Generative Diffusion of Innovations and Knowledge Networks in Open Source Projects¹

Research-in-Progress

Zhewei Zhang  
Temple University  
Philadelphia, PA  
zhang@temple.edu

Youngjin Yoo  
Temple University  
Philadelphia, PA  
yxy23@temple.edu

Sunil Wattal  
Temple University  
Philadelphia, PA  
swattal@temple.edu

Bin Zhang  
University of Arizona  
Tucson, Arizona  
binzhang@arizona.edu

Rob Kulathinal  
Temple University  
Philadelphia, PA  
robkulathinal@temple.edu

Abstract

In this study, we conceptualize an open source community as a constellation of dynamic, multiple and heterogeneous informal knowledge networks. Generative artifact, open source software in our context, is embedded in multiple knowledge networks. Therefore, generative diffusion of innovation will be affected by these networks. We identify two types of knowledge networks, developer-project network and developer-developer network, which a developer can simultaneously belongs to. We argue that the two networks emphasized on different types of knowledge transfer, and have different effects on the generative diffusion of innovation. We propose our hypotheses and methodology to test our hypotheses. We also discuss future plan and expected contributions.

Keywords:  Diffusion of Innovation, Open Source Software, Perspective Making, Perspective Taking, Knowledge Network

Introduction

Open source projects represent a new form of organizing for knowledge creation through global digital infrastructure (Faraj et al. 2011). In such forms of organizing, individuals who do not necessarily belong to the same formal organization can temporarily work together through digital technology. Among many

¹ This paper is based on work supported, in part, by the National Science Foundation under grants VOSS-093010 and VOSS-1120966. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
benefits of such forms of organizing, one important benefit that has not been explored systematically is its generativity. Artifacts (i.e., software codes) developed in open source communities do not remain as they are created. Many open source codes become the basis of new projects, while other open source codes are adopted to solve unique local problems. In other words, once created, these artifacts in an open source community mutate as they are being diffused throughout the community. We refer this new type of diffusion as generative diffusion of innovation, which should be seen as a multi-dimension construct to capture the ever-evolving nature of generative digital artifact been diffused.

In recent years, social networking has been considered as one important feature by many kinds of web sites, even sites for software project hosting, such as Github. “Social coding” emphasizes formal and informal collaboration among developers through such social networking functionality provided by those website. Though few studies, for example Dabbish et al. (2012), have touched on the emerge of social coding, however, we still lack a deep understanding on how such new forms of collaboration may actually impact the long term development of open source software project. Does project really benefit from the social connections among developers? In our paper, we conceptualize an open source community as a constellation of dynamic, multiple and heterogeneous informal knowledge networks (Hansen 2002). It attracts developers with different interests and skill sets who work on many different projects. Therefore, the generativity of open source projects comes from broadly two different sources. First, each open source project consists of multiple individuals who share the same interests, but have potentially different skill sets. These individuals learn from one another as they try to work on the same project. Second, most developers work on more than one project. Therefore, developers can serve as a conduit of ideas from one project to another. Through this, developers might pick up an idea from different projects in a different domain and utilize it in the current project. These two types of relationships form the basis of two different types of knowledge networks in an open source community. We contend that these two types of knowledge networks form the basis of generative diffusion of innovation through “wakes of innovations” in an open source community (Boland et al. 2007).

We conduct this research in the context of online open source software repositories. These repositories, such as GitHub and SourceForge, allow developers to create and upload source code, and also allow other developers to modify code, or adapt code created by others for their own use. Further, these websites also contain social networking features which enable developers to connect with any developer who is registered on the same site. Therefore, such open source repositories provide an ideal context to study the impact of two different knowledge networks on generative diffusion of innovations.

Theoretical Background

Conceptualizing Generative Diffusion of Innovations

Zhang et al. (2012a) propose the idea of generative diffusion of innovations as a multi-dimensional construct. Inspired by genetics, the three proposed dimensions include proliferation, evolvability, and temporality. We measure proliferation by the number of people who adopt the same piece of software code. Evolvability concerns the degree to which an adopted innovation mutates. The more an adopted innovation mutates, the more evolvability it has. In the context of open source project development, evolvability refers to the further modification of a piece of software code after it been adopted and reused by other developers. Evolvability is measured as the extent to which the code changes compared to prior versions. Temporality refers to the fact that diffusion is not a static state but rather, it is an ever-changing process. In other words, the pattern of diffusion in terms of proliferation and evolvability may change over time. Novel and immature ideas may have different diffusion patterns from mature and stable ideas. In software development, temporality can be seen from the number of different versions of open source project that have been adopted and further developed. Taking the three dimensions together, we are able to understand the diffusion of innovation with a more comprehensive view by proposing the composite measures of diffusion: degree of generative diffusion, which is a function of proliferation and evolvability, and speed of generative diffusion, which is a function of proliferation, evolvability and temporality. Zhang et al (2012a) describe these constructs in detail, and also present an overview of how to measure these using techniques adapted from evolutionary genetics.
Past Research on Open Source Projects

Previous studies on open source projects have examined the motivations and contributions of the open source community (Lakhani and Wolf, 2003). Prior research also focuses on the collaboration in open source project development (Yamauchi et al. 2000, David and Rullani, 2008). Grewal et al. (2006) examined how the success of open source project was affected by the structural embeddedness. In their study the developers were only connected by working on the same project. Dabbish et al. (2012) investigated the value of visibility and transparency brought by social coding in Github through interviews. Taken together, past research on open source projects based on network analysis suggests that on-going interactions and social relationships among developers play an important role in shaping the way open source projects develop over time. While the past research on open source focused on what developers do and how they are interconnected, no attention was given to the evolution of technology artifacts that are being created. Literature on software code reuse among open source projects has studied the reasons for reuse (Haefliger et al 2007), patterns of reuse (Mockus 2007), as well as interactions between adoption and invention (King and Lakhani, 2011). However, existing research does not discuss how open source software codes evolve over time as they are reused, how software codes developed from one project can contribute to other projects, and how such evolution of software codes are influenced by the way developers collaborate.

The paucity in research on the evolution of technology artifacts (i.e., software codes) is unfortunate as open source communities are indeed a vibrant and dynamic engine of innovations where new artifacts are constructed and existing artifacts are constantly improved. However, not all artifacts are created and evolve equally. Some artifacts evolve faster than others; some produce more variants than others; and, some become the basis of completely new projects while others don’t. What is not known is what affects such different pattern of evolution of digital artifacts in open source communities. Do different forms of communications and collaboration affect the way these artifacts evolve over time? If so, how? We do not know the underlying social mechanisms for generative diffusion of innovations in open source communities. In this study, we are trying to uncover social mechanisms of generative diffusions of innovation in open source projects and compare their relative efficacy.

Characterizing Knowledge Networks in Open Source Projects

We conceptualize an open source community as a constellation of multiple knowledge networks. Boland and Tenkasi (1995) suggest that generative knowledge creation involves perspective taking and perspective making. Perspective taking refers to communication that strengthens the unique knowledge of a community, while perspective making refers to communication that improves its ability to take knowledge of other community into account. In the context of open source projects, perspective making takes place when developers collaborate with other developers who work on the same project. On the other hand, perspective taking takes place when developers collaborate with other developers who do not work on the same project. In this paper, we propose a network centric approach to characterize perspective making and perspective taking.

First, perspective making in open source projects requires developer-developer knowledge networks among developers on the same project. This network allows us to identify a cluster of developers who share the same interests based on their activities on the same project. Therefore, communications that take place among developers through developer-developer network on the project represents primarily perspective making.

Second, perspective taking in open source projects requires a developer-project knowledge networks. Based on developers’ activities on different projects, one can construct developer-project affiliation network (Monge and Contractor 2003). This allows us see how well a focal project is connected to another project based on developers who work on both projects. These developers can serve as a potential conduit through which an idea developed from one project can flow into another project. Through such knowledge networks, developers encounter heterogeneous knowledge that they may not find in their own project communities. Therefore, communications that take place among developers through developer-project network represent primarily perspective taking.

We further contend that these two knowledge networks represent two different mechanisms of generative diffusion of innovation. Within a developer-developer network on the same project, a developer can
Social Media and Digital Collaborations

directly learn from other developers by watching their activities within the same project. In this case, the developer can face a homogeneous knowledge pool that consists of similar set of ideas from a group of developers who work on same projects. Within a developer-project network, to the contrary, developers directly interact with other projects. Therefore, one developer’s idea is first transferred into specific artifacts (in the form of software codes) for a particular project before another developer can learn from. In this case, the developer is facing a knowledge pool that consists of ideas from many developers trying to solve a specific design challenge. Therefore, knowledge that is shared through this network is likely to be more domain-specific and thus more heterogeneous. Therefore, we argue that through a developer-developer network, artifacts evolve through mutations within the same domain, while through a developer-project network, artifacts evolve through recombination across different domains (Arthur 2009).

Hypotheses

Previous studies on network’s role on open source developments (Grewal et al. 2006) or diffusion of technology (Susarla et al. 2012) focus on the diffusion of one particular project or technology. To the contrary, what we are interested in here is the generative diffusion of innovation that covers not only the diffusion of the original innovation but also its derivative innovations. Thus, the node level analysis such as the centrality of the focal project, project manager or the content creator does not provide an appropriate level of analysis in our case. Our study tries to understand how an artifact transferred through different types of knowledge networks may affect the generative diffusion of innovation differently. Figure 1 provide the conceptual model for our study.

First, when looking at developer-to-developer network on the same project, a developer communicate and learn directly from other projects. Previous studies on knowledge transfer (McFadyen and Cannella 2004, Levin and Cross 2004) suggested that knowledge transfer is positively impacted by the relationship between two parties. In our case, a social connection between developers (follow/followed) indicates a stronger relationship between developers. Therefore, the more connection among developers, the greater their ability for perspective making. To operationalize the intensity of communication among developers, we use density as our measurement, which is a network level measurement suggested by Monge and Contractor (2003). Therefore, we hypothesize:

H1: The connection of developer-to-developer network on the same project has positive effect on the generative diffusion of innovation.

Second, when looking at the developer-project affiliation network, a developer acts as communication channel between projects and transfers artifacts from one project to other. Developers normally work on projects which more or less relevant or similar to the focal projects, such as similar design objective or similar programming language. A developer is more likely to learn specific design techniques that can be directly applied to the focal project because the motivation for this type of knowledge transfer more comes from the need to solve a relevant design challenge. Therefore, the more connection between developers and other projects, the greater their ability to make improvement on the focal project inspired from other projects. Again, we use the density of the developer-project affiliate network. Density as a property of affiliation network has been used by many previous researchers (McPherson 1982 and Breiger 1990). The density captures the most critical aspect of affiliation network structure that how well the project is connected with external projects through its member developers. The pairwise tie between developers and projects bridges two projects and facilitate transferring innovation ideas. Higher density suggests greater capacity for perspective-taking. Thus, we hypothesize:

H2: The connection of developer-project affiliation network has positive effect on the generative diffusion of innovation.

Will these two types of knowledge network affect generative diffusion of innovations in open source projects? Prior research offers two possible contrasting possibilities. Traditional organizational learning and innovation literature would suggest that teams might explore alternative ideas before they settle on a particular trajectory (March 1991; Tushman and Anderson 1986). Also, many anecdotal examples suggest that open source projects often begins through “seeding” base codes from other projects. This line of logic
suggests that in the early stage of an open source project, perspective taking through developer-project network plays more important role than perspective making through developer-developer network. On the other hand, Zhang et al (2012b) found that in open source web services projects, the evolution of web services follows a pattern that is quite different from what is known in the literature. Specifically, they found that for web services projects start their evolutions through recombination followed by mutations. Therefore, we propose a pair of competitive hypotheses:

**H3a:** A developer-project network will have a stronger effect at the early stage of an open source project than a developer-to-developer network, while a weaker effect at the later stage.

**H3b:** A developer-to-developer network will have a stronger effect at the early stage of an open source project than a developer-project network, while a weaker effect at the later stage.

---

**Study Context and Data**

We collect the data for this study from Github which is the largest online hosting repositories of open source projects (Finley 2011) and contains more than 3 million developers and 6 million repositories. It is built on Git that is a widely used distributed version control and source code management system. It allows developers to track every single modification of their software code, providing a perfect research context to our understanding of how innovations evolve over time as diffuse among developers. In addition to the conventional repository service, Github also offers social networking features that help developers to connect with other developers who are registered with GitHub. In our study, we rely on Github Archive website to create our data set. Github Archive is a website that automatically store every event, such as fork, push, commit and follow, on Github since Feb 12th, 2011

**Data Collection and Description**

In this study, we focused on jQuery plugin projects on Github. JQuery plugin is light plugins created to extend JQuery’s functionality using JavaScript. Given the differences among all kinds of programming languages, it is extremely difficult to compare projects written in different languages. Focusing on projects written in same language allows us to make comparisons while controlling other language-associated confounding factors. The focus on jQuery plugins allow us to only focus on JavaScript language only. We choose JavaScript because it is the most popular programming language used on Github, representing 21% of projects hosted on Github. Furthermore, relatively simple code structure of jQuery allows us to calculate the measurement for generative diffusion and compare them among different projects.
Currently, there are 21,315 jQuery plugin repositories on Github. We collected data from projects that match the following criteria. First, the project has to be created between Feb 12 2011 and Feb 12 2013. Feb 12 2011 the earliest time that the data is available on Github Archive, while Feb 12 2013 allows us to have at least one year of data for giving project. Second, we only include projects that have at least 50 forks so that we can have enough nodes and edges to create meaningful network, at early stage of a project.

In addition to the network data, we are also collecting meta data about projects and developers such as age of project, whether the project is sponsored by a company, and developer demographic information such as developer tenure with GitHub and his/her level of engagement. To operationalize the generative diffusion, we adopt the measurement proposed in Zhang et al. (2012) by collecting the source codes of each forks of the focal projects.

Network Definition

First, we need to define developer in our networks. To do so, we use a feature called fork in GitHub. There are two types of relational actions that a developer can take on a project: fork and watch. Fork is an action by which a developer copies an existing software code and creates a separate repository to start an independent development. Watch, instead, is similar to adding the project into developer’s watch list so it can be easily found later. After forking a project, the developer can either work completely independently from the original project or initiate a push back to original software code, requesting any modification he made to be adopted by the original developers. Therefore, fork is a direct evidence of a developer’s participation in a project, a much stronger relationship than watch as it represents a much higher possibility that developers may actually work on the project. Hence, we define a focal project’s developer as those who actually forked the focal project.

Then, to build developer-project network, we first find other projects that a developer also works on besides the focal project. Similarly, we use forking to identify a developer-project network for the focal project. As we described earlier, this affiliation network primarily concerns perspective taking in a knowledge creation process, therefore we only include jQuery-alike projects, which can provide direct, tangible knowledge on jQuery plugin development, in our network. Furthermore, we exclude developers who do not have other forked projects from our network. So we can remove those isolated nodes and focus on networking as knowledge communication channel of developer-project relationship. Therefore, at given time point, for this type of network, the set of developer contains all developers that forked our focal project and at least one other jQuery-alike project. The set of project then contains jQuery-alike projects forked by these developers. Figure 2 (top) shows an example of project-affiliation network of jQuery-Plugin-Validation project. Black dot represents developers working on the focal project while orange dot represents projects connected by these developers.

To build developer-develop network, we use a feature called follow. In addition to allow developers work on different project, Github also provides social networking features similar to Twitter that allows a
developer to follow and be followed by other developers. The public activities such as committing change, following other developer or watching a project can be viewed by other developers. Using this data, we collect how one member developer is related with another member. Similarly, as developer-project network, we only include developers with at least one follow/followed relationship with another developer working on the same project. Figure 2 (bottom) shows an example developer-developer network. Among all the developers working on the same project, only a small portion of developers actually have this type of social relationship, which is not surprising in OSS development as most often only core members have strong social tie with each other.

Figure 3 Developer-Developer Network of jQuery-Plugin-Validation

### Preliminary Result

So far, we have finished network data collection on 78 projects that match our criteria described earlier. This number is much smaller than 21315 we mentioned earlier. The first criteria of being created between Feb. 12 2011 and Feb. 12 2013 reduced the number to 11847. Then having at least 50 forks further reduced the number to 78. Table 1 below summarizes statistics of key properties of developer-project affiliation network. For number of developers, we see a 2-week and 1-month data is quite small at 2.06 and 2.85. This may also be contributed by the strong power law in OOS development that only small portion of projects can actually get enough attention from other developers.

<table>
<thead>
<tr>
<th></th>
<th>2 weeks</th>
<th>1 month</th>
<th>3 month</th>
<th>6 month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of</td>
<td>Average:2.06</td>
<td>Average: 2.85</td>
<td>Average:9.56</td>
<td>Average:18.13</td>
</tr>
<tr>
<td>developers</td>
<td>StD: 8.27</td>
<td>StD: 10.12</td>
<td>StD: 18.86</td>
<td>StD: 27.49</td>
</tr>
<tr>
<td>Density</td>
<td>Average:0.15</td>
<td>Average: 0.18</td>
<td>Average: 0.15</td>
<td>Average: 0.16</td>
</tr>
<tr>
<td></td>
<td>StD:0.32</td>
<td>StD:0.34</td>
<td>StD:0.26</td>
<td>StD:0.25</td>
</tr>
</tbody>
</table>

Table 1 Statistics of Developer-Project Network

Table 2 below summarizes statistics of key properties of developer-developer network. The numbers are much larger in general than those for developer-project affiliation network.

<table>
<thead>
<tr>
<th></th>
<th>2 weeks</th>
<th>1 month</th>
<th>3 month</th>
<th>6 month</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of</td>
<td>Average:22.55</td>
<td>Average: 24.76</td>
<td>Average:33.69</td>
<td>Average:53.87</td>
</tr>
<tr>
<td>developers</td>
<td>StD: 75.15</td>
<td>StD: 75.52</td>
<td>StD: 78.16</td>
<td>StD: 82.90</td>
</tr>
<tr>
<td>Density</td>
<td>Average:0.049</td>
<td>Average: 0.049</td>
<td>Average: 0.038</td>
<td>Average: 0.051</td>
</tr>
<tr>
<td></td>
<td>StD:0.10</td>
<td>StD:0.10</td>
<td>StD:0.069</td>
<td>StD:0.079</td>
</tr>
</tbody>
</table>

Table 2 Statistics of Developer-Developer Network

### Variables and Model

The dependent variable, generative diffusion of innovation (Gen_Diffusion), will be measured as we described in Zhang et al. (2012) through the calculation of source code changes. The core idea of generative diffusion of innovation is the digital artifact been diffused is continuously evolving. By
analyzing at source code level, we will be able to identify any micro level changes (such as addition, deletion or modification) so that we can truly measure how software project is evolving overtime, which cannot be answered though analysis at macro level (i.e. size of project). The independent variables are the network density measures for the developer-project and developer-developer networks ($D_P\_Network$ and $D\_D\_Network$ respectively), which will be measured as the portion of pairwise ties that are present in the network. Control variables are measured as follows: Sponsorship is measured as a binary variable with 0 representing company-sponsored project and 1 representing non-sponsored project. For developer, tenure is measured as how long has the developer joined Github. Engagement is measured as total number of activities within one month. We will collect data at different time points in order to create a dynamic network so that we will be able to examine the relationship between network and generative diffusion. In particular, we will analyze how the network at time $t$ may affect the generative diffusion at time $t+1$. By doing so, we can mitigate the causality issue between the change of network and the change of generative diffusion. The model we plan to test is as follows: for a project $i$ at time $t+1$,

$$\text{Gen \_ Diffusion}_{t+1} = \beta_0 + \beta_1 \times D\_P\_Network + \beta_2 \times D\_D\_Network + \beta_3 \times D\_P\_Network + \beta_4 \times D\_D\_Network + \beta_5 \times \text{Controls} + \delta_i + \varphi_i + \epsilon_{it}$$

**Future Plan**

By ICIS 2014, we plan to complete the measurement of all variables and conduct the analysis. In addition to the model specified in the equation above, we will also investigate possibilities of other measurement that may provide additional insights. For example, network density measure may not be adequate to capture the whole picture of the network. The data enables us to compute alternate measures of network ties such as centrality. Also, all project developers (or non-member developers) are not equal. We currently assume the homogeneity of project and non-member developers. However, the more popular and matured projects are more likely to have different levels of ideas offered to developers connected with. Therefore, a weight (such as number of project member or download counts) can be assigned to adjust the ties, similar for non-member developers.

**Expected Result and Contribution**

Through our proposed analysis, we expect to find the different effects of two networks on the generative diffusion. To our knowledge, this is the first study that compares two types of knowledge networks with different mechanisms for generative diffusion of innovation. This study will contribute to both academic research as well as industry practice. The study contributes to research on open source development literature by putting the research in a much broader social networking context that brings change of traditional collaboration (much bigger scale of collaboration) and change of integration of social networking (more ways to communicate) in open source community. The study also contributes to the research on diffusion of innovation by looking at the diffusion in the lens of generativity and setting it in a nest of two types of innovation networks, providing insight on how can we stimulate the diffusion in current generative era. Finally, we will contribute to the literate of social network by showing how the social network has changed open source development and diffusion of innovation. To practitioners, our study will provide insight to those coding platform providers such as Github or Sourceforge how they can make their platform more vital. As more and more project hosting website try to provide features to facilitate developers socially connected, whether such social networking can really bring benefit to project development remain unanswered. Our study can also help those companies using those platform as a way of open innovation such as Twitter or Facebook to make better decision on how they can make their projects more popular and active through promoting the use of social networking.
References