The Onion Patch: Migration in Open Source Ecosystems

Corey Jergensen, Anita Sarma
Computer Science and Engineering Department
University of Nebraska, Lincoln
Lincoln, NE, USA 98588
{cjergens, asarma}@cse.unl.edu

Patrick Wagstrom
IBM TJ Watson Research Center
19 Skyline Drive, Hawthorne
NY, USA 10532
pwagstro@us.ibm.com

ABSTRACT
Past research established that individuals joining an Open Source community typically follow a socialization process called "the onion model": newcomers join a project by first contributing at the periphery through mailing list discussions and bug trackers and as they develop skill and reputation within the community they advance to central roles of contributing code and making design decisions. However, the modern Open Source landscape has fewer projects that operate independently and many projects under the umbrella of software ecosystems that bring together projects with common underlying components, technology, and social norms. Participants in such an ecosystems may be able to utilize a significant amount of transferrable knowledge when moving between projects in the ecosystem and, thereby, skip steps in the onion model. In this paper, we examine whether the onion model of joining and progressing in a standalone Open Source project still holds true in large project ecosystems and how the model might change in such settings.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management: *programming teams*: D.2.8 **[Software Engineering]**: Metrics: *process metrics*

General Terms
Human Factors, Management, and Measurement.

Keywords
Open Source software, project ecosystem, contribution model.

1. INTRODUCTION
Open Source projects have been widely studied by researchers in the fields of software engineering, computer supported cooperative-work, and management [7, 28]. Research has included studies of motivational factors that drive volunteers to contribute time and code towards the common public good [18, 22, 31], the socialization process through which newcomers become active community members [10, 29], the sustainability of Open Source projects given their high rates of turnover and barriers to contribution [27], and the underlying social organizations in Open Source projects [2-3, 24].

One of the overarching attributes of Open Source is that it draws expertise and contributions from a pool of volunteers. Because these volunteers often exhibit high turnover rates, there is a need to understand how projects continue to recruit, educate, and socialize volunteers to maintain vibrancy [27, 31]. The past characterization of the development (or socialization) process maintains that new comers start at the periphery with low technical skill requirements, for example by posting to project mailing lists or participating in project chat rooms. As the skills and experience of the user evolves they may choose to report bugs, which requires a small amount of technical skill. Through their contributions users continue to build their reputation in the community and some may migrate toward more technical and central roles such as code contribution and moderation. This model, called the onion model, depicts roles as concentric layers with high skill, high reputation roles at the center and low technical skill and reputation at the periphery [6, 19, 32]. Variations on the model have been qualitatively and empirically validated in a number of projects (e.g., Apache [25], Freenet [29], Netscape [19], Mozilla [26], Python [10], and others [28]).

These studies form the foundation of our understanding of Open Source software. The existing characterization of Open Source projects is based on studies of large individual projects. However, these characterizations may be somewhat outdated in light of recent changes. Many modern Open Source projects strongly resemble large enterprise products that comprise numerous smaller, related projects, engage individuals as wells as corporations, and involve contributions from volunteers as well as paid members [4]. This new genre of Open Source, termed OSS 2.0 by Fitzgerald [12], is significantly different from its standalone individual antecedents. Earlier lessons and insights about Open Source projects might not hold true in this new collaborative landscape populated by complex software ecosystems and further studies to characterize this emergent OSS 2.0 phenomenon are required [4, 12]. For example, the Eclipse Foundation, which has its roots in the Eclipse Integrated Development Environment (IDE), which was released as Open Source by IBM, has expanded to encompass a wide variety of end user tools based on a common set of technologies beyond just the IDE experience, and include tools for developing complex web applications on the client and server, version control tools, and identity management. The foundation has numerous and significant contributions from some of the largest players in technology including IBM, Intel, and Oracle, in addition to substantial participation from volunteers and university students [9, 30]. Open Source ecosystems may have a variety of different focuses, such as GNOME and KDEs efforts to create desktop environments [23], developer tools and web infrastructure from Apache [22, 26], and support for Open Source programming languages [4], among others.

While past studies provided insight into how Open Source communities function, most do not address the interconnected nature of modern OSS 2.0 style ecosystems. Since projects in an ecosystem share underlying technical infrastructure and often follow similar social norms, members can participate in multiple projects or move across projects in an ecosystem with relative ease. The socialization process of the onion model may not hold true if developers can easily move from one project to another and utilize

ESEC/FSE'11, September 5-9, 2011, Szeged, Hungary.

much of their knowledge and reputation from elsewhere in the ecosystem. This leads to our first research question:

RQ1: To what extent do members migrate across projects in software project ecosystems?

When a developer joins a project from elsewhere in the ecosystem, rather than starting the socialization and technical knowledge acquisition process from scratch, it is likely that some of their reputation and technical skill will allow the developer to bypass portions of the basic socialization process. The manifestation of this knowledge transfer process, however, is unknown. Does existing knowledge allow a developer to bypass the socializations process, or merely compress the duration? This gives rise to our second research question:

RQ2: When members migrate across projects can they use knowledge common across projects to jumpstart their contribution to a new project?

Finally, previous research has suggested that experience and tenure in an individual project is a driving force in moving developers toward the center or core of a project [3, 26]. This is predicated on developers taking time to learn the code and build up reputation in the community. However, developers in an OSS 2.0 ecosystem are not only able to transfer general process information and reputation from one project to another, but may be able to reuse their technical expertise in shared code components or social expertise in enhancing a particular area. This can improve their ability to directly contribute to the core of a project. In such an environment, where developers can fluidly move from one project to another related project it is not known whether experience and tenure within a project are the primary factors affecting the centrality of contributions or if other factors overshadow these. This brings us to our third research question:

RQ3: In an interconnected software ecosystem, what factors affect the contribution type and quality?

We answer these questions through a longitudinal analysis of a selection of projects in the GNOME ecosystem. These are stable and mature Open Source projects that have attracted significant contributions from volunteer and commercial developers. These projects provide a rich data source because they all have between six to ten years of project archival history. The projects also have a significant overlap of developers, that is, there are many developers who have contributed to multiple projects in the period of our study. For each of these projects, we analyzed mail messages from project mailing lists, comments and actions from the project bug trackers, and code contributions made through the project version control systems. We further refined contributions through the version control system into source code contributions and other types of contributions, such as translation, documentation, and media. This allows a further differentiation of project members and a more robust analysis of developer progression paths and the centrality of developer contributions.

We find that within our sample there is a significant population that is active on multiple projects within the ecosystem. Based on an analysis of patterns that developers follow in joining individual projects and when participating in the entire ecosystem, we find that there is little evidence of individuals following the pure onion model. Rather, we identify multiple patterns that are contradictory to the model or otherwise compress the model. For example, a large majority of developers only made technical contributions to the project. In the subset of the six projects that we analyzed we found 81.65% of members participating in only technical medium, of which 64.67% were active only in source code repositories.

We also note, that in most cases prior experience in a project ecosystem does not have an effect on the centrality of contribution when controlling for tenure within a project. In fact, the longer a developer is associated with a project, the lower the centrality of their contributions. After further generalization of this behavior we find that new developers and those who have been active for between 2 and 5 releases both have about the same level of centrality for contributions to project source code, while very experienced developers tend to move away from direct code creation tasks, leading to lower centrality of contributions.

Finally, we attempt to uncover broader trends that may lead to a higher centrality of contribution. We find that there are certain very specific domains, specifically translation and internationalization, where there appears to be transferrable knowledge across projects in the ecosystem that allows a developer to quickly make high centrality contributions to project source code.

The rest of the paper is organized as follows: In Section 2, we discuss some of the background on the socialization process in Open Source projects and build our hypotheses. Sections 3 and 4 describe our data and present our analysis. In Section 5, we discuss our findings. Sections 6 and 7 close out the paper with a discussion of possible threats to research validity and our conclusions.

2. SOCIALIZATION PROCESS IN OSS

Prior studies have identified a variety of barriers that newcomers face in the course of immigrating to a new Open Source project [11, 18]. Open Source projects typically do not provide formal mentoring and training for newcomers and it is the responsibility of the newcomer to identify the appropriate technical tasks and start contributing [10, 29]. Most projects have a public list of open bugs and issues and newcomers are encouraged to start their investigations there or by addressing a concern the individual developer has identified. It is rare that newcomers are specifically directed to technical tasks. For example in an analysis of the Freenet project, von Krogh et al. found that only 1 in 6 newcomers were given specific technical tasks to work on [29]. Instead, a majority of newcomers were given general encouragement after expressing an interest in joining the community through the mailing list. Further, many projects lack an explicit architecture or system design, making it difficult for newcomers to understand the system before they can start contributing [10]. Finally, irrespective of the depth of technical knowledge that a user may possess, making significant technical contributions to a community requires social standing and identity in the community. In most projects, commit access is only given after a newcomer has proved their worth and potential to the active community members; a process that limits the overall potential contributions of newcomers to the project [10, 29].

The process through which newcomers gain access rights and become code contributors have been studied by many researchers [1, 6, 19]. The most common Open Source development model is called the onion model. This model postulates that members in an Open Source community have different roles ranging from peripheral users to core contributors and these roles are arranged as concentric layers – forming layers in the onion. More specifically, the following roles have been suggested (progressing from most central and most technical layer to outer layers that are the least technical): project leader, core developer, active developer, bug

fixer, bug reporter, documenters, users (active in mail messages), and peripheral user.

von Krogh et al. conducted a qualitative study of the transition of roles in Open Source and proposed the concept of a "joining script" for new developers joining a community [29]. They categorized members into three broad groups: *joiners* are members who are active only in mailing lists, *newcomers* are members who have just gained commit access, and *developers* are active members with commit access who have shown strength of contributions and a technical ability. Potential developers (*joiners*) start by joining project mailing lists that allow them to converse about the project and learn some of the social norms and technical capabilities of the project. As they participate for extended periods of time potential developers learn how to properly participate in the community by submitting bugs, triaging bugs, and eventually working to track down the technical details of bugs by submitting small patches. After a *joiner* has shown competence with managing bugs they may be offered the ability to become a committer (*newcomer*) to a project, which allows them to directly modify the project source code without the need of an intermediary. After an intermediary trial period *newcomers* are considered to have transitioned to *developer*, if no major concerns were raised.

In a complimentary study, Duchenaut identified trajectories for individuals based on successful stories of Open Source developers [10]. One such trajectory has the following stages: (1) peripheral monitoring of activity, (2) bug reporting and patch suggestions, (3) commit rights and bug fixing, (4) module level leadership, (5) becoming vested in the community, and (6) gaining approval of core members for far reaching (architectural) changes. His study suggests that to succeed in becoming a part of the community there are social "rites of passage" at each stage in which peripheral members must gain the acceptance of core members and that political maneuverings are often needed to create an identify for oneself and gain acceptance from the community leaders.

While no consistent naming scheme has arisen for roles in Open Source projects (e.g., maintainers instead of core member, patchers instead of bug fixers [10, 32]), a consistent finding is that members near the center of the model exert more influence over the technical decisions of the project as well as other factors affecting the community [3, 10, 22]. For example, in Linux, the project leader, Linus Torvalds, has the final say regarding technical directions. In Apache, the board of directors forms the core layer and is responsible for making final decisions regarding project plans and features. The onion model of role progression is considered meritocratic and as members gain experience and make larger contributions to the project they migrate to more central roles in the community [32]. This general model of immigration and participation in Open Source projects as a process of moving from non-technical to technical processes provides a foundation to our first hypothesis:

Hypothesis 1: New comers to project communities will begin by participating in the least technically challenging medium, before moving to more technical mediums.

Other studies have also shown the importance of social factors for the success in Open Source projects. Oh and Jeon [27] found that the social network and the strength of the ties in the community was a good indicator for retention of members in the community in the face of external factors such as other projects, monetary incentives, etc. Bird et al. found that in the Open Source communities that they studied (Apache, Postgres), attaining developer status was dependent on the tenure of that individual in the project and that the social status of an individual was a stronger criterion for success [2]. They identified the inherent social structures in the community based on mail messages and found that successful members were also social hubs. In their seminal study von Krogh et al. found that developers who had generational knowledge (active across multiple releases) made more far-reaching changes, whereas new developers largely made localized changes [29]. This leads to our second hypothesis:

Hypothesis 2: As developers gain more experience in a project they will contribute more to the core of the project source code.

The central premise of the onion model is that the progression from a passive user to an active developer entails a learning process, both from a social and technical perspective. However, when projects are interrelated the time needed to learn the social culture or technology might be lower. Project ecosystems often constitute projects that are heavily interrelated. For example, Eclipse hosts a multitude of projects that all are built on a common technology and utilize a common development infrastructure [9]. Similarly, Apache contains numerous common libraries that are shared amongst projects written in both Java and C. It also has a formal process for development and participation, the Apache Way, which describes how developers are to communicate and manage projects [13, 22]. GNOME, a successful desktop environment for Linux and Unix systems, likewise has a consistent infrastructure across projects that contain common libraries and widgets that allow developers to leverage knowledge gained in related projects [16]. Such common infrastructure, therefore, should reduce the amount of new technical and social knowledge that must be acquired when moving between projects. This is the premise of our third hypothesis:

Hypothesis 3: Developers who have been active on related projects in the same ecosystem will be able to transfer knowledge and reputation to short-circuit the onion model of participation and contribute to the core of a project sooner than those who have not.

In summary, the immigration process in Open Source projects has a strong social component. However, the majority of these studies have been performed on individual projects. To the best of our knowledge we are the first to study the development process model in an Open Source ecosystem. Prior work by Dagenais et al. [8] investigates how newcomers get on board new projects within a corporate environment by studying how they learn about the technical landscape and the social culture of individual projects. Findings from the study recommend mentoring guidance, frequent feedback, and creating a project landscape with (technology) markers to make it easier for newcomer to understand the system. Although, this study is for a commercial project, it relates to our work since it investigates migration across projects in a community. However, its findings are not fully applicable to Open Source ecosystems, which have very different characteristics with much less hands-on training and feedback provided to newcomers.

3. DATA COLLECTION

For our analysis, we examined the ecosystem around the GNOME project, an effort to create a robust and usable desktop environment for Linux and other Open Source operating systems. Founded in 1997, GNOME has a fairly open policy of accepting new projects into the ecosystem, which gives the project the ability to use GNOME servers for infrastructure needs. Throughout the history of GNOME there have been more than 1,200 different

projects -- many of which are smaller projects that never made it into the official distribution of GNOME.

3.1 Background of GNOME

GNOME is built on a set of common technologies and libraries that include, among others, a common graphical user interface toolkit with associated user interface guidelines; components for common tasks such as displaying images, libraries for managing program configuration and processing XML files, and mechanisms for translation across the ecosystem. These shared technologies do not, however, enforce a required programming language or set of programming paradigms. The project uses a number of different programming languages for key components including C, C++, Python, C#, and in some older cases, scheme [15-16].

In addition to common technical interfaces, the ecosystem also has a shared environment for managing the technical and social parts of a project. It provides a common hosting framework for project source code (originally CVS, later Subversion, and now git [17]), defect and request management (Bugzilla), and discussion and decision making (mailing lists and real-time chat). The project has an overall foundation board that manages the major directions and business aspects of the ecosystem, but individual projects are given significant amounts of autonomy. Individuals within the community are elected or appointed to major roles in the ecosystem that cross project boundaries, such as release manager [16, 21].

To make our analysis tractable and test our hypotheses, we had to filter the community down to a subset of projects that have multiple releases as part of the official GNOME desktop and have a significant number of developers, bug reporters, and people active on project mailing lists. Further, to understand immigration across projects we needed to select those projects that have a significant overlap in project membership.

3.2 Data Collected

We collected data for the GNOME project from 1997 to 2007, including data from mailing list archives, bug tracking system, and source code repository. In total, more than 1,000 developers made nearly 2.5 million changes to files grouped into approximately 480,000 commits. We worked with the project administrators to obtain a copy of the complete bug database for the project, which contained 790,000 comments on 250,000 bugs, reported by 26,000 different people. This data was loaded into a large database with a single schema that integrates all of these data streams.

As is common with many long-running Open Source projects, the different data streams were not seamlessly integrated with one another, with individuals using different account names for project mailing lists, bug trackers, and source code repository. One of the authors worked with members of the community, and utilized information from norms and practices, such as referencing bug numbers in source code commit messages, to link together all the elements. The most difficult part of cross-linking the GNOME data was in normalizing user names across databases. While, a large part of the normalization process was automated (matching performed by comparing email addresses and provided user names across data sources), it was necessary to consult with individuals in the community to correctly identify and validate the names and identities for about 10% of the participants.

The openness of the community also means that it is easy for anyone to sign up for project mailing lists and report bugs, yielding thousands of individuals with only peripheral interest in the community. As we are concerned with the immigration process through which a member becomes a committer, we examine only those individuals who eventually obtained direct commit access to the project source code. Furthermore, as the community keeps almost all file-based artifacts in project code repositories, including translations and other non-code related files, we had the opportunity to investigate whether members whose contributions are non source code (e.g., translators, documenters, and artists) follow a different path to become a committer. We, therefore separately analyze contributions made to the project source code repository according to the type of artifact contributed. That is, we differentiate between source code, project documentation, project build scripts, translations, and other artifacts. For the purpose of our study, we are most interested in the two largest categories of artifacts, actual project source code (e.g. C, C++, python, etc) and project translations and documentation.

For our analysis, we selected a subset of six projects. These projects were selected on the basis of their extensive history, availability of archival artifacts, prominence in the ecosystem and the overlap of developers between these projects. Three projects are end user applications and three are utility or library packages:

- Project 134: An end user application for viewing and lightweight graphics manipulation.
- Project 135: A web browser that is customized to integrate into the desktop environment.
- Project 190: A library and several tools for applications to manage settings in a standard and unified method. Most end user applications in the ecosystem rely on this library as a critical piece of infrastructure. All projects in the subset we examine utilize this library as a key component.
- Project 377: A collection of utilities for developers and end-users alike to make the most out of their desktop experience.
- Project 378: A system level library for the transparent manipulation of files and other file-like resources on the local machine and across network connections. The use of this library is not required by all end user applications; however, all applications in this subset utilize this library.
- Project 405: An extensible end user spreadsheet application.

4. ANALYSIS

To understand the effect of ecosystems and interconnectedness of projects on the "joining script" of members, we began by analyzing the overlap of individuals who committed code to multiple projects in the ecosystem. In the matrix shown in Table 1, the diagonal shows the total number of unique individuals who were identified as contributing source code (as opposed to translations, documentation, and media) to the project source code repository during the period of study and other cells show the number of developers in common between the two projects.

Table 1: Overlap of Source Code Committers Between Projects in Study

	134	135	190	377	378	405
134	102	32	70	54	73	45
135		85	41	37	44	22
190			148	73	97	54
377				163	74	63
378					175	58
405						124

Despite the broad spectrum of projects in our study, we see that there is a significant overlap of individuals contributing source code to the projects. In fact we see 97 developers who are common across projects numbered 378 and 190. When we expand our observations to include all those who made contributions to the project source code repository (including documentation and translations), we see that, indeed, there is a much greater overlap between projects as seen in Table 2.

Table 2: Overlap of All Committers Between Projects in Study

	134	135	190	377	378	405
134	210	120	162	163	164	107
135		154	122	123	121	69
190			225	169	181	109
377				281	166	133
378					261	112
405						187

Going beyond project source code repositories and including other major project archival mediums (mailing lists and bug tracker), we see that each of these projects attracted significant numbers of contributors and engaged users, many of whom were also active in other projects within the ecosystem as shown in Table 3.

Table 3: Overlap of Participants in Mailing Lists, Bug Tracker, and Source Code Repository Between Projects

	134	135	190	377	378	405
134	369	216	244	264	273	175
135		716	222	226	285	162
190			541	263	311	189
377				475	290	203
378					690	211
405						1,085

Having established that each of these projects within the GNOME ecosystem have a significant number of contributors and that there is significant overlap between individuals working on each project, in the remainder of this section we analyze our research questions: first by examining the pattern of interaction that leads a new contributor to become a developer in section 4.1, followed by an analysis of how project tenure affects the centrality of developer contributions in section 4.2, an evaluation of ecosystem tenure in section 4.3, and a principle component analysis to identify different factors that effects centrality of developer contribution in section 4.4.

4.1 Introductory Interaction Patterns

We begin by analyzing the progression paths of members across the project archives. We performed two different levels of analysis to examine the evolution of developers. At the first level, we examine how developers join individual projects and at the next level we take the pool of developer contributions as a whole across the entire set of six projects that we are examining. For each level, we build a pattern of the developer's contributions by identifying the first appearance of a developer's contributions in each of the three archival mediums: mails, bug tracking, and source code. The release of the first contribution in each medium is recorded and a progression path is established.

We grouped the progression paths into five major categories based on their relationship to the socialization process in the software ecosystem. In the most literal sense, we consider that a developer followed the onion model if they first contributed to project mailing lists, then in a subsequent release contributed to the project bug tracker, and in a yet later release contributed to project source code. We note that many developers may not spread these actions over three or more six-month release cycles, so we identify a similar accelerated progression. Table 4 provides a breakdown of the number of individuals in each progression path.

Table 4. Progression Across Social and Technical Mediums.

	Individual projects		Ecosystem subset	
Category	Members	Percent	Members	Percent
Social-tech	24	1.82%	25	5.45%
Accelerated	100	7.58%	82	17.86%
Tech-social	118	8.95%	103	22.44%
Technical	224	16.98%	74	16.12%
Source only	853	64.67%	175	38.13%

Social-technical path: This includes members who start in social medium (mail) and then progress to technical mediums in subsequent releases. We expanded this category from the original onion model where members progress from mailing lists to bug tracker activities and then to code commits, to also include members who started in mailing lists but then received commit access and then were found to participate in the project bug tracker. We did so, because it might be possible that members contributed to technical discussions of patches through the mailing list as opposed to relying on the project bug tracker. A key criterion for this category is that members are active in only one medium during a release period.

Accelerated path: This category includes members who start in mailing list (social medium) and then participate in either technical medium (e.g., bug tracker or code), but have multiple kinds of contribution during the same release. For example, we found individuals who appear in both mailing lists and bug tracker in the same release. We also found individuals who participated in all three mediums in the initial release. We combined all paths that involved members who first started with a social process and then moved to technical contributions into one group, since it is possible that our analysis at the release stage might miss members who follow the traditional model, but where each stage lasts for weeks.

Technical-social path: This path is contrary to what has been proposed in the onion model. We found members who started by participating in bug tracker or project source code repositories and then moving to mailing list participation. While in total this path contributes a relatively small percentage, the interesting fact is that these members participated in the social medium only after at least one release of participating in the technical medium.

Technical only path: This category includes members who have participated only in technical medium. Table 4 further subdivides this category into members who had only contributed to project source code repositories and members who had contributed source code and participated in the bug tracker in any order (code contribution followed by bug tracker activity, or vice versa). This directly contradicts the onion model and shows that members in a project ecosystem can start by directly contributing to code without prior socialization.

Our analysis shows that very few project members follow the socialization process as predicted by the onion model, even when we combine the "social" and "accelerated" categories (9.44% when we combine the "Social-Tech" and the "Accelerated" patterns). We only found a small percentage of users (8.95%) participating in social medium after making technical contributions. Our largest group consisted of users who directly contributed to tech-

nical medium (81.65% when we combine the "Technical" and the "Source only" patterns). We therefore conclude that there is little support for our first hypothesis that newcomers to project communities begin by participating in the least technically challenging medium, before moving to more technical mediums.

Next, we wanted to test whether a reason for the high percentage of users directly contributing to a project could arise because these members have experienced the socialization process in another project within the ecosystem. We tracked user contributions and their progression across all the six projects (see Table 4). By doing so, we see a near tripling of people who follow some portion of the onion model (individual projects: 1.82+7.58=9.4), which increased to 23.3 (5.45+17.86), an increase by 2.47; while the number of developers contributing only to the technical mediums has fallen to 54.25% (combining the "Technical" and "Source only" paths). This shows that within the broader ecosystem developers tend to follow a socialization process more similar to those proposed in hypothesis 1, but still only a quarter of developers follow a variation of the pattern. Therefore, at the ecosystem level we also reject hypothesis 1.

4.2 Project Tenure and Code Centrality

After identifying the overlap of developers in projects and general paths that developers take after joining a project, we evaluated the effect tenure has on participation in a project, specifically with respect to code centrality. That is, we investigate whether the number of releases during which a participant is active in a project affects whether they make core contributions. von Krogh et al. found that members with generational knowledge (active across multiple releases) made changes that spanned multiple files, whereas new developers typically made changes that involved a smaller set of localized files [29]. Similarly, Duchenaut claimed that developers need certain social status before they can implement high impact changes [10].

To evaluate the centrality of a developer's contributions, we needed a method to score the centrality of each commit made to the project source code repository. Source code can be thought of as forming a network of different files that are related to each other. There are a variety of ways to construct such a network, for example one can use call graphs or package imports in languages such as Java, or use the concept of logical commits [14]. We chose to use the latter since it is not dependent on a particular programming language and works for projects that use multiple programming languages. Briefly, this method infers connections between two different files in the source code repository when they are committed together. For example, if a developer commits files A, B, and C to the repository at the same time and as part of the same commit, we infer that there is some common thread between files A, B, and C and create a triad in the network between those files. The more times that files are committed together, the higher the weight that is placed on the edges. As a project evolves, this slowly creates a more complete network-based view of the project history and source code.

Once a network of source code is created, it is possible to use various social network analysis metrics to generate a numeric centrality score for each file in the network at each time period, thus identifying the files that are considered to be most central to the project. Although there are a variety of different candidate metrics, many are not applicable on disconnected networks or make assumptions about the structure of disconnected networks that are not appropriate for our analysis. One metric that is robust and avoids issues with network structure while maintaining a consistent implementation is eigenvector centrality [5, 20]. Mathematically, eigenvector centrality is the first eigenvector of the adjacency matrix formulation of the network. In general terms the interpretation of eigenvector centrality is such that nodes with high eigenvector centrality tend to be connected to many other nodes with high eigenvector centrality, while nodes with low eigenvector centrality tend to have few connections that are primarily to other low scoring nodes. When we refer to the centrality of a file at a particular release we refer to the eigenvector centrality of that file based on the network of logical commits generated from all commits up to and including that release cycle. In this way we preserve relationships from the past while building the network for future changes. For the purposes of this work, we consider only source code files contained in the project source code repository and exclude other files such as those that support translation and documentation.

Since we are interested in the centrality of a developer's contribution, the file level centralities need to be translated into developer centrality. This requires attributing the centrality score of the file to the developer who committed it. However, note that commits made by developers often touch multiple files and developers typically make numerous commits during each community release cycle. Therefore, for an individual developer's commit we define the centrality as the mean of the eigenvector centrality of each of the files that comprises the commit. From this we generate an overall source code centrality score for each developer, which is the sum of the centralities for each of the commits. In our calculations, a developer can become prominent in a project either through making many commits to files with low or medium centralities, or by making many fewer commits to files with high centrality scores, both of which are valuable measures about the importance of developers' contributions.

In addition to the centrality of files and developers, we collect other pieces of information for each developer in the community to assist us in understanding how a developer progresses within the community. Unless otherwise specified, these metrics are collected for each developer on each project on which they worked during each time period in which they were active.

- Source Code Commits: Total number of commits containing source code, documentation, and translation.
- Mail Count: Number of messages posted to project mailing lists and the number of responses obtained from those messages.
- Tracker Activity: Number of comments created in project bug tracker and total number of actions in the bug tracker. These discussions are often technical in nature and focus on a specific defect or feature.
- Project Experience: Number of releases since the developer's first activity on the project.
- Project Active Experience: Total number of releases in which the developer was active on the project. This is Project Experience minus the number of releases for which the developer made no contributions to the project.

In the process of building a regression model it is necessary to evaluate predictor variables for independence from one another, and also whether or not there is an undue reliance on the dependent variable. We examined the correlation of the various variables to the outcome metric, Source Code Centrality, and the other collected variables. We found three variables that had sufficient independence for use as control variables in a regression model:

Mail Count, Tracker Activity and Project Experience. We could not use Source Code Commits in the regression because the dependent variable, Source Code Centrality, is a construct that relies on a multiplicative transform of Source Code Commits and therefore had a very high correlation. Project active experience could not be used because it had a very high correlation, approximately 0.96, with Project Experience. We selected Project Experience as our variable of choice as it is the best depiction of the fact that tenure accrues over the entire time a member is involved in a project.

Instead of performing a simple regression with a single intercept, we acknowledge that there is significant variance between the projects in the ecosystem and instead create a regression model with multiple intercepts, one for each project. The output of this regression model can be seen in Table 5.

Table 5: Base Regression Model Illustrating Relationship Between Project Experience and Source Code Centrality

Variable	Estimate	Significance
Mail Count	0.0675	< .0001
Tracker Activity	0.0100	<.0001
Project Experience	-0.0522	0.0253
Adj R-squared: 0.4777, F: 311.4 on 9 and 3046 DF, p < .0001		

In this basic model, we find that as expected social activities such as posting messages to project mailing lists and combination of social and technical activities such as being active on a project bug tracker increase the Source Code centrality for a developer, as shown by the positive sign on the estimates from the regression model. This indicates a relationship between general project activity and Source Code centrality, but we also find that the number of releases that a developer has been active on the project, which has a negative sign on the regression estimate, decreases the overall Source Code centrality for the developer. In essence, while controlling for other social and socio-technical activities, the longer a developer remains active on a project, the lower the expected level of Source Code centrality. This may appear in contrast to other findings that show that tenure increases importance in a project [1, 3], but we note that we are examining Source Code centrality, and not an overall metric for importance in the project. A developer with significant experience on a project may have shifted to a different role that involves more leadership and less actual development.

Our analysis shows that our second hypothesis: "*as developers gain more technical experience in a project they will contribute more to the core of project source code"* is not supported.

To gain further insight into the lack of effect of a developer's tenure in a project we performed additional analyses. Additional explorations utilizing mathematical manipulations of Project Experience provided little insight, however, a simple binning algorithm that grouped developers into three categories, New (first release), Normal (second through fifth release), and Experienced (sixth or more release), provided additional insight when used in a regression model as shown in Table 6.

We once again find that the centrality of commits to the project source code repository decreases the longer a developer has been contributing to a project (as seen by the low estimate and non-significance in results for Experienced Developers). However, there is only a slight difference between new developers and developers who have been active for between 2-5 total releases (estimates of 0.5917 and 0.5489 with significant values). This suggests that centrality for developers slightly increase for Normal Developers and then reduces for Experienced Developers.

Table 6: Enhanced Model Illustrating Relationship Between Project Experience and Source Code Centrality

Variable	Estimate	Significance
Mail Count	0.0653	< .0001
Activity Count	0.0099	<.0001
New Developer	0.5917	<.0001
Normal Developer	0.5489	<.0001
Experienced Developer	0.2327	0.2315
Adj R-squared: 0.4773, F: 279.9 on 10 and 3045 DF, p < .0001		

It is unclear what happens to the fate of experienced developers. A random examination of six developers who were active for six or more releases on a single project showed that three of the developers remained central to the project in core leadership roles, two of the developers made many fewer contributions to project source code because they had adopted broader leadership roles in the community (e.g. release manager and member of the foundation board), and the sixth developer stopped contributing to developer source code entirely, instead focusing on bugs and shepherding the process of evaluating new feature requests for the project.

Additional explorations with different combinations and numbers of bins for developer experience yielded no additional insight. These results suggest that experienced developers can follow several different paths, some remain active code contributors while others may transition into more managerial or leadership roles focusing instead on managing the community. Because of this dispersion of roles we do not see a net effect of tenure in a project on developers' code centrality and, therefore, reject hypothesis 2.

4.3 Ecosystem Tenure and Code Centrality

In the next part of our work, we evaluated whether overall tenure in the ecosystem had an impact on developers' code centrality. If it is true that there is substantial transferrable knowledge between different projects in the ecosystem then developers with experience on other projects in the ecosystem should achieve higher levels of Source Code centrality at an accelerated rate.

To assist in this model, in addition to the metrics collected in the previous section we also collect the following metrics:

- Prior Experience: Number of releases the developer was active on other projects before their first contribution to the project in question (calculated once per developer per project).
- Total Experience: Number of releases since the developer was first active in the ecosystem.
- Total Active Experience: Number of previous releases for which the developer made contributions in the ecosystem. Functionally, this is the Total Experience minus the number of releases for which the developer made no contributions to projects in the ecosystem.

For a developer who is completely new to the entire ecosystem, their prior experience will be '0' and their total experience and total active experience will be the same as in the previous section.

In most cases, we found that developers were active without any major breaks in participation. The correlation between Total Experience and Total Active Experience was 0.97, preventing their mutual use in a single regression model. The correlation between Total Experience and Project Experience was 0.78, indicating that in most cases it is inappropriate to use both variables in a regres-

sion model. We select Prior Experience as the more relevant additional metric for depicting experience in the ecosystem.

Thus, building on the models in the previous section we add the additional variable of Prior Experience to evaluate to what degree a developer's experience on projects in the ecosystem prior to joining this project affects the centrality of their contributions. The results of this regression are shown in Table 7 and indicate that a developer's Prior Experience in the ecosystem does not play a role in the centrality of source code commits as evidenced by the negative estimate and non-significance of the result. Therefore, we reject hypothesis 3.

Table 7: Regression Model Relating Project Experience, Ecosystem Experience, and Source Code Centrality

Variable	Estimate	Significance
Mail Count	0.0675	< .0001
Activity Count	0.0100	<.0001
Project Experience	-0.0533	0.0232
Prior Experience	-0.0138	0.6809
Adj R-squared: 0.4775, F: 280.2 on 10 and 3045 DF, p < .0001		

4.4 Broad Factors that Affect Code Centrality

As we found that prior experience was not indicative of code centrality, we examined the data to find other factors that may be related to increased developer centrality. Further, a major problem that we faced in the analysis of this community is the fact that many of the variables related to developer participation that can be collected from the archives were highly correlated, making their use inappropriate in a regression model. However, it is possible to gain additional insight into which variables have the greatest effect on our dependent variable by performing a principal component analysis and using those components as predictors for Source Code Centrality. This also allows for broader generalization of attributes that lead to an increase in Source Code centrality. Table 8 (column 1) lists the variables that we collected from the archives: C, C#, Java code commits, documentation related commits, translation commits, total number of mail messages, number of messages started by a developer, number responses to the original message by a developer, bug tracker activity (e.g. bugs opened, closed, statuses changed), comments left in the bug tracker, and the five experience variables previously discussed.

There were four components that had a standard deviation of greater than one and ten factors with a standard deviation of less than one. In keeping with convention, we selected the four components with greatest explanatory power for further analysis. The interpretations of the factors, as presented in Table 8 are as follows:

Table 8: Major Component Loadings from Principal Component Analysis

Component	1	2	3	4
Std Dev	2.3286	1.8840	1.1644	1.0550
Source Commits	-0.371			0.254
Doc Commits	-0.386			0.275
Trans Commits	-0.145		-0.215	0.492
Message Count	-0.388			-0.159
Message Started	-0.343		0.194	-0.442
Message Responses	-0.341		0.187	-0.445
Bug Tracker Activity	-0.367			0.232
Bug Tracker Comments	-0.388			
Prior Experience		-0.146	0.773	0.311
Total Experience		-0.502	0.229	
Total Active Experience		-0.476	-0.300	-0.138
Project Experience		-0.509	0.147	
Project Active Experience		-0.467	-0.310	-0.126

Component 1: The inverse of artifacts created by the developer, which translates to the inverse of total participation. Developers who score very low on this component have created numerous artifacts in any of the archival mediums.

Component 2: The inverse of the time that a developer has spent on the project. New developers will score higher on this component than developers who have been active in the community or project for a significant amount of time.

Component 3: Broad social experience. Developers who score high on this component have been active on mailing lists and have been in the ecosystem for a long time. In contrast, developers who score lower are newer to the ecosystem and project. Note that translators can fall in the latter group since translation is often a more solitary role within the community and translators frequently pop in and out of projects.

Component 4: Technical medium expertise. Developers scoring high in the component have been active in both the bug tracker and Source Code repositories, but less active on project mailing lists. They typically have significant amounts of experience in the ecosystem, but may be new to a project. An expert translator who enters a project to improve its support for internationalization would score high on this component.

As before, we create a regression model to evaluate the relationship between these generated components and Source Code Centrality. We again allow the intercept for each project to vary to account for differences in code structure in the ecosystem. The results can be seen in Table 9.

Table 9: Regression Model Illustrating Relationship Between Generated Components and Source Code Centrality

Variable	Estimate	Significance
Component 1	-1.0341	< .0001
Component 2	0.1848	<.0001
Component 3	-0.1230	0.0001
Component 4	0.6600	<.0001
Adj R-squared: 0.6268, F: 514.1 on 10 and 3045 DF, p < .0001		

Unsurprisingly we find that developers who create the most artifacts, as shown by a low score in Component 1, typically have the highest Source Code Centrality (negative regression estimate and highly significant). This is to be expected from our definition of Source Code centrality.

We also find that individuals with more experience in the project, as shown by a lower score in component 2 will have lower overall Source Code centrality (on account of the component loadings being the inverse of overall experience and the positive sign on the estimate in the regression model). This reinforces our findings from earlier in section 4.2 and 4.3 that experience in the community leads to a lower level of Source Code Centrality.

Component 3 shows that individuals with broad social experience typically have lower amounts of Source Code Centrality. Remember, developers with high social experience rate high in the component loading and the component has a negative estimate in our regression, which leads to the above conclusion. This could be as a result of a distribution of work within the projects (e.g. some people write code, some people manage aspects of the project on the mailing lists). The reverse interpretation is that developers with little experience in the ecosystem and few social contribu-

tions on mail messages may be more likely to contribute centralized source code.

Component 4 indicates that developers with expertise in the technical mediums, particularly those involved in translation and internationalization, and those who also have prior experience in the ecosystem but may be new to the project have the ability to achieve high levels of Source Code centrality. In this narrow context, we see some of the only evidence that prior experience in the project ecosystem may allow an individual to sidestep some of the standard learning processes and begin to commit directly to core elements of the project.

Therefore, we see that while hypothesis 3 can generally be rejected across the ecosystem, there are some small niches where it may hold true. In particular, this appears to hold true for the specialized case of translators migrating across projects in the ecosystem, as evidenced by component 4.

5. DISCUSSION

From our analysis of developers in the selected projects we found significant overlap in the population of developers between projects. This held true not only for developers who wrote the code for the projects, but also individuals who documented, translated, created other media, and even the engaged users who were only active on project mailing lists. This established the strong possibility that developers would have a shared body of transferrable knowledge that they can take from one project to another and answers research question 1 by showing that developers do move between and participate in multiple projects in an ecosystem, often times in significant numbers.

Surprisingly, however, when we analyzed the progression paths of individual developers, both within individual projects and across the projects in the ecosystem, we found many individuals who eschewed social communication mediums and focused only on the technical mediums, in strong contrast to what was predicted by the onion model of Open Source participation. At the individual project level fewer than 1 in 10 developers follows the onion model predicted pattern of moving from social to socio-technical to technical mediums. When we broadened our analysis to include the entire ecosystem we found slightly stronger signs of the onion model as 23.31% of the members followed the onion model suggested progression from social to technical mediums.

In contrast to the onion model, a majority of participants were found to have participated only in technical medium (81.65% per project and 54.25% when looking at all six projects). This is contrary to most current studies that have analyzed the development process in Open Source projects, albeit these were conducted on single, standalone projects. One explanation for these results could be that developers who jump right in with code contributions might have undergone a socialization process in another project in the ecosystem that helped them jumpstart their contribution; or there might be other social communication medium such as direct person-person email or real-time chat (IRC) that were used by these technical contributors for which we do not have archival data. The lack of support for the onion model and for our first hypothesis suggests that the joining script for developers, especially in the new category of OSS 2.0 projects, is still a fertile field of research. Particularly, the interaction of additional communication mediums, such as blogs, Twitter, real team chat, social software development sites such as GitHub[1] and BitBucket[2], and social networking sites in combination with the increased participation of commercial firms means that Open Source has evolved significantly in the last ten years and that previous models of social participation may need to be updated for the current state of the art in Open Source software development.

The next step was to analyze whether prior experience, either on a single project or in the broader ecosystem had any effect on the contribution of developers. As opposed to simply summing up the number of commits that an individual made, we calculated the centrality of each developer's contribution as a proxy for the importance or depth of contribution. In this way, an individual who changes files that are considered to be central to the project will score higher than a developer who is highly active but focuses all of their work on a peripheral component such as a plugin.

In contrast to our expectations and hypothesis 2 and 3, we found that experience in communities, both in individual projects and the ecosystem as a whole, play little role in the centrality of contributions to project source code. In fact, new developers were slightly more likely to have central contributions than experienced developers and those developers with extensive experience were found to have widely varying levels of centrality. There are many possible reasons for this phenomenon. First, it is possible that experienced developers, such as those who have been active for six or more releases, are venturing into project management and leadership roles, which require more time focusing on management or architecture than actual development. While it is likely that these developers have a deep and robust understanding of the project source code and architecture, it may be that their skills and experience are better utilized in managing and shepherding other users; a finding that was partially echoed by a random examination of experienced developers.

A possible cause for the lack of effect of prior experience across projects in the ecosystem could be as a result of the structure of the projects themselves. The bulk of the code for most of the projects was written in C, a language that isn't always as amenable to modular code structures as managed languages such as Java and Python. If developers migrate across projects in the ecosystem and bring with them a feature gift to the new project [29], it is possible that the architecture of the new project requires substantial modifications to core elements of the project, therefore increasing the centrality of the new developer's contributions by virtue of the number of modifications made and the location of those modifications.

When we attempted to identify additional factors that played a role in source code centrality as per research question 3, we found one group of developers who appeared to benefit from learning in the ecosystem: translators. A major influencing factor of this is likely the very standard method for internationalization across all projects. A translator can introduce internationalization features to a project with modifications to only a handful of header files and by modifying references to strings in the project in a fairly mechanical manner, something that is easily reproducible across projects. The actions of translators and other individuals who do not write code are ripe for future research.

In summary, the major implications of our study relate to the process of managing a software development team. For individual projects, we found little evidence for the commonly discussed "onion" model of participation that involves individuals migrating from the edges of a project to the core through a gradual socializa-

[1] https://github.com/

2 https://bitbucket.org/

tion process, although there was slightly more support in connected ecosystems. This suggests that open source projects should re-evaluate the process used for socializing new developers and eventually awarding committer status. For software engineering researchers, this study provides a foundation for understanding participation in complex systems using a multi-modal approach, for example, using source code, email, bug trackers, and other social media to understand participation.

Finally, our study provides an impetus for re-evaluating the commonly accepted onion model, which may be overly limiting. Complex ecosystems involve developers moving across multiple projects. A well-designed architecture can facilitate this flow of individuals and knowledge. Further, because of this migration and a subsequent lack of generational knowledge, further investigations into technology and methods that help in externalizing the semantic knowledge of experienced developers is needed.

6. THREATS TO VALIDITY

As our findings are in contrast to many previous studies of individual Open Source projects, we must be conscientious of the limits of our research and the possible threats to the internal, construct, and external validity of our work.

Internal validity: Our analysis is based on data extracted from public project archives – mailing lists, bug trackers, and source code repositories. We have largely assumed that social interactions and decision making are conducted via mailing lists and we could have missed communication that occurred through personal email, IRC channels or face-to-face meetings during conferences and developer meetings. However, research literature suggests that Open Source socialization processes largely occur through project mailing lists, so in this respect we are in line with previous literature but we realize that this might not be sufficient to understand the communication in modern open source projects [3, 24]. Further Open Source norms and traditions encourage project discussions and communications to take place in the developer mailing lists [13]. Next, our treatment of comments in the bug tracker as a technical activity may be problematic. While most discussions are about a particular issue or defect, there are situations where users post non-technical information to the bug tracker, such as feature requests, however these requests rarely come from experienced project developers who have commit access and are a minority of discussions.

Construct validity: There are three major construct validity threats to our study. First, the use of a release as a unit of analysis for our regressions and analysis of interaction patterns, which was done to provide a consistent way of comparing across projects as all projects followed the same 6 month cycle, may be problematic if developers move through the onion model phases in shorter time periods than the 6-month release cycles in GNOME. We accommodated this concern where feasible, for example, when calculating the progression paths in section 5.1 we optimistically considered individuals to have followed a "social-technical" path even when their first contributions to social and technical mediums were in the same release. The second threat to construct validity lies in the creation and use of Source Code Centrality as a proxy for core contributions. Past research has often used the raw amount of contribution (e.g., number of commits or lines of code changed) as a measure of developer activity [26, 29] or mailing list communications as a measure of centrality in a project network [2, 16]. In our study, we needed a finer grained measure to characterize both the volume and importance of contributions, which allowed us to identify whether prior experience enabled developers to start making complicated, central changes earlier. Logical commits were used to create the network as they provide a language agnostic method to associate source code files and eigenvector centrality was used because of its general robustness in the face of various network topologies. Third, our analysis centers on code contributions in a project and, therefore, ignores contributions of other stakeholders in a project (e.g., architects, testers) and their socialization processes. However, note that past literature that has demonstrated the existence of the onion model investigated the socialization process of developers based on their code contributions and is similar to our study.

External validity: Finally, there is a chance that GNOME may not be a suitable sample of an Open Source ecosystem, limiting the degree to which we can generalize our findings to other Open Source ecosystems. Further, our study considered a subset of six projects with significant history and participation as representative of the greater GNOME ecosystem. The fact that these six projects are well adopted within the community, have long histories, and significant participation may make them outliers in the broader context of the GNOME ecosystem. It is possible that other projects might demonstrate different socialization processes.

7. CONCLUSIONS

Within a modern OSS 2.0 project ecosystem our results are substantially in contrast to earlier research conducted on individual projects when the state of Open Source software development had not yet evolved to its current state. In particular, this work is novel because to the best of the authors' knowledge it is the first study to consider these contributions in the context of an ecosystem. We found little support for the traditional onion model within a single project, although it was supported slightly more often when contributions across multiple projects in the ecosystem were considered. Our results show that prior experience in the project or the ecosystem does not seem to have a high effect on the overall centrality of a developer's contribution. Further, we found that tenure in a project or even the ecosystem did not have a high impact on the centrality of one's code contributions when measured using our metric that takes into account both number of contributions and the eigenvector centrality. Our findings provide interesting insights and contradict existing assumptions of the onion model; these findings merit further explorations into project ecosystems.

In future work, we intend to further investigate these relationships in modern OSS 2.0 communities. In particular the adoption of decentralized version control systems such as git and collaborative development sites like GitHub and BitBucket has fundamentally rewritten the socialization process by allowing anyone to fork project code and begin working on a project without the need for formal designation as a committer. The degree to which these sites support code annotation and discussion may both decrease the number of people following the onion model and also assist new developers in understanding complex technical aspects of project source code. While this makes the development process more accessible, it also has the potential to make contributions more difficult by requiring developers to request that their code be pulled into the main branch for each set of patches, rather than receiving committer status once for all time.

Open Source has changed significantly over the past decade. What we know about the socialization process in Open Source projects may not be true in the ecosystems of Open Source 2.0. The evolution of tools, project ecosystems, and the increasing ability of developers to easily move between different projects create a variety of social and technical effects that require more study.

8. ACKNOWLEDGMENTS

This research is supported by grants NSF IIS-0414698, NSF CCF-1016134, AFSOR FA9550-09-1-0129, and an NSF Graduate Research Fellowship to the third author. The authors also thank James Herbsleb for his assistance in structuring the original research questions.

9. REFERENCES

[1] C. Bird, A. Gourley, P. Devanbu, M. Gertz, and A. Swaminathan. Mining email social Networks. In *Third International Workshop on Mining Software Repositories*, pages 137-143. IEEE, 2006.

[2] C. Bird, A. Gourley, P. Devanbu, A. Swaminathan, and G. Hsu. Open borders? Immigration in open source projects. In *Fourth International Workshop on Mining Software Repositories*. IEEE, 2007.

[3] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu. Latent social structure in open source projects. In *16th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 24-35. ACM, 2008.

[4] C. Boldyreff, K. Crowston, B. Lundell, and A. Wasserman. Open source ecosystems: diverse communities interacting. In *5th International Conference on Open Source Systems*. Springer, 2009.

[5] P. Bonacich. Power and centrality: a family of measures. *American Journal of Sociology*, 92:1170-1182. March 1987.

[6] K. Crowston and J. Howison. The social structure of open source software development teams. *First Monday*, 10(2), February 2005.

[7] K. Crowston, K. Wei, J. Howison, and A. Wiggins. Free/libre open source software development: what we know and what we do not know. *ACM Computing Surveys*, 44(2), 2012 (Forthcoming).

[8] B. Dagenais, H. Ossher, R. Bellamy, M. Robillard, and J. de Vries. Moving into a new software project landscape. In *32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, pages 275-284, ACM, May 2010.

[9] J. Des Rivieres and J. Wiegand. Eclipse: A platform for integrating development tools. *IBM Systems Journal*, 43(2):371-383, 2004.

[10] N. Ducheneaut. Socialization in an open source software community: a socio-technical analysis. *Computer Supported Cooperative Work*, 14(4):323-368, 2005.

[11] R. Farzan, L. Dabbish, R. Kraut, and T. Postmes. Increasing commitment to online communities via building social attachment. In *ACM 2011 Conference on Computer Supported Cooperative Work*, 321-330, ACM, March 2011.

[12] B. Fitzgerald. The transformation of open source software. *MIS Quarterly*, 30(3):587-598, September 2006.

[13] K.Fogel. *Producing open source software: how to run a successful free software project*, O'Reilly Media, Sebastapol, California, 2005.

[14] H. Gall, K. Hajek, and M. Jazayeri. Detection of logical coupling based on product release history. In *14th IEEE International Conference on Software Maintenance*, pages 190-198, IEEE Press, March 1998.

[15] D. German. The evolution of the GNOME project. In *2nd International Workshop on Open Source Software*, 2002.

[16] D. German. The GNOME project: a case study of open source, global software development. *Software Process: Improvement and Practice*, 8(4):201-215, 2003.

[17] git. the fast version control system: http://git-scm.com/

[18] G. Hertel, S. Niedner, S. Hermann. Motivation of software developers in open source projects: an internet-based survey of contributors to the Linux kernel. *Research Policy*, 23(7):1159-1177, July 2003.

[19] C. Jensen and W. Scacchi. Role migration and advancement processes in OSSD projects: a comparative case study. In *International Conference on Software Engineering*, pages 364-374. IEEE, 2007.

[20] L. Katz, A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39-43, March 1953.

[21] S. Koch and G. Schneider. Effort, cooperation and coordination in an open source software project: GNOME. *Information Systems Journal*, 12(1):27-42, January 2002.

[22] K. Lakhani and E. Von Hippel. How open source software works: "free" user-to-user assistance. *Research Policy*, 32(6):923-943, June 2003.

[23] J. Lerner and J. Tirole. Some simple economics of open source. *Journal of Industrial Economics*, 50(2):197-234, June 2002.

[24] A. Meneely, L. Wililams, W. Snipes, and J. Osborne. Predicting failures with developer networks and social network analysis. In *16th ACM SIGSOFT International Symposium on the Foundations of Software Engineering*, pages 12-23. ACM, 2008.

[25] A. Mockus, R. Fielding, and J. Herbsleb. A case study of open source software development: the Apache server. In *22nd International Conference on Software Engineering*, pages 263-272. ACM, 2000.

[26] A. Mockus, R. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions Software Engineering Methodology,* 11(3):309-346, July 2002.

[27] W. Oh and S. Jeon. Membership Herding and Network Stability in the Open Source Community: The Ising Perspective. *Management Science*, 53(7):1068-1101, July 2007.

[28] W. Scacchi. Free/open source software development: recent research results and emerging opportunities. In *6th Joint Meeting on European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 459-468. ACM, September 2007.

[29] G. von Krogh, S. Spaeth, and K. Lakhani. Community, joining, and specialization in open source software innovation: a case study. *Research Policy*, 32(7):1217-1241, July 2003.

[30] A. Wolfe. Eclipse: A platform becomes an open-source woodstock. *ACM Queue*,1(8):14-16, 2003.

[31] B. Xu and D. R. Jones. Volunteers' participation in open source software development: a study from the social-relational perspective. *ACM SIGMIS Database*, 41(3):69-84, August, 2010.

[32] Y. Ye and K. Kishida. Toward an understanding of the motivation of open source software developers. In *25th International Conference on Software Engineering*, pages 419- 429, IEEE Computer Society, May 2003.