be dependent on my time and not using the tools. Because looking at the source code, which is manual work."

Most subjects in the experimental groups commented in the post-task questionnaire that they did not feel the tools were very difficult to use. They preferred using visualization tools over the default toolsets (with the exception of Version Tree). The high task completion rates reflect that the tools did not impose a high learning curve of their own.

The result of our experiment also suggests that visualization tools do not solve all problems. For instance, the limitations of Version Tree in visualizing changes (i.e. scope larger than individual files) not only made the tool less useful but also affected subjects' performance negatively. The performance of Group B in Task 4 shows the limitations and pitfalls of tools designed to serve limited information needs.

Graphical representations tend to be very domain- and task-specific as Koschke [29] states: "In order to find suitable ways of visualization, we need to understand when and why certain kinds of visualization work dependent upon viewers and task at hand. How do viewers 'read' and understand visualizations? And what are their real needs?" That is, properties such as the level of abstraction, forms of representation, types and scope of data visualized, granularity of information and interaction with views, are closely connected to the needs and types of tasks that the tools are intended to serve.

At the start of the experiment, we asked subjects to explore the default environment (i.e. websites, development artifacts) of the OSS project and report what they would wanted to know about the project before deciding to join. The kinds of information subjects sought included not only technical aspects, but also social aspects of the community and its members (e.g. who are they, how many people are working together, how friendly are they, etc.). Besides, subjects expressed a need for information specific to newcomers, for instance, how to get involved and become active (e.g. communication channels, available sources of information for starters, etc.), what to contribute to (e.g. open issues, required features, sample tasks to start with), and working practices.

Our experiment suggests that the default tools/resources did not provide efficient ways to meet the information needs of newcomers. Information relevant to the social side of the project typically requires “a big picture view.” Although information about past activities and current status of the project is available in forums, mailing lists, and bug repositories, participants were not sure where to begin and seemed overwhelmed by the amount of data they needed to sort through. The default toolset did not support generating an aggregated view of a large information space.

Many participants also mentioned a lack of information for new developers: “The SourceForge repository can be made more interactive and user-friendly. This would make it look less geeky/daunting for first-timers.” Some participants, despite being competent developers, sought to place blame on their own inexperience: “I’m not so familiar with open source…”, “for me, I haven’t worked on open source... so I don’t know where they have this statistic.” These comments lead us believe that existing collaborative environments for OSS development need to change to provide better support for newcomers.

Code visualization tools of different types can help create more novice-friendly environments. It is also possible that learning to use new tools may itself pose a hurdle to newcomers. The higher scores and lower completion rates of the experimental groups show this tradeoff between efficiency and effectiveness. While prolonged use of the tools would likely amortize the initial cost of learning how to use them, introducing new tools should be made based on a solid understanding of the needs of newcomers and a careful assessment of the associated risks and benefits.

6. Conclusion

This paper shows that visualization tools can be beneficial to those learning about an OSS project and help newcomers find information more efficiently and effectively. Even novices with little training were able to benefit from these tools. Providing more efficient ways to handle large amounts of information can lower the learning curve and information overload newcomers experience when joining an OSS project. The participants’ positive feedback on the visualization tools suggests that incorporating information visualization into existing collaborative environments may lead to enhancing the joining process. We believe that features that help deal with existing development artifacts and extract useful information from them should be adopted by OSS projects.

7. References

[1] Ankolekar, A., Sycara, K., Herbsleb, J., Kraut, R., and Welty, C. Supporting online problem-solving communities with the semantic web. In Proc. of the 15th international Conference on World Wide Web. WWW '06. ACM, New York, NY.

[2] Anvik, J., Hiew, L., and Murphy, G. C. Who should fix this bug?. In Proc. of the 28th Int. Conf. on Software Engineering. ICSE '06. ACM, New York, NY.

[3] Bassil, S. and Keller, R. K. "Software Visualization Tools: Survey and Analysis," icpc,pp.0007, 9th Int. Workshop on Program Comprehension (IWPC'01).

[4] Berdou, E. 'Managing the bazaar: Commercialization and peripheral participation in mature, community-led F/OS software projects.' Doctoral Dissertation. London School of Economics and Political Science, Department of Media and Communications.

[5] Berlin, L. M., Jeffries, R., O'Day, V. L., Paepcke, A., and Wharton, C. Where did you put it? Issues in the design and use of a group memory. In Proc. of CHI '93 Conference on Human Factors in Computing Systems. CHI '93. ACM, New York, NY.

[6] Biehl, J. T., Czerwinski, M., Smith, G., and Robertson, G. G. FASTDash: a visual dashboard for fostering awareness in software teams. In Proc. of the SIGCHI Conference on Human Factors in Computing Systems. CHI '07. ACM, New York, NY.

[7] Cadiz, J. J., Gupta, A., and Grudin, J. Using Web annotations for asynchronous collaboration around documents. In Proc. of the 2000 ACM Conference on Computer Supported Cooperative Work. CSCW '00. ACM, New York, NY, 309-318.

[8] Cheng, L., Hupfer, S., Ross, S., and Patterson, J. Jazzing up Eclipse with collaborative tools. In Proc. of the 2003 OOPSLA Workshop on Eclipse Technology Exchange. ACM, New York, NY.

[9] Collberg, C., Kobourov, S., Nagra, J., Pitts, J., and Wampler, K. A system for graph-based visualization of the evolution of software. In *Proc. of the 2003 ACM Symposium on Software Visualization*. SoftVis '03. ACM, New York, NY, 77-ff.

[10] Cubranic, D., Singer, J., and Booth, K. S. Hipikat: A Project Memory for Software Development. IEEE Trans. Softw. Eng. 31, 6.

[11] David, P.A., Waterman, A., and Arora, S. FLOSS-US the free/libre/open source software survey for 2003, (online) http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf.

[12] de Souza, C., Froehlich, J., and Dourish, P. Seeking the source: software source code as a social and technical artifact. In Proc. of the 2005 international ACM SIGGROUP Conference on Supporting Group Work. GROUP '05. ACM, New York, NY, 197-206.

[13] de Souza, C. R., Quirk, S., Trainer, E., and Redmiles, D. F. 2007. Supporting collaborative software development through the visualization of socio-technical dependencies. In Proc. of the 2007 international ACM Conference on Supporting Group Work. GROUP '07. ACM, New York, NY, 147-156.

[14] Ducheneaut, N. Socialization in an Open Source Software Community: A Socio-Technical Analysis. Computer Supported Cooperative Work (CSCW), 14(4):323–368, 2005.

[15] Eick, S., Steffen, J., and Sumner, E. SeeSoft: A Tool for Visualizing Line-Oriented Software Statistics. IEEE Trans. Software Engineering, 18, 11 (1992), 957-968.

[16] Elliot, M.S. and Scacchi, W. Free software developers as an occupational community: Resolving conflicts and fostering collaboration. In GROUP ’03, 2003.

[17] Sim, S. E.. and Holt, R. C. 1998. The ramp-up problem in software projects: a case study of how software immigrants naturalize. In Proc. of the 20th international Conference on Software Engineering. IEEE Computer Society, Washington, DC, 361-370.

[18] Erdos, K. and Sneed, H.M. Partial Comprehension of Complex Programs (Enough to Perform Maintenance), Proc. of the 6th International Workshop on Program Comprehension, Ischia, Italy, 1998.

[19] Ghosh, R.A. and Prakash, V.V. The Orbiten Free Software Survey. First Monday, 5(7), July 2000, http://www.firstmonday.org/issues/issue5_7/ ghosh

[20] Gutwin, C., Penner, R., and Schneider, K. 2004. Group awareness in distributed software development. In Proc. of the 2004 ACM Conference on Computer Supported Cooperative Work. ACM, New York, NY.

[21] Halverson, C. A., Ellis, J. B., Danis, C., and Kellogg, W. A. 2006. Designing task visualizations to support the coordination of work in software development. In Proc. of the 2006 Conf. on Computer Supported Cooperative Work. CSCW '06. ACM, New York, NY, 39-48.

[22] Heer, J., Viégas, F. B., and Wattenberg, M. 2009. Voyagers and voyeurs: Supporting asynchronous collaborative visualization. Commun. ACM 52, 1 (Jan. 2009), 87-97.

[23] Herbsleb, J.D., Mockus, A., Finholt, T.A. and Grinter, R.E. Distance, dependencies and delay in a global collaboration. In Conference on Computer Supported Cooperative Work (CSCW). 2000. 319- 328.

[24] International Institute of Infonomics, University of Maastricht, The Netherlands, and Berlecon Research GmbH, Berlin, Germany, Free/Libre and Open Source Software: Survey and Study, Final Report, 2002.

[25] Jensen, C. and Scacchi, W. Role Migration and Advancement Processes in OSSD Projects: A Comparative Case Study, Proc. of the 29th Int. Conference on Software Engineering, 2007.

[26] Kagdi, H., Collard, M. L., and Maletic, J. I. 2007. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *J. Softw. Maint. Evol.* 19, 2 (Mar. 2007), 77-131.

[27] Ko, A., DeLine, R., and Venolia, G. Information Needs in Collocated Software Development Teams, Proc. of the 29th Int. Conf. on Software, 2007.

[28] Kock, N. and Davison, R. Can Lean Media Support Knowledge Sharing? IEEE Transactions on Engineering management, Vol. 50, No. 2, May 2003

[29] Koschke, R. Software Visualization in Software Maintenancee, Reverse Engineering, and Re-engineering: A Research Survey. Journal of Software maintenance: Research and Practice, Vol. 15, 2, 2003

[30] Lakhani, K., and von Hippel, E. How Open Source Software Works: Free User-to-User Assistance, Research Policy, Vol.32, 2003, 923-943.

[31] Lampe, C. and Johnston E. 2005. Follow the (Slash) dot: Effects of Feedback on New Members in an Online Community, GROUP’05, November 6–9, 2005, Sanibel Island, Florida, USA, 2005.

[32] Lintern, R., Michaud, J., Storey, M., and Wu, X. 2003. Plugging-in visualization: experiences integrating a visualization tool with Eclipse. In Proc. of the 2003 ACM Symp. on Software. ACM, New York, NY.

[33] Lougher, R. and Rodden, T. 1993. Supporting long-term collaboration in software maintenance. In Proc. of the Conference on Organizational Computing Systems. S. Kaplan, Ed. COCS '93. ACM, New York, NY

[34] Mockus, A., Roy, T.F. and James, D.H. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla, ACM Transactions on Software Engineering and Methodology, 11, 2002.

[35] Moon, J.Y. and Sproull, L. Essence of Distributed Work: The Case of the Linux Kernel, First Monday, 5(11). November 2000, http://www.firstmonday.org/issues/issue5_11/moon/

[36] Nonnecke, B., and Preece, J. 2002. Silent participants: Getting to know lurkers better. In C. Lueg & D. Fisher (Eds.), From Usenet to CoWebs: Interacting with Social Information Spaces, Springer.

[37] Scacchi, W., 2002. Understanding the Requirements for Developing Open Source Software Systems, IEEE Proc.-Software, 149(1), 2002.

[38] Sillito, J., Murphy, G.C., and De Volder, K. Questions Programmers Ask During Software Evolution Tasks, Proc. of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, Oregon, USA, 2006, 23-34.

[39] Spence, R. 2007. Information Visualization. Pearson Education Limited, Harlow, England, 2nd edition, 2007.

[40] Storey, M. D., Čubranić, D., and German, D. M. 2005. On the use of visualization to support awareness of human activities in software development: a survey and a framework. In Proc. of the 2005 ACM Symposium on Software Visualization. ACM, New York, NY.

[41] Storey, M., Cheng, L., Bull, I., and Rigby, P. 2006. Shared waypoints and social tagging to support collaboration in software development. In Proc. of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work. CSCW '06. ACM, New York, NY, 195-198.

[42] von Krogh, G. Spaeth, S. and Lakhani, K. "Community, Joining and Specialization in Open Source Software Innovation: A Case Study", Research Policy, 32, 2003, 1217-1241.

[43] Whittaker, S., Terveen, L., Hill, W., and Cherny, L. 1998. The dynamics of mass interaction. In Proc. of the 1998 ACM Conference on Computer Supported Cooperative Work. CSCW '98. ACM, New York, NY.

[44] Yamauchi, Y., Yokozawa, M., Shinohara, T., and Ishida, T. Collaboration with Lean Media: How Open-Source Software Succeeds. CSCW’00, December 2-6, 2000, Philadelphia, PA.

[45] Yatani, K., Chung, E., Jensen, C., and Truong., K. N. "Understanding How and Why Open Source Contributors Use Diagrams in the Development of Ubuntu" In Proc. of the SIGCHI Conf. on Human Factors in Computing Systems. CHI ‘09. April 2009.