The role of social computing as enabler of peripheral participation in open source innovation:
through the lens of stigmergy

ABSTRACT

Peripheral developers in open source software development (OSSD) are often overlooked despite the fact that they account for most contributions in OSSD. Still little is known about how peripheral developers make such impacts on OSSD projects and how online code hosting platforms support peripheral participation. In this paper, we examine these issues with empirical data from the OSSD platform, Github. Drawing on the theoretical lens of stigmergy, we hypothesize that social computing functions of the platform enable peripheral developers to have a positive impact on OSSD projects. With empirical data from a sample of 2034 projects and 544 developers accounting for the entire population of JavaScript projects and related developers on Github, two studies were conducted to examine how social computing functions enable peripheral participations. We find that greater use of peripheral contribution support features (PCSF) is positively associated with code maintainability. Also, peripheral developers contributed as a result of both quantitative stigmergic factors such as activeness and popularity of projects, and qualitative stigmergic factors such as creation-related events. The paper extends research on stigmergic interactions in OSSD and also sheds light on how the design of online collaboration platforms may enable peripheral participation in open innovation.

Keywords: open source innovation, peripheral participation, social computing, stigmergic interaction, self-organization
The role of social computing as enabler of peripheral participation in open source innovation: through the lens of stigmergy

INTRODUCTION

Open source software development (OSSD) has positioned itself as a major method of software development (Bonaccorsi and Rossi, 2003; Weber 2004). The advent of online code repositories and version control systems (VCS) to support distributed collaboration in OSSD has also promoted increased participation with improved convenience. Most OSSD projects are hosted in online platforms offering not only code repositories and VCS but also various social computing features for effective collaboration. Of these platforms, Github is the most popular, with more than 7.1 million registered users and 15.9 million projects as of September, 2014 (http://github.com/about/press). The core spirit of OSSD lies in voluntary participations from publicizing the source code and bottom up governance (Hars and Ou, 2001; Raymond 1999). Yet, this voluntary participation comes mostly from peripheral developers, who are not highly involved with the projects. According to the literature, over 75% of total contributors are one-time contributors (Ghosh and Prakash 2000; Lerner and Tirole 2002).

An OSSD developer may be a core member of one project at the same time as being a peripheral developer on another project. Developers have limited cognitive resources and time to contribute to those projects in which they are not a core member. Hence, it is natural for any OSSD developer to act as a peripheral developer in such projects. Despite the prevalence of peripheral contributions, most studies focus on understanding how such
Peripheral participants become a core member in open innovation projects (Lave and Wenger 1999; von Krogh et al. 2003). However, few studies have investigated the impact of peripheral contribution itself on an OSSD project (for an exception see, Setia et al. 2012).

The association between the characteristics of the online platform where an OSSD project is hosted and peripheral contributions to the project, however, is rarely studied even in the research focusing on peripheral participants. While core members are likely to exploit diverse communication channels to manage a project, through both internal channels implemented in the platform and external channels such as mailing lists and messengers, peripheral developers will be mainly bound to the internal communication functions adopted by the platform, and will not exploit external channels to contribute to projects. Although opportunities for contributions may be serendipitously encountered through external channels, such as media and OSSD related forums, such channels offer no direct clues as to what specific contribution is needed by a focal project.

Peripheral developers are not as well informed or acquainted with the focal project as core members. Thus, it is important for them to know when and what is needed to contribute at a minimum cost in terms of time and effort. Social computing features on Github, in particular the watch and follow features support constant awareness of other projects and developers on the platform (Dabbish et al. 2012). These social feeds signal ongoing project activities, enabling peripheral developers to make impactful contributions to the focal projects, by increasing the chances of becoming aware of a contribution opportunity.

What impact, then, do such peripheral contributions have on a project? In line with Raymond’s observation that "given enough eyeballs, all bugs are shallow" in OSSD (1999), peripheral developers mostly perform bug fixes. Recent studies found that peripheral developers mainly engaged in quality assurance of OSSD projects by refining existing code
(Setia et al. 2012). In short, peripheral developers contribute mostly to improving the meta-quality of the code while not altering the core of the code. We thus argue that peripheral developers on Github will make similar contributions, with the only difference that they have increased awareness of when and what part of the code through their social feeds. Thus, we examine what impact peripheral developers have on code maintainability in OSSD projects.

The goal of our study is to investigate how social networking functions that we label peripheral contribution support functions (PCSF) trigger peripheral developers to contribute to projects where they are not core members. We adopt the theoretical lens of stigmergy to explain the bottom up governance through such indirect interactions (Grasse 1995). Stigmergy is an essential mechanism through which bottom-up self-organization process emerges (Bonabeau et al. 1997). The theory was developed in the field of zoology to explain how collectives of insects and animals self-organize without top down governance and direct communications (Grasse 1959). According to research on the stigmergic process, self-organization emerges through indirect interaction between agents who react to and are influenced by the environment and artifacts embedded in the environment that provide cues about the preceding actors’ actions (Holland and Melhuish 1999; Theraulaz and Bonabeau 1999). This concept has been conceptually adapted to examine OSSD communities (Howison et al. 2012). Our research further builds on such research, and presents results of a large-scale empirical study of the phenomenon. Furthermore, our research addresses the following research questions. 1) Does the use of peripheral contribution support functions (PCSF) impact open source code maintainability? 2) What are the individual level choices through PCSF that presents as stigmergic interactions? To answer these questions, we conducted two separate studies to examine the association between the social feature use by developers and code maintainability at the project level, as well as between peripheral participation and
stigmergic information at the individual level.

The paper is structured as follows. Section 2 presents theoretical perspective of stigmergy on which our study is grounded. We examine prior research on OSSD and our research framework in section 3. Section 4 and 5 provide details for each study conducted at the project level and individual level, respectively. Section 6 summarizes the results of the studies, and section 7 concludes with a discussion of the limitations and implications of our study.

THEORETICAL BACKGROUND

The influence of peripheral developers on open source software development projects

The peripheral developer in our research context refers to the contributor in a project who does not belong to its core development team. Xu divided the role of participating developers into 4 categories, including peripheral, central, core developer, and project leader. In this classification, peripheral developers are described to make minor contributions to a project irregularly (Xu 2004). Ye and Kishida had more diverse classifications of 6 groups depending on the types of role each group plays, involving project leader, core member, active developer and peripheral developer, bug fixer and bug reporter. Also, the peripheral developer is characterized as making sporadic and short-period contributions (Ye and Kishida 2003). Recently, a dyadic classification between core and peripheral members is commonly adopted in OSSD research with such research focus that often highlights the contrast between two groups with most distinctive difference in their contribution patterns (Terceiro et al. 2010).

The studies about the impact of peripheral developers are still not sufficient, but few studies focusing on the topic provided interesting findings about the pattern of contributions
by peripheral developers and the impacts they have on OSSD projects. A study conducted in-depth investigation regarding the role of peripheral developers within an OSSD project through visualization of development process (Souza et al. 2005). According to the result, peripheral developers showed circumscribing interaction patterns around core developers, sometimes shifting its role with core developers. This shifting of roles took pace when a module of code written outside a project was adopted in the project, or a peripheral developer's understanding about the project improves after certain time amount. The influence of peripheral developers on OSSD projects is often undermined given that their expertise and understanding about a project are lower than that of core members. Due to this lack of expertise and knowledge, peripheral developers often increase the structural complexity of a project's source code, resulting in more efforts required on the project leader's end (Samoladas et al. 2004; Terceiro et al. 2010).

However, a recent study showed that peripheral developers can influence OSSD projects in a positive way by being engaged in quality assessment and enhancement. It is found that the bigger number of contributions by peripheral developers is positively associated with more bug reports and bug fixes, whereas the association between contributions by core developers and the same dependent measures turned out to be negative (Setia et al. 2012). In line with the study, our research also focused on the positive influence of contributions by peripheral developers to OSSD projects, further accounting for the facilitating effect of information technology, Github, in the process.

**Stigmergic Coordination**

Stigmergy is the major mechanism that enables diverse self-organization phenomena. The term is defined as the indirect local interactions occurring between an agent and the environment where the agent is present, through the information or artifact embedded in the
environment left by other agents (Heylighen. 2006; Theraulaz and Bonabeau 1999). It also explicates how a higher order (corresponding with project-level in our research context) can arise even when individuals are not aware of or willing to build the order. Although the term ‘stigmergic interaction’ was first coined in the context of insect colonies (Grassé 1959), previous research has observed the phenomenon in human collectives as well (Ricci et al. 2007; Van Dyke Parunak 2006). The flocking of birds and the herding of social animals or insects are other examples of self-organization phenomena that can be understood with the concept of stigmergic interaction (Bonabeau et al. 1997; Theraulaz and Bonabeau 1999).

Recent research suggests that the online development environments on which multiple OSSD projects are hosted may provide an ideal setting within which self-organization emerges through stigmergic coordination (Bolici et al. 2009; Crowston et al 2007; Heylighen 2007). Popular online development environments today provide a persistent trace of information cues regarding participants’ activities, and thus have the potential to enable stigmergic interaction and coordination amongst OSSD project participants (Susi, 2001). Additionally, online collaborations have high dependencies with online collaboration platforms, since they need the platforms for mediating the information exchanges throughout the entire development process. Therefore, the features and functions provided by an online collaboration platform can facilitate or hinder the group coordination process by mediating participants’ interactions (Bafoutsou and Mentzas 2002). Given that, the concept of stigmergy is appropriate for the theoretical base of our research, particularly for its emphasis on artifacts embedded in an environment as communicative media.

In our study, we contend that stigmergic interaction between the digital environment and developers will be facilitated by the Github platform, since the PCSF provided by the Github platform automatically convey other developers’ and projects’ activity information to
individual developers. In order to substantiate the argument, we first investigated the general association between the use of PCSF and group coordination performance at project level. We then aimed to identify how stigmergic interactions through PCSF are presented in various fashions.

Previous studies have conceptualized stigmergic interaction as in Figure 1 (Dorigoa et al 2000; Van Dyke Parunak 2006). The dynamics caused by the activities of the agents alter the environmental state in which the agents dwell, but then again the altered environmental state induces changes in the activities of other agents. Taken the trail of ants as an example, an agent conducted a seed behavior of leaving a pheromone in certain areas to which the one is heading. Then following agents perform the same actions on one of the spot where the first pheromone is left by perceiving it, reinforcing the following behaviors triggered by the pheromone. After the several cycles of this serial actions, the group of ants following a lined path appears without top down governance (Bonabeau et al 1997; Karsai 1999). In open source software development, this stigmergic interaction appears naturally to cope with the limited capacity of an individual to engage in explicit communications with a large number of agents as the project size grows (Heylighen, 2006; Valverde et al 2006; Yu 2008).

![Figure 1 Stigmergy / Stigmergic interaction](image)

The logic of stigmergic interaction as shown in Figure 1 is operationalized to fit our research context, Github (Figure 2).
The peripheral contribution support feature implemented by Github itself will support the formation of relationships between Github users and projects, also triggering environmental change by simultaneously delivering the social feeds generated by other developer activities within each project. These feeds will then bring changes to the digital environment triggering other developers, and indirect interaction via a stream of social feeds will naturally occur as this cycle repeats itself.

Stigmergic interaction works as a ground to we’ assumptions about enabling mechanism of improving code maintainability of source codes through peripheral contributions in Github. By using PCSF embedded in Github, developers are able to track the current status of projects in real time, ready for taking peripheral contributions when they are needed.

RESEARCH DESIGN

Research Framework

To answer research goals and question, overall research framework is constructed consisting of two related studies. Study 1 aimed to identify the general association between the number of social computing features including PCSF used for each project and its code quality measures across projects. How the individual developer uses PCSF and reacts accordingly is explicated in Study 2.

Research Background: Github

Github is an OSSD project foundry that supports git as the version control system, and thus
enables much faster and distributed development (Hamano 2006). Since its launch in 2008, Github has grown explosively. According to data from January 2011 to May 2011, Github ranked first among online code hosting sites in terms of the total number of code commits. Its 1,153,059 commits accounted for 54% of all code commits at popular OSSD hosting sites (Redmonk, 2011). Examples and explanations of the Github pages are given below.

Figure 3 is the example of a project page provided by Github. The page is also called ‘Repository’. Sometimes a repository is a sub-project under a bigger project, while a repository equals a project most of the time. Red box (1) in figure 3 indicates the project title, which is composed of its owner/project name. Box (2) indicates the number of people who watched and forked the project.

![Github Project Page Example]

**Figure 3. Project page on Github**

Github also provides social computing features that facilitate social interaction within
projects for effective coordination. 1) Github’s Fork feature explicitly provides developers with the ability to create deviations from the project’s original code by creating a linked copy of the project code to which developers can directly add new lines of code. While forking has generally been regarded in a negative light by OSS communities, forking is publicly encouraged and easy to perform in Github. 3) Github also provides issue and pull request features in the form of communication forums for project participants. Of these, the issue forum has no restrictions on participation, whereas the pull request forum is a space where developers who have made a code contribution may communicate. 4) Github also automatically tabulates and prominently displays the number of contributors. 5) The commit feature is based on the git version control system, and hence enables a less restricted modification of existing codes and the addition of new codes compared to the previous development environment. 6) Finally, Github provides a user profile feature that displays, for each developer, personal information, the list of projects he or she owns or is participating in, the number of other developers whom they follow, and the number of projects they are watching.
Figure 4. Developer page in Github

The above image is the screen capture of a developer page on Github. This page is very similar to the profile pages provided by Social Network Sites (SNS) in the way that it displays a developer’s profile and the number of social relationships that he has (annotated as (2) in the figure 4). The red box (1) represents the developer’s handle and real name. This page also contains information about the repositories owned or forked by the user, and his/her recent activities in Github. Github converts all the activity logs of its users into XML feeds so that they can easily be broadcast to their followers, as is the case with the RSS feeds mechanism.
In Github, when one user A “follows” another user B, user A can receive all of B’s activity feeds delivered into A’s dashboard. The same thing happens when user A chooses to “watch” project B. All the events occurring within the project stream into the dashboards of the project’s watchers. The dashboard (Figure 5) is the web page that users see first when they log in to Github. It shows the activity feeds from watched projects and followed users synchronously. The types of feeds that can be shown on a dashboard are listed in Table 1. What users receive by either watching a project or following a developer is marked on the right side of each type of feed.

**Table 1.** List of Feed Types ([http://developer.github.com/v3/events/types/](http://developer.github.com/v3/events/types/))
<p>| Event Type       | Description                                                                 |  |  |
|------------------|-----------------------------------------------------------------------------|  |  |
| Commit Comment   | Comments show up in feed and each repository has its own comment feed.     | 0 | 0 |
| Create Event     | Represents a created repository, branch, or tag.                             | 0 | 0 |
| Delete Event     | Represents a deleted branch or tag.                                          | 0 | 0 |
| Download Event   | The downloads API is for package downloads only.                             | 0 | 0 |
| Follow Event     | The event’s actor is the follower, and the event’s target is a user of Github. | 0 | 0 |
| Fork Event       | Create(s) a fork for the authenticated user.                                | 0 | 0 |
| Fork Apply Event | Triggered when a patch is applied in the Fork Queue.                        | 0 | 0 |
| Gist Event       | Users can read public gists and create them for anonymous users without a token. | 0 | 0 |</p>
<table>
<thead>
<tr>
<th>Event Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gollum Event</td>
<td>Creation, modification, or deletion of wiki events.</td>
</tr>
<tr>
<td>Issue Comment</td>
<td>Issue Comments leverage custom mime types.</td>
</tr>
<tr>
<td>Issues Event</td>
<td>Records various events that occur around an Issue or Pull Request.</td>
</tr>
<tr>
<td>Member Event</td>
<td>Triggered when a user is added as a collaborator to a repository.</td>
</tr>
<tr>
<td>Public Event</td>
<td>This is triggered when a private repo is open-sourced.</td>
</tr>
<tr>
<td>Pull Request Event</td>
<td>Once a pull request has been sent, interested parties can review the set of changes, discuss potential modifications, and even push follow-up commits if</td>
</tr>
<tr>
<td>Event Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Pull Request</td>
<td>Necessary. Pull Request Review Comments are comments on a portion of the unified diff.</td>
</tr>
<tr>
<td>Review</td>
<td></td>
</tr>
<tr>
<td>Comment Event</td>
<td></td>
</tr>
<tr>
<td>Push Event</td>
<td>Make commits to a project.</td>
</tr>
<tr>
<td>Team Add Event</td>
<td>Add(s) a collaborator to a project or to an organization.</td>
</tr>
<tr>
<td>Watch Event</td>
<td>The event’s actor is the watcher, and the event’s repo starts being watched.</td>
</tr>
</tbody>
</table>

Of social computing features that Github provides to support OSSD, we focus on two PCSF that enable interactions across projects, and distinguish Github from other OSSD platforms: 1) by “following” other developers, one can keep abreast of the developer’s coding activities across multiple projects, while 2) by “watching” other projects, one can keep track of all changes to the projects that one is interested in, regardless of actual membership in the project. The watch and follow relationships are prominently displayed on the Github developer and project pages. Together, these features enable connections to be made between registered developers on Github and projects hosted on Github, much akin to the links forged
within social networking sites, and provide the basis for increased social interactions beyond
the boundaries of one’s own OSSD projects. Every event generated by a developer or project
that one has watched or followed will generate information cues that are prominently
displayed in the form of dynamically generated social feeds on Github’s landing page. In this
study, we focus on how these characteristics of Github trigger stigmergic interactions with
respect to development.

STUDY 1

A study on the relationship between social computing feature use and the code
maintainability of OSSD projects

The purpose of study 1 is to find a general relationship between the use of social
computing features and the quality of outcomes across projects in Github. Particularly, we
tried to identify the relationship between PCSF and project outcomes, particularly the code
maintainability of source codes, in an attempt to sharpen the focus of this study.

Hypothesis

We assumed that developers will perform peripheral contributions through indirect
interaction with feeds from using PCSF. Adding to that, once multiple peripheral developers
come to receive identical feeds simultaneously, the total number of contributions may
increase by creating contribution opportunities if otherwise would have been missed without
receiving the feeds.

Given that this contributions by peripheral developers require only indirect communication
prior to being made, it can be assumed that the contributions tend to be of refining the
previously written codes or fixing bugs, which do not have direct effects to the content of the
codes of each project. The refinement of codes typically involves code refactoring, or
versioning up a module of code. Thus, peripheral contributions are thought to be in close relationship with code maintainability. So we hypothesized as following.

_Hypothesis 1. As more features attracting contributions by peripheral developers are used for a project, the code maintainability of the project will be improved._

Features supporting social interactions in each project are also included as variables in order to identify relative effects by the features of interest of which number stands for more potential peripheral developer base. Each variable is elaborated as following.

**Independent Variables: social computing features on Github:** We constructed a matrix between the use of PCSF and the code maintainability of the source codes measures about Github projects for statistical analysis. The features used in the analysis include the numbers of watchers, followers, forks, pull requests, issues, and commits, which consist of entire PCSF provided for each project on Github as of April, 2012.

**Dependent variables: Code maintainability of OSSD projects:** Regarding the measure for the quality of OSSD project outcomes, our research has conditioned it to be quantifiable measures for the maintainability of source codes for statistical analysis. Since no single measure for maintainability of the source codes exists drawing on its subjective nature (Stamelos et al. 2002), most studies about OSSD customize the success measures according to their context (Capiluppi et al. 2009). In our research, two types of code maintainability measures (violation total, complexity per function) by Sonar\(^1\) represents the influence of peripheral contributions. OSSD projects are considered to have difficulties in achieving high-

---

\(^1\) Sonar, a well-known open-source tool for code maintainability of the source codes evaluation [http://sonarsource.org](http://sonarsource.org)

\(^3\) [http://docs.codehaus.org/display/SONAR/Metric+definitions](http://docs.codehaus.org/display/SONAR/Metric+definitions)
level maintainability in particular due to its voluntary governance (Samoladas et al. 2004). Violation Total (following VT) represent the total number of violations to the traditional code conventions. Complexity per Function (following CF) is earned from dividing complexity score computed as the Cyclomatic Complexity Number (Denaro and Pezze, 2002), which measures how complicated the structure of a unit of source code (usually a function) is by the number of Functions in a project’s source codes. The regression model for each dependent measure is as follows.

\[
\text{Violation Total} = \beta_0 + \beta_1 \text{Watcher} + \beta_2 \text{Follower} + \beta_3 \text{Fork} + \beta_4 \text{Issue} \\
+ \beta_5 \text{Commit} + \beta_6 \text{Pullrequest} + \epsilon
\]

\[
\text{Complexity per Function} = \beta_0 + \beta_1 \text{Watcher} + \beta_2 \text{Follower} + \beta_3 \text{Fork} + \beta_4 \text{Issue} \\
+ \beta_5 \text{Commit} + \beta_6 \text{Pullrequest} + \epsilon
\]

Data Sampling and Collection

In order to address the phenomenon empirically, web log data was used as the main data of the study. Two sample sets are composed, one is at project level for study 1 and the other at individual developer level sample accounting for the scope of study 2, respectively. Both sets are coherent in that only those sample developers, who participated in the sample projects for study 1, are included for Study 2. A custom script using Github’s API logged all users’ actions every three minutes into a database. These included: who (developer), what (action), where (project), and when (timestamp). The types of actions included are: commit comment, create, delete, gollum, issue comment, issues, pull request review comment, pull request, push, and watch events (which are further explained in https://help.github.com/categories/63/articles). Furthermore, supplementary metadata was collected. This included: when a project first began, who developers were connected to (i.e., who they were following, and who were following them), and what projects they were connected to (i.e., watching).
The project sample is constructed as follows. In order to test the association between code maintainability and social computing feature use across projects, we sampled all the JavaScript projects on Github on 31 August 2011, as it is the most frequently used language in Github (https://github.com/languages). Additionally, by focusing on one language, the result is less confounded by the differences between projects based on different languages. We included only the projects which made their first commit on the day after the ‘pull request’ feature became available to public (i.e. after 1 September 2010). This is because pull request is an important feature that supports extensive communication related to a project. The number of JavaScript projects which had their first commit after 1 September 2010 came to 104,838, accounting for 70.38% of the full sample of JavaScript projects (148,960 projects). Projects that were forked from other projects were excluded due to data integrity issues, resulting in a sample of 49,643 projects (i.e. 33.3% of all sampled projects). In order to investigate the effect of PCSF, projects which did not once make use of PCSF were also dropped. Projects that had more than two watchers (17,412 remaining projects, 11.7% of all the projects), two forks (8,194 projects, 5.5% of all the projects), one pull request (2,783 projects, 2% of all the projects), and one contributor (2,034 projects, 1.4% of all the JavaScript projects) were selected. Our final sample consisted of 2,034 projects accounting for total population of JavaScript projects on Github with at least one event of every social activity. Afterwards, the number of the use of each social computing feature for sampled projects is collected by parsing data from the Github using Github API and a custom-developed data parser.

Cross-sectional linear regression analysis is performed for data analysis with PASW (SPSS) 18. Since the distributions of all variables were highly skewed, we performed log transformations for each variable.
Result

Descriptive statistics and Intercorrelations

The average number of uses of each feature per project is as follows: 58.66 for Watch (ranging from 2 to 3421, Median = 9, SD = 196.13), 21.88 for Public Repo (ranging from 1 to 3421, Median = 12, SD = 31.11), 88.69 for Commit (ranging from 2 to 3259, Median = 37, SD = 187.77), 80.27 for Follow (ranging from 0 to 3739, Median = 11, SD = 261.52), 16.10 for Issue (ranging from 1 to 2426, Median = 4, SD = 84.08), 8.02 for Fork (ranging from 2 to 584, Median = 3, SD = 22.32), 5.53 for Pull Request (ranging from 1 to 463, Median = 2, SD = 18.41) and, lastly, 3.38 for Contributors (ranging from 2 to 100, Median = 2, SD = 4.26). In sum, these features have been utilized in the projects with huge variance.

Table 2. Intercorrelation between variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
<th>(7)</th>
<th>(8)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) Watch</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(2) Fork</td>
<td>0.81**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3) Issue</td>
<td>0.49**</td>
<td>0.65**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4) Commit</td>
<td>-0.39**</td>
<td>0.45**</td>
<td>0.60**</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(5) Follow</td>
<td>0.17**</td>
<td>0.08**</td>
<td>0.01</td>
<td>-0.01</td>
<td>1.00</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(6) Pull.req</td>
<td>0.57**</td>
<td>0.71**</td>
<td>0.74**</td>
<td>0.62**</td>
<td>0.05’</td>
<td>1.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(7) Complexity</td>
<td>-0.06**</td>
<td>-0.05’</td>
<td>-0.04</td>
<td>-0.04</td>
<td>-0.07’</td>
<td>-0.04</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>(8) Violation</td>
<td>0.18</td>
<td>0.05’</td>
<td>0.10**</td>
<td>-0.21”</td>
<td>-0.05’</td>
<td>0.11”</td>
<td>-0.05’</td>
<td>1.00</td>
</tr>
</tbody>
</table>

As some of intercorrelations have values above 0.5, the multicolinearity and Durbin-Watson test were conducted in the following analysis to assure the validity of our regression model.

Multivariable Linear Regression Test

We explored the effects of the social computing features on the two code maintainability measures as separate regression models. The regression model for each outcome variable
satisfied the statistical criteria for valid regression models with VIF less than the value of 10 for multicolinearity test, and Durbin-Watson close to the value of 2 (1.897, 1.956). The results are shown in Table 3. The first column, showing the model with VT as the dependent variable, is also significant overall (F (5, 1995) = 108.668, p<0.001, Adjusted R² = .256). The second model, which uses CF as the dependent variable, is described in the last column, and also shows overall significance (F (5, 2001) = 23.411, p<0.001, Adjusted R² = .068).

**Table 3.** The results for multivariable regression analyses

<table>
<thead>
<tr>
<th>Variable</th>
<th>VT</th>
<th></th>
<th></th>
<th></th>
<th>CF</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B</td>
<td>S.E.</td>
<td>β</td>
<td>B</td>
<td>S.E.</td>
<td>β</td>
<td></td>
</tr>
<tr>
<td>(Constant)</td>
<td>1.561</td>
<td>.049</td>
<td>-130**</td>
<td>1.590</td>
<td>.064</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watch</td>
<td>-.124</td>
<td>.040</td>
<td>-130**</td>
<td>-.285</td>
<td>.051</td>
<td>-.259**</td>
<td></td>
</tr>
<tr>
<td>Fork</td>
<td>.241</td>
<td>.072</td>
<td>145**</td>
<td>.460</td>
<td>.094</td>
<td>.239**</td>
<td></td>
</tr>
<tr>
<td>Issue</td>
<td>-.018</td>
<td>.043</td>
<td>-.016</td>
<td>.069</td>
<td>.056</td>
<td>.054</td>
<td></td>
</tr>
<tr>
<td>Commit</td>
<td>.634</td>
<td>.032</td>
<td>513**</td>
<td>.312</td>
<td>.042</td>
<td>.218**</td>
<td></td>
</tr>
<tr>
<td>Follower</td>
<td>-.085</td>
<td>.019</td>
<td>-.098**</td>
<td>-.098</td>
<td>.025</td>
<td>-.097**</td>
<td></td>
</tr>
<tr>
<td>Pull Request</td>
<td>-.054</td>
<td>.051</td>
<td>-.037</td>
<td>-.278</td>
<td>.066</td>
<td>-.165**</td>
<td></td>
</tr>
<tr>
<td>Adj. R²</td>
<td>.256</td>
<td></td>
<td></td>
<td></td>
<td>.068</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>108.668**</td>
<td></td>
<td></td>
<td></td>
<td>23.411**</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VT: Violation Total, CF: Complexity per Function (*: p<.05, **: p<.01)

Consistent patterns between the number of social computing features used and code maintainability appeared in the results obtained for the two measures. Across two models, the
numbers of issues does not have any significant effect. Both the numbers of watchers and followers for either project or the owner of the project negatively affect the measures, that is, they had a positive effect on improving the maintainability of source codes. In contrast, the numbers of forks and commits significantly increases both complexity and violation. The number of pull requests is in significantly negative relationship only with code complexity per function.

The interesting implication from this result is that the features inducing the inflow of peripheral contributions are in significant negative relationship with the code quality. This supports our hypothesis that more use of the features forming broader peripheral developer network will have improving effects on code maintainability. As long as the developers are logged on Github, they are receiving news feed from the projects that they are watching forming a peripheral developer networking of the project. The number of public repositories by the project owner also broadens the number of feed receivers from the focal project thereby offering more chances for peripheral contributions by peripheral developers. We speculated the mechanism between the relationships as following. Knowing the presence of opportunities for minor contributions without background knowledge is important for peripheral developers who are not acquainted with a certain project to make peripheral contributions to the project. The use of PCSF increases the opportunities through stigmergic interactions via feed stream. Hence it appears as the significantly negative relationships between code complexity, which is the reverse indicator of code maintainability, and the uses of the features related to attracting peripheral contributions.

**STUDY 2**

*Types of Stigmergic interactions on Github*
Drawing on the theory of stigmergy, Study 2 investigates individual developers’ choices to make peripheral contributions through PCSF, possibly leading to the effect on the code maintainability. Peripheral contribution was regarded as the indicator of stigmergic interactions by peripheral developers. Entire developers who have contributed to the sample projects in Study 1 were taken as samples for Study 2. Same as Study 1, this sample set is reliable in that no arbitrary sampling was performed from total population. We argue that peripheral contributions without intensive direct communication is only possible when the contribution is minor and the contributor notices the contribution opportunities indirectly from social feeds. To examine that feeds from using PCSF do engender peripheral contributions, following hypotheses are developed.

Hypotheses

Stigmergic interaction is suggested to be already a pervasive phenomenon both within an online platform like Github and any public offline environment (Couzin and Krause 2003; Cui et al 2009; Marsh 2008). The subject of our study, however, is specified to those stigmergic interactions that induce peripheral contributions over OSS projects since our purpose is to account for how contributions by peripheral developers have impacts on OSSD projects. In order to account for various types of stigmergic interactions at most, we distinguished between quantitative and qualitative stigmergy drawing on existing studies of stigmergy (Theraulaz and Bonabeau 1999). Also, possible cues that can be signaled from the feeds on Github were first identified, resulting in 5 categories of cues: process efficiency, feed generator’s popularity or activeness, the focal project’s popularity or activeness, project language use and types of work being done (the type of a feed signaled).

Quantitative Stigmergy: Quantitative stigmergy stands for the indirect interactions between two agents, especially when the actions of a following agent being triggered by the mere presence or
quantitative increase of the cues left by the preceding agent (Theraulaz and Bonabeau 1999). We argue that the changes in the number or time intervals of these cues is likely to incur indirect interactions between the generator of the cue (either its project or an owner developer of the project) and the receptor of the cue (periphery developers), leading to peripheral contributions made to the pertinent project by the receptor of the cue. Each type of cues is elaborated respectively in the following.

The social characteristics are easily observable and accessible from the newsfeed page of the developer. These cues can be distinguished between cues originating from a developer or from a project. Developer's social cues are defined as the success and the popularity of the developer that is being followed. Crowston et al. (2003) defined activity as one of the success factors. The activity is easily noticeable because more active developers will generate more notifications on their followers' newsfeed page. Therefore, the activeness of a developer is defined by the number of newsfeed notification a developer creates on their followers' newsfeed page per day (Dev.Act.). Furthermore developers with thousands of followers are deemed to have some special skill or knowledge for others to learn from them (Dabbish et al. 2012). Therefore, the popularity of a developer is defined with the number of followers they have (Dev.Pop.).

Hypothesis 1a. The likelihood that a receptor of a feed will perform a peripheral contribution to the project is positively related to the feed generator's popularity or activeness.

Project's social cues are just as the developer's social cues easily accessible and observable from the newsfeed page of developers. The activeness of the project is conceptualized by the number of notifications a project creates on the developer's newsfeed page (Proj.Act.). These notifications can be created by any actions by a peripheral developer in the project as well as accumulation of the actions by multiple developers. The measurement for popularity is therefore being defined as the number of different developers contributing to the project's
activity (Proj.Pop.).

*Hypothesis 1b. The likelihood that a receptor of a feed will perform a peripheral contribution to the project is positively related to the project's popularity or activeness.*

Previous research has shown that the time it takes for code bugs to be fixed is an indicator for the project's success (Crowston et al. 2003). In line with this we chose to use the average time of closing an issue or responding to a pull request as indicators for the project's process efficiency (Proj.Close). That is, the time duration that a problem is solved or the time duration a code change is accepted or rejected in the project. This signals how active the project is, and developers will take bigger interest in more active project leading to more peripheral contributions to the project. Therefore, we hypothesized as below.

*Hypothesis 1c. The likelihood that a developer will perform a peripheral contribution to the project is positively related to the process efficiency.*
**Qualitative Stigmergy:** Qualitative stigmergic interaction refers to the indirect interactions that following agent who recognize the cues left by preceding agent changes his behavior types according to the qualitative difference of the cue (Theraulaz and Bonabeau, 1999). In Github context, qualitative stigmergy is an imperative mechanism for peripheral developers to find rightful opportunities for contributions. Drawing on the content of feeds on Github, two qualitative cues can be signaled to peripheral developer within a networked formed by the use of PCSF. First is the programming language used and the other is the type of contributions made to the focal project. These cues hold imperative information as to the peripheral developer’s competency (programming language) and the work requirements (types of prior work in progress) of the opportunities through feeds. How each cue triggers stigmergic interactions leading to peripheral contributions is described in detail below.

Matching programming language is one of the basic requirements for periphery developers to make any contributions (Mockus et al 2002; Subramaniam et al. 2009). Peripheral developers will not normally make contributions to the projects of which programming language is not familiar (Bagozzi and Dholakia 2006). The programming language, however, is not directly observable from a developer's newsfeed page or from the project's information page. Dependent on the project, the developer needs to evaluate different information to discover the programming language to decide if they are able to contribute with their current skills. A lot of programming languages have a common foundation; a developer is able to contribute to a project without having a perfect match with their current skills. To measure how well the programming language of a project matches the developer's programming skills we calculated the ratio between the programming languages the developer uses in his own project and the programming languages of projects seen on the developer's newsfeed page. That is, a smaller ratio means that the developer is less familiar with the programming language (Proj.Lang).
Hypothesis 2a. The likelihood that a developer will perform a peripheral contribution to the project is positively related to a closer match of their programming skills and the programming language of the project.

As the spectrum of contributions possible based on indirect interactions is very limited, the decision about "what" contributions to make is less important than "when" to make the contributions. Accordingly, the peripheral developers will be more responsive to the type of feeds that signals the right timing for peripheral contributions. Each feed broadcasted from a project has a different importance in terms of its informational value. People tend to weigh information differently in order to strategically understand their surroundings and situations. In our research contexts, it can be assumed that feeds containing the important changes to a project, such as the release of a new version or the creation of a new branch, that are decided by the owner(s) of a project, will be valued more by watchers than other types of feeds. In Github, the watchers of a project will also react and focus more on the types of feeds that indicate important changes to the project. The type of feed reflecting those types of changes is coded as ‘Create Event’ feed in our data, which entails the events of creating a repository, a branch, or a tag. Therefore, we hypothesized as follows:

Hypothesis 2b. A user is more likely to make a peripheral contribution in a project with more 'Create' events during a given period, compared to the other projects that he/she is watching.

Data Sampling and Collection

As to the individual level data, a sample-based longitudinal database was constructed since the phenomenon we were to observe required a certain period of time for individuals to interact and engage. The sample for individual developers entailed 544 Github users who had contributed at least for once to 2,034 sample projects in Study 1 at the time of the collection. Since 2034 sample accounts for total JavaScript projects on Github with social activities after
the adoption of pull request function, 544 developers can be regarded to represent the full samples not biased to an arbitrary criteria. We collected the entire activity feeds generated from our samples. A total of 18 types of activities\(^2\), all types of activities recorded as feeds in Github, were collected. Additionally, we also collected data about with whom the sample developers were connected.

The final dataset used to construct our sample consists of not only the entire activity logs made by 544 developers, but also all logs viewed by the 544 developers from appearing on their feed streams, within a period of 100 days. The total amount for our unit for analysis consists of 5,803,541 events containing 4826 peripheral contributions.

**Analysis Method**

Two independent analyses were conducted to examine the hypotheses due to that only the relationship between the types of feeds and peripheral contribution is based on different unit of analysis. The unit of analysis for six types of cues other than the type of feeds is a triad consisting of a date, a developer and a project, whereas the unit of analysis for the feed type is each instance of feed.

To analyze the influence of prior days we adapted the nonlinear logistic exponential decay models. A decision by a developer to perform a peripheral contribution can be influenced by social feeds seen on a previous day. For example, notifications that a developer has seen over the past few days may have a bigger effect on the developer's decision to perform an action to the project than when a developer perceives a notification for the first time. In order to capture the effects of prior social feeds we adopted up to three prior days and tracked back the social characteristics of developers and projects corresponding to that day. The first

\(^2\) A detailed description of each feed type is provided in section 2 of this paper.
regression model is as following.

\[
\text{Logit(Peripheral Contribution)} = \beta_0 + \beta_1 \sum_{t=0}^{3} Dev.pop^{t}a^{t} + \beta_2 \sum_{t=0}^{3} Dev.Act^{t}a^{t} + \\
\beta_3 \sum_{t=0}^{3} Proj.pop^{t}a^{t} + \beta_4 \sum_{t=0}^{3} Proj.Act^{t}a^{t} + \beta_5 Proj.Close + \beta_6 Proj.Lang + \epsilon
\]

In these models the dependent variable is whether or not an event has been made to a project. \( \beta \) presents the parameter coefficient. The decay is defined by the product term \( t \cdot a \).

Where \( t \) is the prior day and \( a \) is the decay parameter. In order for an exponential decay function to occur, \( a \) needs to be \(< 0\). When \( a \) goes to \( -\infty \) it implies that the influence of the prior days decreases. We found a decay parameter to indicate the depreciation effect of the prior days (\( c_{\text{contribution}} = -1.578, p < 0.001 \)).

Because a peripheral contribution is such a rare event (4826 of 9,921,300 events accounting for less than 0.05% of total events), we adopted a rare event logistic (relogit) to control for the underestimation of event probabilities that occurs with conventional logistic regression when underestimating on the dependent variable (King and Zeng 2001). To be able to control for the bias of under sampling of the dependent variable, two to five times more of zero values compared to the values coded as one is needed for relogit to be reliable. To meet this requirement we sampled the data with a 5:1 ratio. In contrast to random sampling, choice-based sampling strategically selects observations based on values of the response variable \( Y \). Given that observations with a response variable of 1 carry much more information about the variables influencing the occurrence of the event, we adopted a choice-based sampling method.

The robustness of the estimation results are ensured by drawing 1,000 bootstrap choice-based samples of 9,822,087 triads to derive the boot-strap mean and the confidence intervals for each parameter estimate (Efron and Efron 1982; Efron and Tibshirani 1986). Bootstrap
confidence intervals are used to verify the statistical significance of the parameter estimates.

Second regression model to identify the relationship between the type of feeds and peripheral participation is as following. With different unit of analysis from previous analysis, the second regression model was constructed and took the types of feeds that can be seen from a peripheral developer's social feed dashboard as variables. The goal is to understand the relative importance of "create" event feed compared to other types of feeds. Since all variables are highly skewed (>2), we took log transformations to all of them.

\[
\text{Logit(Peripheral Contribution)} = \beta_0 + \beta_1 \sum_{t=0}^{3} Create + \beta_2 \sum_{t=0}^{3} Delete + \\
\beta_3 \sum_{t=0}^{3} Issue + \beta_4 \sum_{t=0}^{3} Pullrequest + \beta_5 \sum_{t=0}^{3} Push + \beta_6 \sum_{t=0}^{3} Gollum + \\
\beta_7 \sum_{t=0}^{3} Comment. issue + \beta_8 \sum_{t=0}^{3} Comment. pull + \beta_9 \sum_{t=0}^{3} Comment. commit + \varepsilon
\]

Result

Inter-correlations and Descriptive statistics

To test for any inter-correlations between variables, we performed a pairwise correlation test of the bootstrapped means (Table 4). Most variables show a weak correlation (<.50). The highest correlation is between project's popularity (Proj.Pop.) and project's activeness (Proj.Act.) (\(\rho = .74, p<0.005\)). By obtaining the bootstrap means of the tolerances and the variance inflation factors, we conducted diagnostic checks for multicollinearity (Green 2000; Mansfield and Helms, 1982). The tolerance (> .20) and the variance inflation factors (< 10) were all within the acceptable margins suggesting that the potential bias in parameter estimates stemming from multicollinearity is not problematic (Neter, Kutner, Nachtsheim, and Wasserman, 1996).

<table>
<thead>
<tr>
<th>Variable</th>
<th>(1)</th>
<th>(2)</th>
<th>(3)</th>
<th>(4)</th>
<th>(5)</th>
<th>(6)</th>
</tr>
</thead>
</table>

Table 4. Inter-correlations.
Significance levels: *p<0.1, **p<0.05, ***p<0.01

The descriptive statistics are presented in Table 5.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Mean</th>
<th>Std. Dev.</th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peripheral contribution</td>
<td>0.000486</td>
<td>0.022050</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Dev.Pop</td>
<td>26.97404</td>
<td>270.8919</td>
<td>0</td>
<td>8267</td>
</tr>
<tr>
<td>Dev.Act</td>
<td>0.162781</td>
<td>2.933395</td>
<td>0</td>
<td>3810</td>
</tr>
<tr>
<td>Proj.Pop</td>
<td>0.207794</td>
<td>1.636043</td>
<td>0</td>
<td>125</td>
</tr>
<tr>
<td>Proj.Act</td>
<td>0.579678</td>
<td>6.573441</td>
<td>0</td>
<td>3810</td>
</tr>
<tr>
<td>Proj.Close</td>
<td>1400869</td>
<td>2991516</td>
<td>0</td>
<td>9200000</td>
</tr>
<tr>
<td>Proj.Lang</td>
<td>0.326759</td>
<td>0.325964</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Quantitative and Qualitative Stigmergy

Table 6 presents the results from first analysis.
More popular (Dev.Pop.) or more active (Dev.Act.) developer will have a bigger influence on their followers performing an action to the project (H1a). Our results indicate a significant and positive effect of popularity (\( \beta = 0.061, p<0.05 \)) and activity (\( \beta = 0.102, p<0.05 \)) of developers on the behavior of their followers performing contribution actions. More popular developers have a bigger influence on their followers to perform an action to the project. 

H1b posited that the project's social cues influence a developer's decision to act to the project positively. Meaning that more popular (Proj.Pop.) or active (Proj.Act.) projects will have a bigger influence on developers to start contributing to the project. For that we looked at the popularity of a project by measuring the actions of different developers, as well as to the activity of the project since the activity can also be created by only one individual. The popularity measure of a project is significant and positive (\( \beta = 0.101, p<0.05 \)) indicating that developers are influenced by more popular projects to start contributing. More active projects have a significant and positive effect on developers as well (\( \beta = 0.104, p<0.05 \)). This shows that developers are more likely to start contributing to more active projects. Furthermore, the project's closing time (H1c) is significant and negative as expected for contribution actions (\( \beta = -0.148, p<0.001 \)). This result indicates that for projects where the closing time of issues is

<table>
<thead>
<tr>
<th>Variable</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev.Pop.</td>
<td>-0.148***</td>
</tr>
<tr>
<td>Dev.Act.</td>
<td>0.536***</td>
</tr>
<tr>
<td>Proj.Pop</td>
<td>0.061*</td>
</tr>
<tr>
<td>Proj.Act</td>
<td>0.102*</td>
</tr>
<tr>
<td>Proj.Close</td>
<td>0.101*</td>
</tr>
<tr>
<td>Proj.Lang</td>
<td>0.104*</td>
</tr>
</tbody>
</table>

Significance Level: *p<.05, **p<.01, ***p<.001.
shorter, developers are more likely to make an action to the project.

Lastly, the project's programming language (H2a) is also positive and significant for contribution actions as shown in Table 7 ($\bar{\beta} = 0.536, p<0.001$). This means that developers are more likely to make an action to a project when the programming skills match the programming language of the project more closely. Regarding the second analysis, the types of feeds have different effects on the prediction of peripheral contributions as we hypothesized (H2b).

**Table 7. Logistic Regression analysis about Feed types**

<table>
<thead>
<tr>
<th>Variable</th>
<th>Model B (se)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>.146* (.058)</td>
</tr>
<tr>
<td>Delete</td>
<td>-.060 (.050)</td>
</tr>
<tr>
<td>Issue</td>
<td>.023 (.016)</td>
</tr>
<tr>
<td>Pull Request</td>
<td>.014 (.011)</td>
</tr>
<tr>
<td>Push</td>
<td>-.007 (.012)</td>
</tr>
<tr>
<td>Gollum (Wiki)</td>
<td>-.004 (.014)</td>
</tr>
<tr>
<td>Comment (Issue)</td>
<td>-.002 (.003)</td>
</tr>
<tr>
<td>Comment (Pull)</td>
<td>-.008 (.013)</td>
</tr>
<tr>
<td>Comment (Commit)</td>
<td>-.002 (.008)</td>
</tr>
<tr>
<td>$X^2$</td>
<td>20.27</td>
</tr>
<tr>
<td>df</td>
<td>9</td>
</tr>
<tr>
<td>p</td>
<td>.016</td>
</tr>
</tbody>
</table>

*p<.05, **p<.01

Only the amount of ‘Create Event’ type feeds is significantly and positively related to peripheral contributions (B (se) = .146 (.058), p <.05). The regression model itself is also significant at p<.05 level (p=.016).
In sum, the hypotheses about individual level stigmergic interaction in Github are all supported. In particular, the finding that the amount of feeds generated within three days before peripheral contribution significantly predicts the occurrence of peripheral contribution demonstrates the quantitative stigmergic interaction in Github. Meanwhile, selective significant association between the number of ‘Create’ event feed and peripheral contribution indicates qualitative stigmergic interaction (H2b). Taken together, these show that the use of PCSF at individual level provides action cues for peripheral contributions.

CONCLUSION

In our study, we conducted the research about the voluntary coordination patterns appearing at the online collaboration platform, Github, which has adopted peripheral contribution support features (PCSF) for the first time for OSSD platform. Through theoretical lens of stigmergy, how the individual Github user, at the same time an OSS developer, utilizes PCSF and was lead to contribute to other projects, not owned by the oneself. We also tracked the effect of those peripheral contributions triggered by stigmergic interactions on the code maintainability improvements of the OSSD projects.

By examining 2,034 projects on Github, we found that both PCSF (watch and follow) are consistently in negative association with reverse measures for code maintainability, violation total and code complexity per files. This substantiates the hypothesis that peripheral developers tend to engage with code refinement activities through PCSF. As the result shows an opposite pattern from the current belief about the influence of peripheral developers on code quality, increasing structural complexity of source codes from participations by peripheral developers, the paper further assumed the intervening role of system functions provided by Github, to induce such result. The assumption was empirically tested in the following Study 2.
With the longitudinal study of 544 sample developers for over two months, we found that the level of peripheral contribution support feature use and the frequency of making peripheral contributions is in significantly positive correlation. Both quantitative and qualitative stigmergy are also tested significant in support for the hypotheses. Not only the increase of certain cues was likely to induce peripheral contributions, but also the existence of ‘create’ event feed within the same time window before peripheral contributions is proven to be significant. Drawing on these findings, both types of stigmergic interaction as elaborated from existent research - the quantitative stigmergy, indicating the mere increase of environmental cues followed by certain type of actions from agents, and the qualitative Stigmergy, meaning the different reaction by an agent following qualitatively different types of cues – are observed to be present in Github.

Combining the results from both studies, it is evident that peripheral contributions through stigmergic interactions with PCSF are prevalent in Github, having an impact on code maintainability.

**DISCUSSION**

The results of our log data analyses show that the PCSF provided by the Github platform seem to guide individual-level project selection as well as inducing group behaviors by developers who make use of those features. Drawing on the findings obtained from the individual-level analysis, it is shown that users selectively react to the ‘Create Event,’ which signals the important changes made within a certain project by the project owner. Receiving more feeds either by watching more projects or via the increased activity level of a watching project is likely to induce peripheral contributions by the users. By coding the amount of feeds generated before users’ contributions, these findings show clear directions in the relationships, ensuring the reliability of the result.
Extant literature about OSSD has discussed extensively about how to involve peripheral developers more deeply in a certain project (Chengalur-Smith and Sidorova 2003; Crowston et al. 2012; Hars and Ou 2001), but not on how peripheral contributions by the developers themselves have impacts on OSSD projects. In our research, we demonstrate that peripheral contributors self-organize themselves by finding rightful opportunities for contributions through stigmergic interactions via PCSF, the support of information technology.

More specifically, peripheral developers tend to be affected by popularities and activeness of both projects and project owners that are represented as the number of either watchers or followers and the number of feeds generated during the observation period. This course of being affected by the mere presence or increased number of cues are quantitative stigmergy as described in previous studies.

Pertaining to qualitative stigmergy, the result implies that peripheral developers are likely to consider both types of cues signaling what contribution actions are in need at given projects and whether their expertise match the needed actions. Additional qualitative evidences for each hypothesis are presented below from the transcripts of interview data with eight Github users (average experience over 1 year). We conducted semi-structured in-depth interviews to understand how actual Github users utilize the PCSF (Watch, Follow), especially in the context of group work coordination. For more accurate recall of their experiences on Github, all of the interviewees were asked to explore and show their Github pages by logging on to their own account during the session. A monetary reward equivalent to 30 USD equivalent was provided to each interviewee. Eight users in total were interviewed; all were working as professional developers. For hypothesis 1, the interviewees described their usage of the PCSF provided by Github. It was evident that they used those features to keep a list of developers and projects in order to be able to easily keep up with whatever was
happening in the listed developers and projects.

“Whenever I have some spare time, I go to my profile page, where my Watch and Follow list is stored, in order to look for what I can do to contribute to a project or to a developer, rather than looking for it from scratch.” (p8)

“I will use the Watch feature when I’m really interested in a particular project...” (p3)

“I check the timeline to catch up with the progress of projects in which I’m interested.” (p2)

Interviewees also showed such behaviors as utilizing their dashboards to look for rightful opportunities to contribute to projects.

“In short, a feed is some sort of a catch...... Upon seeing a task that looks interesting, I can’t help taking a closer look at what is going on in that project.” (p4)

Lastly, they tend to evaluate the types of feeds differently in terms of how big an impact a type of feed implies. ‘Create Event’ feeds that can only be made by the project owners are considered more important than others for this reason.

“When watching projects, it’s those critical events, for example, when a project gets updated from version 0.1 to version 0.12, that gets me going. Events such as a few lines being changed here, or a pull request being pushed to the master, don’t interest me much. But a new release ... that's the type of event I really take an interest in.” (p2)

The role of PCSF is evident in the results, significantly predicting both individual level and project level outcomes. No difference is shown between the different types of PCSF, i.e. Watch and Follow. In sum, the use of PCSF synchronously broadcasts all actions that might interest peripheral developers. Then a user reacts to certain feeds generated, resulting in peripheral contributions. Finally, this leads to a project-level concentration of contributions,
improving the quality of project outcomes

Theoretical Implications

Our study contributes to the extant literature in several ways: First, we provide a more definitive account of the mechanisms for self-organization at OSSD project level that result from the stigmergic interactions afforded especially by Github’s PCSF, using the data gathered from real-world settings. Second, we extend the extant knowledge about collaborative OSSD by explaining both individual-level choices and project-level outcomes. Previous literature has only focused on either side of the mechanism. Moreover, our research also shed light on the role of PCSF, increasing the applicability of the research results. Third, our study is one of the first to conduct large-scale empirical analysis of the phenomenon. Our study contributes to research on the impact of introducing social computing technologies on collaboration outcomes in a naturally occurring context. Additionally, we find that greater usage of PCSF had a more positive effect on OSSD project outcomes through stigmergically gathered developers. In short, developers are able to keep abreast of a wider range of OSSD projects, widening the range of social interactions, thus resulting in the shift from lone hacker projects to a network of socially integrated projects.

Practical Implications

Based on the findings of our study, we propose design guidelines for building an open platform for online collaboration.

Support interest-based network with increased awareness

As shown in studies 1 and 2, the use of PCSF has a direct association with improved maintainability of source code qualities, and facilitates the project level patterns that help to improve outcome qualities. From the interviews with Github users, it would appear that they
use these PCSF to filter out only those projects or developers in which they are interested. This increases the chance of encountering rightful opportunities to make contributions to other projects by reducing the search cost. Moreover, supporting the synchronous broadcasting of events in interesting projects and by interested developers, as Github does, also helps followers and watchers both to catch up with the changes made and to understand the current situation. With their increased knowledge of either projects or developers, watchers and followers thereby become even more interested in them.

**Differentiating the representation of information according to its impact and relative importance**

In our analysis, one type of feed is significantly related to peripheral contributions, proving that users selectively process the information they receive. The type of feed, ‘Create Event,’ can only be generated by the project owners, since they signal of creating a repository, a branch or a tag that can only be authored by the owners. Given that, highlighting the types of information which can cause major changes or have potentially large influences, will help users to identify them more easily.
REFERENCE


Terceiro, A., Rios, L. R., and Chavez, C. 2010, September. An empirical study on the structural complexity introduced by core and peripheral developers in free software projects. In *Software Engineering (SBES), 2010 Brazilian Symposium on* (pp. 21-29). IEEE.


Top languages on github. https://github.com/languages


