PAPER

Peer Review Social Network (PeRSoN) in Open Source Projects

Xin YANG[†a)], ***Nonmember***, **Norihiro YOSHIDA**[††b)], ***Member***, **Raula GAIKOVINA KULA**[†††c)], ***Nonmember***, ***and*** **Hajimu IIDA**[†d)], ***Member***

SUMMARY Software peer review is regarded as one of the most important approaches to preserving software quality. Due to the distributed collaborations in Open Source Software (OSS) development, the review techniques and processes conducted in OSS environment differ from the traditional review method that based on formal face-to-face meetings. Unlike other related works, this study investigates peer review processes of OSS projects from the social perspective: communication and interaction in peer review by using social network analysis (SNA). Moreover, the relationship between peer review contributors and their activities is studied. We propose an approach to evaluating contributors' activeness and social relationship using SNA named Peer Review Social Network (PeRSoN). We evaluate our approach by empirical case study, 326,286 review comments and 1,745 contributors from three representative industrial OSS projects have been extracted and analyzed. The results indicate that the social network structure influences the realistic activeness of contributors significantly. Based on the results, we suggest our approach can support project leaders in assigning review tasks, appointing reviewers and other activities to improve current software processes.

key words: *peer review, Open Source Software, social network analysis*

1. Introduction

Software peer review refers to the code inspections by developers, rather than the authors themselves. It can be regarded as one of the most important activities to guarantee the quality of software products [1], [2]. Software projects adopt peer review for two principal reasons: reducing defects and saving development cost. The traditional code review (a.k.a code inspection) was established 30 years ago. Code inspection requires experienced reviewers meet and discuss the source code written by other developers [3], [4]. Peer review is not only an indicator of the quality of source code but also signifies a healthy organization. Stable and growing development communities always hope the experienced developers could share their knowledge with new members. For example, developers share knowledge when they perform code review activities [5].

Recently, the peer review process of Open Source Software (OSS) varies from the traditional industrial setting. One main reason is most OSS projects are geographically distributed, whereas traditional industry projects take the form of gathering developers in the same room. The modern OSS peer review applies a broadcasting method to announce code review tasks and locate appropriate reviewers. However, we find that only a few of studies have focused on the code review process.

Our objective is establishing a model of OSS code review community and a set of quantitative measures to describe the code review process from both technical metrics and non-technical metrics. Another motivation comes from the importance of the human factor in software development. The human factor has been researched from the diversity of different cultures and the rise of globally distributed projects [6].

To understand how contributors work and communicate together, we need to investigate the structure of code review community. In this work, we present PeRSoN, which is a construction of social networks from peer review activities, which is based on our previous work [7], [8]. We categorize contributors into different role groups based on their authorities, and we define the review activity as any contribution in the review process. We use the PeRSoN to evaluate two research questions: **RQ1.** *Which contributor role is the most important in the peer review community?* and **RQ2.***What is the relationship between contributors' activities and their network position?*.

We applied PeRSoN to three large-scale OSS projects: Android Open Source Project (AOSP), Qt and OpenStack by case study. The results of analysis addressed our research questions and gave hints about the relationships among OSS peer review contributor roles, their activities and their network structure. Our main findings can be summarized as two points. First, the contributors who have the verification authority are the most important (most central) role in the review community (see Table 8). Second, a strong linear relationship exists between activities of the contributors who have verification authority and their network positions (see Table 9). The main contribution of this work can be summarized as follows:

- We established a novel approach to extract data for social network analysis from our raw dataset that mined

Manuscript received July 2, 2015.
Manuscript revised October 21, 2015.
Manuscript publicized November 27, 2015.

[†]The authors are with Graduate School of Information Science, NAIST, Ikoma-shi, 630–0192 Japan.
[††]The author is with Graduate School of Information Science, Nagoya University, Nagoya-shi, 464–8601 Japan.
[†††]The author is with Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871 Japan.
a) E-mail: kin-y@is.naist.jp
b) E-mail: yoshida@ertl.jp
c) E-mail: raula-k@ist.osaka-u.ac.jp
d) E-mail: iida@itc.naist.jp

DOI: 10.1587/transinf.2015EDP7261

Table 1 Contributor roles in the peer review process.

Role	Definition
Contributor	A *contributor* represents a participant who takes part in the code review process.
Author	An *author* represents the contributor who submits a patchset and the owner of the review report belonged to this patchset.
Reviewer	A *reviewer* represents the contributor who reviews the patchset to find defect and bug inside.
Committer	A *committer* represents the contributor who has the authority to commit the patchset into the code repository.
Approver	A *approver* is an experienced reviewer who reviews and approves the patchsets by checking whether the patchset changes follows the best practices that have been established by the project fits the project's stated purpose and the existing architecture.
Verifier	A *verifier* handles building, testing and verifying the patchset and decides whether it is suitable for merging into the source code. In many OSS projects, *verifiers* can be automated tools.

from code review repository.

- This study investigated the importance of OSS peer review contributor roles and their review activities by using social network analysis.
- We could evaluate the performance of review contributors from their personal activeness, behaviors, and social relationship using our approach.

The article is organized as follows. Section 2 describes code review in software engineering as our research background. In Sect. 3, we present PeRSoN as our approach that studies code review process from the social aspect. We also detail the steps of the approach and the metrics we used. Section 4 describes our experiment design, research questions, data processing, and results. Section 5 discuss the implications and validity based on our findings. In Sect. 6, we lay out those important related work and the relationship with this study. Section 7 describes the conclusion of this paper and the future plan of our study.

2. Background

In the past decade, OSS as a dynamic software development manner has been adopted by many software organizations. Unlike traditional industry projects, OSS code is accessible for patch contributions. With the rise of OSS projects, regular code inspection also has been modified to cater for OSS projects development. It is not feasible for developers to always have communication or collaboration directly. As a result, it is hard to search appropriate reviewers to perform the high-quality review because of lacking the information of contributors. Based on these difficulties, OSS peer review applies a broadcasting method to announce and search the appropriate reviewer for particular source code [9].

To facilitate these OSS peer review style, many software projects even industrial projects have adopted peer review tools. These companies perform code review instead of face-to-face inspection meeting (e.g., Google uses Gerrit[†] in AOSP, and Microsoft uses CodeFlow as their review tools[††]). Many large-scale OSS projects have adopted web-based peer review. They also have claimed that peer review is an important quality assurance technique in their projects [10]–[12]. During the early time of OSS projects, most OSS projects assign review task and retrieve feedback using mailing list [13]. Currently, OSS projects adopt light-weight peer review tools instead of using mailing-list. Applying peer review tools, the status of code review can be tracked easily and review contributors can participate in peer review freely. The mechanism being used in OSS projects for peer review provides benefits to communities like sharing knowledge and experience among contributors.

3. Peer Review Social Network (PeRSoN)

3.1 Process and Networks

Peer Review Process.

The *peer review process* represents the process to perform code review related activities by code authors and reviewers to guarantee the reliable software. In our study, contribution activity refers to the activities carried out by contributors, which means both authors of code change and reviewers. The primary contribution activities include submission of new patchset, revision of patchset, code review, review approval, review verification, review discussion and others. Based on the activities records in code review system, we calculate the number of activities then separate contributors into different roles by their activities. Also, we introduce the definitions of contributor roles that will be used throughout this paper in Table 1.

Figure 1 is a practical example of review process. This figure represents the review process of Change #17768 of AOSP[†††]. The first step of review is commit phase that author submit new patchset and the system will notify reviewers by email. Then in review phase, reviewers will perform reviews based on this patchset (Change #17768). Every contributor can perform review in a project, but only those reviewers who have the authority of Approval or Verification (Always chosen from core members, or experienced and active reviewers) can determine whether this change can be merged (Some projects also use bots to build and test the patchset as Verifiers). In this case, three different reviewers have performed reviews. Mark Gross has reviewed but still need someone with approval authority to approve it. As a result, David Turner has approved this patchset, and JBQ (Jean-Baptiste Queru) has verified it. The final step is integration phase, system or particular core members will merge the approved and verified patchset into project code repository. In this example, JBQ has merged this patchset to the

[†]https://code.google.com/p/gerrit/

[††]http://goo.gl/5zk0wF

[†††]https://goo.gl/8uFZiv

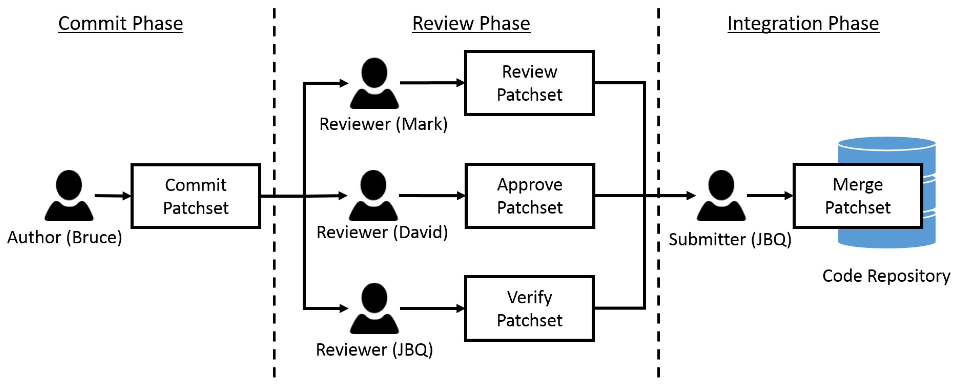

Fig. 1 The review process of change #17768 in AOSP

repository.

PeRSoN Definition.

Our approach uses social network described as a graph network called PeRSoN. PeRSoN is a social network constructed by peer review dataset, which a vertex represents a review contributor and an edge represents a review activity happened between two review contributors (e.g., in Fig. 1, Mark, David, and JBQ performed code review for Bruce's patchset, and then they left comments as feedback to everyone who have contributed in these reviews). In this study, we define the network model as a undirected and weighted network.

We assume that a contributor c_i has a set of reviews R_{ci}. A review r in a patch set has a set of contributors including both authors and reviewers $\{c_1, c_2, \ldots, c_n\}$. A PeRSoN edge e is formed when two contributors (e.g., c_i and c_j) are members of the same review. Formally, $e(c_i, c_j)$ exists if $c_i \in R_{ci}$ and $c_j \in R_{cj}$ where $R_{ci} \cap R_{cj} \neq \emptyset$.

Network Measures

Our model can be used to describe more complex distribution characteristics of the contributors. In our approach, we evaluate the reviewers network by using the three standard centrality measures of Degree, Betweenness and Closeness based on the definitions from Freeman [14] as below:

- **Degree Centrality.**
  Degree Centrality indicates the number of edges that a vertex has, A vertex (contributor) is defined as c_k and $a(c_i, c_k) = 1$ if c_i and c_k are connected, otherwise 0. Degree Centrality of c_k is defined as $C_D(c_k)$:

$$C_D(c_k) = \sum_{i=1}^{n} a(c_i, c_k)$$

- **Betweenness Centrality.**
  Betweenness Centrality of a given vertex indicates the number of shortest paths from all vertices to all other vertices that pass through this vertex. We define g_{ij} = the number of edges from vertex c_i to vertex c_j, and $g_{ij}(c_k)$ = the number of edges from vertex c_i to vertex c_j that passing through c_k. Then calculate the probability $b_{ij} = \dfrac{g_{ij}(c_k)}{g_{ij}}$. Betweenness Centrality of c_k is defined as $C_B(c_k)$:

$$C_B(c_k) = \sum_{i<j}^{n}\sum b_{ij}(c_k)$$

- **Closeness Centrality.**
  Closeness Centrality of one vertex indicates the inverse of its *farness*. *Farness* indicates the sum of the distance between this vertex to all other vertices. We define $d(c_i, c_k)$ = the distance (number of edges) linking c_i and c_k. Then Closeness Centrality of c_k is defined as $C_C(c_k)^{-1}$:

$$C_C(c_k)^{-1} = \sum_{i=1}^{n} d(c_i, c_k)$$

Table 2 Centrality measures and social implication.

Centrality Measures	Social Implication
Degree	Activity
Betweenness	Control
Closeness	Independence

Freeman also suggested that each centrality measures have social implications as shown in Table 2 [14]. First, Degree centrality implies activity degree, a vertex with a high degree in the network suggests this person should be active and enthusiastic. Second, betweenness centrality implies control. A vertex with high betweenness centrality acts as a bridge among other vertices in social networks. Third, closeness centrality implies independence of a vertex. A vertex with low closeness represents that this people is independent and far away from all other people.

Figure 2 is an example of how we evaluate the performance of contributors from their human factor and social aspect using our approach. Here we simplified the networks by ignoring the weight of edges. However, in the practical experiment, we calculate the weight of edges. The vertices represent contributors and edges represent the review activities between contributors. We perform social network analysis for this network by calculating the centrality measures for each contributor. After we calculated the centrality distribution, some observation can be summarized as follows:

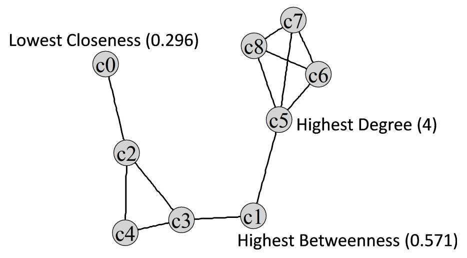

Fig. 2 An example of contributors evaluation using PeRSoN

c5 has the highest degree in this network, which indicated he/she is the most active contributors; **c1** has the highest betweenness and it represents that he/she work as a bridge in the community, and he/she could be a potential bottleneck, which may cause problems to the process; **c0** has the lowest closeness which means he/she contribute as individual or rarely collaborate with others.

Due to the importance of social aspect study on peer review, we introduce the following research questions that we mentioned in Sect. 1.

RQ1 *Which contributor role is the most important in the peer review community?*

RQ2 *What is the relationship between contributors' activities and their network position?*

As we introduced in Sect. 3.1, social network analysis provides the data constructed from the social relationship between people. We want to investigate who are the most important contributors and is there a relationship between review authority and social network position in peer review community (**RQ1**). Moreover, whether realistic activities of contributors correlate with their social network position (**RQ2**). To address these research questions, we introduce the details of our approach in next subsection.

3.2 Approach

We now introduce the main steps to our approach as three steps; 1.) Dataset mining, 2.) Network generation and role classification and 3.) Metrics analysis.

Dataset Mining. We used a dataset from our previous work [15][†].

To extract raw peer review dataset we apply Gerrit official API to obtain raw dataset. Gerrit code review system provides an REST-like API for users[††]. Users can access and gain the raw dataset through HTTP for their use. The raw dataset is stored in the form of JSON files. For each review in peer review system, the JSON dataset has a unique ChangeID. The JSON dataset includes the following features: The review information with reviewers' comments, the updates history of patches, and the details of when and how they have been merged into the source code or be abandoned. We created scripts using Python to download the raw dataset by using official API. The primary functions of the scripts are extracting useful data out from raw dataset and store this useful refined dataset into a database. In detail, we extracted the data from the comments history of each review reports in raw dataset. Some important data we used is described as Table 3. In next paragraph, we will introduce how we use these data to create the social network and investigate the potential relationship between contributors.

Table 3 Field and definition in comment history.

Field	Definition
reviewId	ID of the review report.
authorId	The author who created the patch.
reviewerId	The contributors who reviewed this patch.
lastUpdate	The timestamp of last update in this review.
writtenOn	the time when this review comment is created.
message	the content of this review comment.

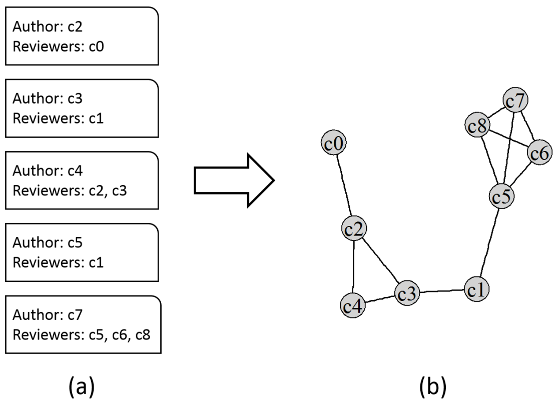

Fig. 3 An illustrative example of network generation

Network Generation and Role Classification. We regard the review as a communication channel between code submitters and reviewers. Every review is a time of discussion or feedback between patch authors and reviewers. Using the history dataset of each review, we extract the connection of all the contributors in peer review communities. The steps how we extract the review connection is introduced as follows: First, it was assumed that all the review related activities of the contributors in a review process are recorded in the comments on the review reports, such as an author has submitted a new patch or another reviewer has reviewed a patch and given his/her comments based on the patch change. This step is shown in Fig. 3 (a).

Second, to form the network, all contributors participating in the same review were connected, which indicates the contributors who work in the same review have used the same communication channel as a team work, as shown in Fig. 3 (b).

Finally, we perform role classification by contributors' activities from their comments (e.g., An approval comment from a contributor can be regarded that this contributor has the authority to approve a patch). We separate contributors by their different activities and authorities. More details about classification will be introduced in Sect. 4.3.

Metrics Analysis. The analysis of this study in-

[†]The dataset is available to download at http://sdlab.naist.jp/reviewmining/

[††]https://goo.gl/PsKwFx

cludes analysis of the network metrics. Specifically, we use the standard centrality measures: degree, betweenness and closeness to obtain contributors' network position. Then we compare the centralities between different contributor role groups to investigate which group is most important. We then use statistical analysis to find correlations between the contributors centrality measures and their activities.

4. Empirical Case Study

We evaluate our approach by applying PeRSoN network to three real world OSS projects that using Gerrit as code review system. These three projects require strict code review mechanism, which means every code change must be reviewed first, then be commit to project code repository. Another reason for choosing these projects is that they adopt code review system instead of a mailing list, which bring more convenience when collecting the data.

4.1 Experiment Setup

At first, we studied the peer review process using Gerrit environment in three projects. We found that Gerrit system manages contributors by providing different authorities. For example, normal contributors (not core members) can review code, but they have no permission to make the final decision of change. While core members can both review and judge a change. Because high-authority contributors have more responsibility to the quality of change, we first hypothesize the high-authority contributors are more important to the review system as follow.

H1 *The contributors who have the highest authorities in code review play the most important role in code review community.*

We address **RQ1** by accepting **H1**, and we introduce how we compare the difference between high review authorities contributors and other people in statistical analysis in Sect. 4.4. Complementary to **H1**, we add a **H0** as a null hypothesis that indicates *the contributors who have the highest review authorities and other contributors are from the same distribution*.

We address **RQ2** by calculating the correlation of contributors' activities and their network positions for different contributor groups, which separated by the roles. We use degree, closeness and betweenness to measure contributors' network positions, and then compare their activities.

4.2 PeRSoN Generation

Datasets. AOSP (Android Open Source Projects) is an industrial open source project that developing software products for mobile device[†]. OpenStack project provides cloud services platform[††]. Only a few of verifiers are human, but the primary verification works are done by Continuous Integration (CI) tool. Qt Project is an application framework that mainly use to develop graphical user interface[†††]. Qt uses the term *Sanity Review* instead of *Verify* in its Gerrit system.

Table 4 Basic information of AOSP, OpenStack and Qt.

	AOSP	OpenStack	Qt
Period	2009/01~ 2011/06	2011/07~ 2012/06	2011/07~ 2012/06
# of Comments	42449	64793	219044
# of Contributors	1086	426	620
# of Reviewers	451	165	207
# of Approvers	99	86	201
# of Verifiers	111	29	117
# of Vertices	808	379	558
# of Edges	15429	55301	150017

Several necessary data in change information table is needed to generate PeRSoN. e.g., *Change-Id* represents the review report identification, a unique Hash code generated by Git. *Uploaded* and *Updated* represents the timestamps when this report created and when is the latest update.

When a patchset is under review, the author who has submitted this patchset could still update and fix this patchset. A review report could include more than one patchsets if the author upload revisions. In Gerrit, contributors can check the status for current patchset such as who has reviewed or who will verify it. As a result, the comments record all the reviewers who take part in the review are important. In this study, PeRSoN was generated by R[††††] and igraph package. The statistical analysis of this study was performed by R.

Generated Networks. Applied the network generating method mentioned in 3.2 and in Fig. 3, the social networks have been generated from these projects separately. The basic information of these social networks is shown as Table 4. In this study, the dataset of AOSP covers 2.5 years but OpenStack and Qt have only one year. Because projects have different periods, the review dataset was separated into smaller samples. We split the dataset by one month, three months and six months to observe how the networks of AOSP evolved through time.

In experiment period, we found that results of every six months are most obvious and evident, and then we decided to divide dataset by every six months. We also generate PeRSoN by the whole dataset, and the number of vertices and edges are shown as Table 4. We include all the participants in our networks as vertices, and all the review comments between contributors as edges. We can found that each project has a different size in networks.

From the information in Table 4, it is easy to find out the common reviewers group is larger than the verifiers group. This observation complies with the Onion Model of OSS development [16] that considers the core members as the most important and the smallest group. It can be supposed that the smallest group of peer review, the verifiers should also play the most important roles in code review

[†]http://source.android.com/
[††]http://www.openstack.org/
[†††]http://qt-project.org/
[††††]http://www.r-project.org/

process. To observe results easily, we separated contributors into different roles from the activities of their review comments that can be regarded as the history of their activities.

4.3 Contributor Roles Classifications

We classified the roles of the contributors by the activities they performed during the code review. We extracted contributors' activities by mining their review comments. Administrators or team leaders in a project can set the review rules, such as evaluating code changes by a different score or approving code change with certain review authorities. Each project may have their rules and own authorities assignment. In this study, we investigated the review rules of three projects to extract review activities from the comment history. Moreover, we classify contributors roles more detail to gain a better understanding of the relationship between contributor groups.

As a result, a new classification method using activities are proposed and applied. Based on our previous work related to contributor classifications [7], [17], [18], we apply a classification method based on the activity types. Each contributor is labeled based on their roles, such as V as Verification, A as Approval, C as Code-Review. We investigate contributors from all seven combinations of V, A, C, VA, VC, AC, and VAC. For example, If one contributor has activities records of review and verification, he will be noted as VC. A contributor only contributed by approval without other activities, he will be noted as A. If a contributor has Verification activities also did code review and approved patchsets, who takes part in everything can be noted as VAC.

4.4 Results

RQ1: Most Important Contributors. We address **RQ1** by comparing the distributions of contributors' network positions. In peer review process, verification should be the highest authority in the peer review as it was the last stage of change before its merge or abandoned. Moreover, from Freeman's study, we know in a network structure, the centrality measures imply the importance of each node. As a result, the most important contributors indicate they have the highest centralities. In this study, three standard centrality measures have been used, and all of them have different social implications. From the observations of our previous work based on AOSP, which we separated contributor roles into *Verifier* and *Non-Verifier*, we created cumulative graphs of contributors' frequency, and we observed: Verifiers' degree and betweenness increased over time while non-verifiers did not change too much. Verifiers have a relatively greater degree and betweenness than non-verifiers. However, we did not find any relationship between Verifiers and non-verifiers in terms of closeness.

Also, we separate contributors by more detail way as we mentioned in Sect. 4.1. The classification results like follows: AOSP has four different contributor roles as VAC, VC, AC, and C. From the observations we found: VAC group has greater median value than other contributors' groups, and the interquartile range of VAC group is wider than other groups in terms of degree and betweenness but not in closeness (see Table 5). OpenStack has three groups: VAC, AC, and C. The results show that the VAC group has greater median than other groups, and same as AOSP, the interquartile range of VAC group is wider than other groups in degree and betweenness but not in closeness (see Table 6). Qt has four groups: VAC, AC, A and C. The observation shows, Qt's VAC group has greater median value than other groups in degree and closeness but not obviously in betweenness, and the interquartile range of VAC group is wider than any other group (see Table 7).

To compare the different centrality distributions among VAC and other roles, we applied a Wilcoxon-Mann-Whitney test to evaluate **H1** [19]. We adopt Wilcoxon-Mann-Whitney because that we found the population has only one

Table 5 Summary of distributions for VAC, VC, AC, and C contributors in AOSP

		VAC	VC	AC	C
Degree	**Min**	5	4	1	1
	1st Qu.	46	15	7	3
	Median	108	29	24	6
	Mean	251	56	25	13
	3rd Qu.	293	49	31	12
	Max	3349	439	79	779
Betweenness	**Min**	0.0000	0.0000	0.0000	0.0000
	1st Qu.	0.0002	0.0000	0.0000	0.0000
	Median	0.0036	0.0000	0.0000	0.0000
	Mean	0.0176	0.0017	0.0006	0.0002
	3rd Qu.	0.0111	0.0024	0.0001	0.0000
	Max	0.6509	0.0155	0.0065	0.0440
Closeness	**Min**	0.3417	0.2837	0.3194	0.0024
	1st Qu.	0.4274	0.3438	0.4089	0.3328
	Median	0.4474	0.4099	0.4199	0.4033
	Mean	0.4455	0.3884	0.4094	0.3708
	3rd Qu.	0.4683	0.4263	0.4226	0.4121
	Max	0.6793	0.4738	0.4473	0.5176

Table 6 Summary of distributions for VAC, AC, and C contributors in OpenStack

		VAC	AC	C
Degree	**Min**	263	9	1
	1st Qu.	995	106	11
	Median	1497	277	29
	Mean	2665	429	69
	3rd Qu.	2604	650	78
	Max	20050	2273	1016
Betweenness	**Min**	0.0000	0.0000	0.0000
	1st Qu.	0.0006	0.0000	0.0000
	Median	0.0033	0.0002	0.0000
	Mean	0.0337	0.0011	0.0000
	3rd Qu.	0.0128	0.0012	0.0000
	Max	0.6290	0.0139	0.0060
Closeness	**Min**	0.5108	0.4725	0.3510
	1st Qu.	0.5602	0.5099	0.4809
	Median	0.5753	0.5232	0.4993
	Mean	0.5966	0.5306	0.4966
	3rd Qu.	0.6082	0.5498	0.5101
	Max	0.8873	0.6087	0.5745

Table 8 Comparison of three centrality measures distributions for different role groups in AOSP, OpenStack and Qt. V represents Verification, a represents approval, c represents Code-Review, and the following number indicate the amount of reviewers in this group (e.g., VC (26) represents that 26 contributors have performed verification and also Code-Review in the current project). Each row presents the hypothesis being tested (e.g., VAC ~ VC), the one-side alternative hypothesis (i.e., >) and the p-value (with (*) indicate that the alternative hypothesis is accepted.)

Projects	Comparison	Degree	Closeness	Betweenness
AOSP	VAC (80) ~ VC (26)	>, p = 5.091e - 06, (*)	>, p = 8.751e - 07, (*)	>, p = 2.416e - 05, (*)
	VAC (80)~ AC (16)	>, p = 9.292e - 07, (*)	>, p = 2.897e - 05, (*)	>, p = 2.059e - 05, (*)
	VAC (80)~ C (451)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)
OpenStack	VAC (26) ~ AC (69)	>, p = 9.292e - 07, (*)	>, p = 2.897e - 05, (*)	>, p = 2.059e - 05, (*)
	VAC (26) ~ C (165)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)
Qt	VAC (116) ~ AC (72)	>, p = 1.447e - 14, (*)	>, p = 1.536e - 11, (*)	>, p = 1.096e - 09, (*)
	VAC (116)~ A (13)	>, p = 1.939e - 08, (*)	>, p = 2.356e - 08, (*)	>, p = 2.645e - 07, (*)
	VAC (116)~ C (207)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)	>, p < 2.2e - 16, (*)

Table 7 Summary of distributions for VAC, VC, A, and C contributors in Qt

		VAC	VC	AC	C
Degree	**Min**	8	7	2	2
	1st Qu.	363	48	6	6
	Median	776	124	36	20
	Mean	1204	314	45	70
	3rd Qu.	1420	346	82	68
	Max	12010	4194	124	764
Betweenness	**Min**	0.0000	0.0000	0.0000	0.0000
	1st Qu.	0.0000	0.0000	0.0000	0.0000
	Median	0.0000	0.0000	0.0000	0.0000
	Mean	0.0002	0.0000	0.0000	0.0000
	3rd Qu.	0.0000	0.0000	0.0000	0.0000
	Max	0.0053	0.0006	0.0000	0.0001
Closeness	**Min**	0.5014	0.5018	0.5004	0.5009
	1st Qu.	0.5119	0.5053	0.5018	0.5023
	Median	0.5213	0.5082	0.5036	0.5036
	Mean	0.5283	0.5123	0.5039	0.5061
	3rd Qu.	0.5369	0.5134	0.5059	0.5071
	Max	0.6545	0.5919	0.5073	0.5351

variable (each centrality) but with more than two levels (different roles), and we assume that each role group have independent centrality distribution and not affect each other. All three project have tested by comparing the VAC role with other groups existing in each project. VAC role with VC, AC and C has proved in AOSP. The null hypothesis **H0** that related to **H1** is that VAC and other roles come from the same distribution. **H0** can be rejected, and **H1** can be accepted if VAC > VC, VAC > AC and VAC > C, the p-value of all comparison is below the significant threshold of 0.05. The results of p-value are given in Table 8 are all below 0.05. The null hypothesis is rejected, and the one-sided alternative hypothesis is accepted, the true location shift is greater than 0. For AOSP, OpenStack and Qt, The most active Verifier (VAC) have significantly higher centrality than other contributors.

As mentioned above, **RQ1** can be addressed that the most active Verifier (VAC) are the most important (central) role in the review process.

RQ2: Activities and Network Position. We address **RQ2** by calculating the correlation of contributors' activities and their network positions. We use Spearman because we take the measurements from ordinal scales, while Pearson correlation is more appropriate for the measurement taken from interval scale. The correlations between the activities of different contributors and their centrality measures in three projects have been calculated. We calculated the correlation for each role group, but we only found a strong relationship in VAC contributors. The results in Table 9 shows that in OpenStack and Qt, activities of VAC have a strong linear relationship to all centralities. In AOSP, activities of VAC have a strong linear relationship between degree and betweenness, but not in closeness.

Table 9 Correlation (spearman) of VAC activity and centrality measure.

Projects	Degree	Betweenness	Closeness
AOSP	0.952	0.789	0.485
OpenStack	0.964	0.992	0.868
Qt	0.953	0.884	0.795

As a result, we addressed **RQ2** by analyzing contributors' activities and their centrality measures that indicate their network position. We found that most active verifiers (VAC), the relationship between their activities and their network position had a strong positive correlation. Except closeness centrality is unusual in AOSP, all the results show that a strong positive correlation exists between with centralities and contributors' activities. However, we did not found a strong relationship for other contributors' role groups.

5. Discussion

5.1 Implications

The results of the case study show that the network metrics (centrality) can be used to provide useful information about users roles based on their review activities. In RQ1, we find that verifiers and not the approvers are the most important roles of a review process. In RQ2, we find that again contributors that are verifiers (VAC) have a significant correlation with all other network measures.

To further understand the reviewer roles, additionally studied the evolution of role types and network positions over time. As depicted in Fig. 4, the relationship between the three centrality measures is compared, to estimate correlation coefficients over time. This figure shows that strong positive correlation exists between verifiers' betweenness

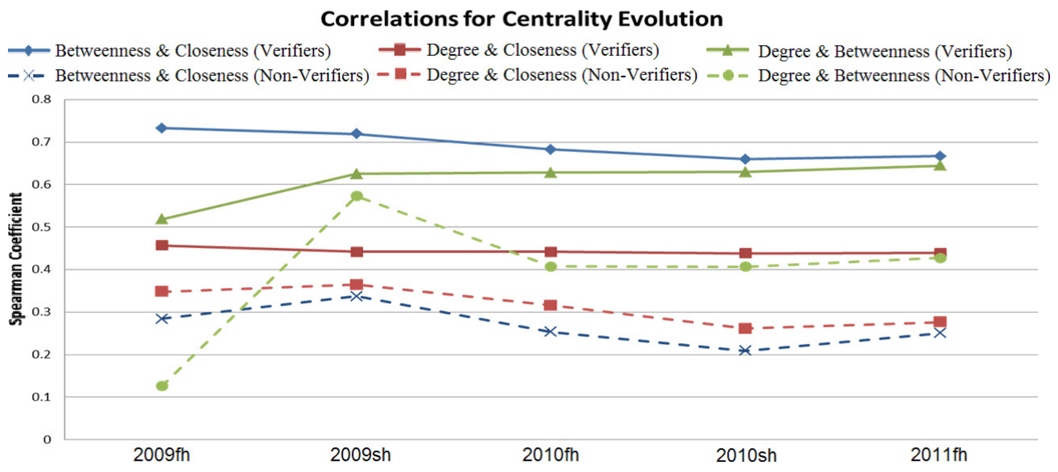

Fig. 4 Correlations for three centrality measures evolution.

and closeness, and also between their degree and betweenness. We used this results to categorizing several exception cases of contributors' centralities. Based on the positive correlation between verifiers' betweenness and closeness, it is impossible to find a verifier has high betweenness, but low closeness or a verifier has low betweenness but high closeness. Exception cases have been analyzed in which:

- A verifier had high-degree and low-closeness. A verifier has high-degree means he or she performed more activities than other people, and the low-closeness means he or she is far away from the network center.
- A verifier had low-degree and high-closeness. A verifier has low-degree means he or she performed few activities, and the high-closeness means he or she is close to the network center.

We manually checked dataset to find special cases we described above. We found a core member from AOSP, who has a high degree and a low closeness (comparing with median value). This contributor has contributed a lot in Approvals and Verifications but participated only in *kernel/common* project. While we found another maintainer in AOSP who has low degree but high closeness, we investigated that this contributor contributed very limited times but participated in many projects (e.g., *platform/build*, *platform/external/qemu*, *platform/external/clearsilver*, and *platform/system/core*).

In addition, the correlation results (see Fig. 4) and the exception cases to create Table 10 were combined. From which the following observations can be made: First, verifiers with high-closeness (close to the network center) may have high-betweenness (more control). Second, verifiers with high-degree (more contribution) may have high-betweenness (more control). However, results show that no significant correlation exists between verifiers' degree (contribution activity) and closeness (network position). Finally, from these findings, several suggestions for the exception cases can be provided:

- Verifiers with high-degree and low-closeness may be experts in specialized fields because they perform more activity and have few connections with other members or work teams. Therefore, if they result to be specialists, the important review requests can be suggested to send to them.
- Verifiers with low-degree and high-closeness may contribute less; however, being close to the network center, they may represent key figures tied to many other people. So, verifiers as high authorities can be suggested not to remove them or change their roles thoughtlessly.

Table 10 Exception cases for verifiers.

	Low Degree	Low Closeness	Low Betweenness
High Degree	—	Active / Far away	—
High Closeness	Inactive / Central	—	—
High Betweenness	—	—	—

Based on the analysis of detecting particular cases, certain contributors can be found in their different behaviors.

5.2 Threats to Validity

This study researched three large-scale OSS projects and proved that the verifiers in peer review process are the most important contributors. We discuss the limitations of this study as the following:

- **Dataset Period**. Modern peer review process using code review tools is a new technique for OSS projects, different from many bug tracking systems that have been used for more than ten years. In this study, review dataset was extracted from three projects, the longest period (AOSP) are still less than three years. Because AOSP server has shut down for six months, the data structures have been changed after that. As a result, our study based on the dataset has relatively short term period.

- **Community Size**. The community size can be regarded as an important factor in the project's development. Centrality measures are dependent on network size, which is presumably changing over time. The three projects have the different size that may affect the analysis of results. This approach needs to be evaluated by more projects with the different size.
- **Measures of SNA**. Due to our limited knowledge, in this study centrality measures were introduced into the study. More measures need to be introduced such as component, k-core, in-degree/out-degree, modularity. Also, the relationship between actors and time phases can be improved to analyze the network evolution in many different situations.

6. Related Work

Prior work related to this study could be divided into two aspects: Studies on OSS community and OSS peer review; Studies on the social aspect of software engineering. We provide these related work as following two parts.

Raymond referred to the different structures and processes of industry software and OSS as Cathedral and Bazaar [20]. Rigby et al. examined Apache Server Project for two techniques and created several metrics similar to traditional inspection experiments to find an efficient and effective OSS review technique [13]. Rigby et al. also have studied the broadcast nature of OSS peer review, which is totally different with traditional method [9]. Balachandran suggested use review-bot to reduce human effort and improve review quality [21].

Bird et al. extracted and studied the potential structure for latent sub-communities in OSS projects using SNA [22]. Zanetti et al. adopted SNA to predict bugs into valid and invalid as the bug triage approach of OSS projects [23]. Kwak et al. have studied social networks based on social networking media such as Twitter [24].

The main difference between our study and related works above is we study code review from social aspect while traditional study are mainly from the technical perspective only. Moreover, based on the prior studies that studied the human and social aspect of software engineering, we found the value and importance to perform study from human and social perspective.

7. Conclusion and Future Work

The motivation of using the approach of Social Network Analysis to research OSS peer review process comes from the distributed construction of OSS community and human factors in software development. OSS projects, especially industry-leaded OSS projects need developers to contribute enthusiastically. As proposed human factor should affect OSS review process, SNA approach was applied into this case study has researched three OSS projects. Then reviewers can be classified into several role groups with significant differences. The results show there is a strong correlation between the activities of most important contributors and their network positions. Network measures distributions of contributors can be used for evaluating contributors' activeness. For example, project managements can identify contributors who are enthusiastic but in a specialized field, and contributors who are in important network position but unenthusiastic.

In our future plan, we plan to apply our approach to more software projects. Since different projects have different review processes and different review techniques, we will apply this social network based approach to other projects to prove the usefulness of our approach. We believe our approach will identify the main factors influencing the review communities, and what are the main differences between the review techniques and processes.

- We plan to establish a set of metrics that could measure the activeness and social relationship of each review contributor in a review community, from both the social aspect and technical aspect. We could investigate the experienced and active contributors and assign them to the important task and higher authority in certain fields.
- In order to establish healthy communities with the high-productivity and good relationship, we plan to investigate more networks from different projects. Comparing the difference between network between projects will give hints about the different patterns of review networks. Based on these patterns, we can suggest what kind of network structures are more suitable for certain cases.
- All of this analysis could help to detect the weakness or potential risk in the development process. In the end, the results could help project managements to monitor the communication and collaboration networks among developers to control the quality of the software product from human and social aspects.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers 26730036.

References

[1] A.F. Ackerman, L.S. Buchwald, and F.H. Lewski, "Software inspections: an effective verification process," IEEE Software, vol.6, no.3, pp.31–36, 1989.

[2] A.F. Ackerman, P.J. Fowler, and R.G. Ebenau, "Software inspections and the industrial production of software," Proc. A Symposium on Software Validation: Inspection-Testing-Verification-Alternatives, pp.13–40, 1984.

[3] M.E. Fagan, "Advances in software inspections," IEEE Transactions on Software Engineering, vol.SE-12, no.7, pp.744–751, 1986.

[4] M.E. Fagan, "Design and code inspections to reduce errors in program development," IBM Systems Journal, vol.15, no.3,

pp.182–211, 1976.
[5] P.C. Rigby and C. Bird, "Convergent contemporary software peer review practices," Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp.202–212, 2013.
[6] D.E. Damian and D. Zowghi, "An insight into the interplay between culture, conflict and distance in globally distributed requirements negotiations," Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003, pp.10–pp, 2003.
[7] X. Yang, R.G. Kula, C.C.A. Erika, N. Yoshida, K. Hamasaki, K. Fujiwara, and H. Iida, "Understanding oss peer review roles in peer review social network (person)," Proc. APSEC 2012, pp.709–712, 2012.
[8] X. Yang, "Social network analysis in open source software peer review," Proc. FSE 2014, pp.820–822, 2014.
[9] P.C. Rigby and M.-A. Storey, "Understanding broadcast based peer review on open source software projects," Proc. ICSE 2011, pp.541–550, 2011.
[10] A. Mockus, R.T. Fielding, and J. Herbsleb, "A case study of open source software development: the apache server," Proc. ICSE 2000, pp.263–272, 2000.
[11] S. Breu, R. Premraj, J. Sillito, and T. Zimmermann, "Information needs in bug reports: improving cooperation between developers and users," Proceedings of the 2010 ACM conference on Computer supported cooperative work, pp.301–310, 2010.
[12] G. Jeong, S. Kim, T. Zimmermann, and K. Yi, "Improving code review by predicting reviewers and acceptance of patches," Research on Software Analysis for Error-free Computing Center Tech-Memo (ROSAEC MEMO 2009-006), 2009.
[13] P.C. Rigby, D.M. German, and M.A. Storey, "Open source software peer review practices: a case study of the apache server," Proc. ICSE 2008, pp.541–550, 2008.
[14] L.C. Freeman, "Centrality in social networks conceptual clarification," Social Networks, vol.1, no.3, pp.215–239, 1978-1979.
[15] K. Hamasaki, R.G. Kula, N. Yoshida, A.E.C. Cruz, K. Fujiwara, and H. Iida, "Who does what during a code review? datasets of oss peer review repositories," Proc. MSR 2013, pp.49–52, 2013.
[16] K. Crowston and J. Howison, "The social structure of free and open source software development," First Monday, vol.10, no.2, 2005.
[17] R.G. Kula, A.E.C. Cruz, N. Yoshida, K. Hamasaki, K. Fujiwara, X. Yang, and H. Iida, "Using profiling metrics to categorise peer review types in the android project," Proc. ISSRE 2012, pp.146–151, 2012.
[18] P. Thongtanunam, X. Yang, N. Yoshida, R.G. Kula, A.E.C. Cruz, K. Fujiwara, and H. Iida, "Reda: A web-based visualization tool for analyzing modern code review dataset," Proc. ICSME 2014, pp.605–608, 2014.
[19] D.A. Wolfe and M. Hollander, Nonparametric statistical methods, John Wiley & Sons, 1973.
[20] E.S. Raymond, "The Cathedral and the Bazaar," 1st ed., O'Reilly & Associates, Inc., Sebastopol, CA, USA, vol.3, no.3-2, 1999.
[21] V. Balachandran, "Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation," Proc. ICSE 2013, pp.931–940, 2013.
[22] C. Bird, D. Pattison, R. D'Souza, V. Filkov, and P. Devanbu, "Latent social structure in open source projects," Proc. FSE 2008, pp.24–35, 2008.
[23] M.S. Zanetti, I. Scholtes, C.J. Tessone, and F. Schweitzer, "Categorizing bugs with social networks: a case study on four open source software communities," Proc. ICSE 2013, pp.1032–1041, 2013.
[24] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?," Proceedings of the 19th International Conference on World Wide Web, pp.591–600, 2010.

Xin Yang is a PhD student in the Graduate School of Information Science at Nara Institute of Science and Technology (NAIST). He received his M.E. from NAIST in 2013. His research interests are in Empirical Software Engineering, Mining Software Repository, Software Peer Review, and Social Network Analysis. He is a student member of ACM.

Norihiro Yoshida received his BE from the Kyushu Institute of Technology in 2004 and his Master and PhD from Osaka University in 2006 and 2009, respectively. He is an associate professor at Nagoya University. Before joining Nagoya University in 2014, he was an assistant professor at the Nara Institute of Science and Technology from 2010. His research interests include program analysis and software development environment. He is a member of the IEEE, the IEEE Computer Society, and the ACM.

Raula Gaikovina Kula is a currently Research Assistant Professor at Software Engineering Lab, Osaka University. In 2013, he graduated with a PhD from Nara Institute of Science and Technology. He is currently an active member of the IEEE Computer Society and ACM. His research interests include repository mining, peer review, API & software reuse and software visualizations.

Hajimu Iida received his B.E., M.E., and Dr. of Eng. degrees from Osaka University in 1988, 1990, and 1993, respectively. From 1991 to 1995, he worked for the Department of Information and Computer Science, Faculty of Engineering Science, Osaka University as a research associate. Since 1995 he has been with the Graduate School of Information Science, Nara Institute of Science and Technology, Japan. His current position is a Professor of the Laboratory of Software Design and Analysis. His research interests include modeling and analysis of software and development process.