1.

Core-Periphery Communication and the Success of Free/Libre Open Source Software Projects

Kevin Crowston* and Ivan Shamshurin

Syracuse University
School of Information Studies,
348 Hinds Hall
Syracuse, NY 13244–4100 USA

{crowston,ishamshu}@syr.edu

* Please address correspondence to this author.

Core-Periphery Communication and the Success of Free/Libre Open Source Software Projects

2. Abstract. We examine the relationship between communications by core and peripheral members and Free/Libre Open Source Software project success. The study uses data from 74 projects in the Apache Software Foundation Incubator. We conceptualize project success in terms of success building a community, as assessed by graduation from the Incubator. We compare successful and unsuccessful projects on volume of communication and on use of inclusive pronouns as an indication of efforts to create intimacy among team members. An innovation of the paper is that use of inclusive pronouns is measured using natural language processing techniques. We also compare the volume and content of communication produced by core (committer) and peripheral members and by those peripheral members who are later elected to be core members. We find that volume of communication is related to project success but use of inclusive pronouns does not distinguish successful projects. Core members exhibit more contribution and use of inclusive pronouns than peripheral members.

3. Keywords: free/libre open source software (FLOSS), core and periphery, communication, project success, Apache Software Foundation, natural language processing, inclusive pronouns

4 Introduction

Community-based Free/Libre Open Source Software (FLOSS) projects are developed and maintained by teams of individuals collaborating in globally-distributed environments [1]. The health of the developer community is critical for the performance of projects [2], but it is challenging to sustain a project with voluntary members over the long term [3; 4]. Social-relational issues have been seen as a key component of achieving project effectiveness [5] and

enhancing online group involvement and collaboration [6]. In this paper, we explore how community interactions are related to community health and so project success.

Specifically, we examine contributions made by members in different roles. Members have different levels of participation in FLOSS development and so take on different project roles [7]. A widely-accepted model of roles in community-based FLOSS teams is the core-periphery structure [5; 8; 9]. For example, Crowston and Howison [2] see community-based FLOSS teams as having an onion-like core-periphery structure, in which the core category includes core developers and the periphery includes co-developers and active users. Rullani and Haefliger [10] described periphery as a "cloud" of members that orbits around the core members of open source software development teams.

Generally speaking, access to core roles is based on technical skills demonstrated through the development tasks that the developer performs [11]. Core developers usually contribute most of the code and oversee the design and evolution of the project, which requires a high level of technical skills [2]. Peripheral members contribute at a lower level. Some submit patches such as bug fixes (i.e., co-developers), which provides an opportunity to demonstrate skills and interest. Others provide use cases and bug reports or test new releases but without contributing code directly (i.e., active users), which requires less technical skill [2].

Despite the difference in contributions, both core and peripheral members are important to the success of the project. It is evident that, by making direct contributions to the software developed, core members are vital to project development. On the other hand, even though they contribute only sporadically, peripheral members provide bug reports, suggestions and critical expertise that are fundamental for innovation [10]. In addition, the periphery is the source of new core members [12; 13], so maintaining a strong periphery is important to the long-term success of a project. Amrit and van Hillegersberg [8] examined core-periphery movement in open source

projects and concluded that a steady movement toward the core is beneficial to a project, while a shift away from the core is not. But how communication among core and periphery predicts project success has yet to be investigated systematically, a gap that this paper addresses.

Theory and hypotheses

To develop hypotheses for our study, we discuss in turn the dependent and independent variables in our study. The outcome of interest for our study is project success. Project success for FLOSS projects can be measured in many different ways, ranging from code quality to member satisfaction to market share [14]. For the community-based FLOSS projects we examine, success in building a developer community is a critical issue, so we chose building a developer community as our measure of success.

To identify the constructs that predict success, we examined communication among community members. A starting hypothesis is that more communication is predictive of project success:

H1: Successful projects will have a higher volume of communication than unsuccessful projects

More specifically, we are interested in how members in different roles contribute to projects. As noted above, projects rely on contributions from both core and peripheral members. We can therefore extend H1 to consider roles. Specifically, we hypothesize that:

H1a: Successful projects will have a higher volume of communication by core members than unsuccessful projects.

H1b: Successful projects will have a higher volume of communication by peripheral members than unsuccessful projects.

Prior research on the core-periphery structure in FLOSS development has found inequality in participation between core and peripheral members. For example, Luthiger Stoll [15] found that core members make greater time commitment than peripheral members: core participants spend an average of 12 hours per week, with project leaders averaging 14 hours, and bug-fixers and otherwise active users, around 5 hours per week. Similarly, using social network analysis, Toral et al. [16] found that a few core members post the majority of messages and act as middlemen or brokers among other peripheral members. We therefore hypothesize that:

H2: Core members will contribute more communication than will peripheral members.

Prior research on the distinction between core-periphery has mostly focused on coding-related behaviour, as project roles are defined by the coding activities performed [5]. However, developers do more than just coding [5]. Both core and peripheral members need to engage in social-relational behaviour in addition to task-oriented behaviour such as coding. Consideration of these non-task activities is important because effective interpersonal communication plays a vital role in the development of online social interaction [17].

Scialdone et al. [18] and Wei et al. [19] analyzed group maintenance behaviours used by members to build and maintain reciprocal trust and cooperation in their everyday interaction messages, e.g., through emotional expressions and politeness strategies. Specifically, Scialdone et al. [18] found that core members of two teams used more politeness strategies than did peripheral members. They noted in particular that "peripheral members in general do not feel as comfortable expressing a sense of belonging within their groups". We therefore hypothesize that:

H3: Core members will use more expressions of belonging to the team in their communication than will peripheral members.

Scialdone et al. [18] further noted that one team they studied that had ceased production had exhibited a greater gap between core and periphery in usage of expressions of belonging to the team. Such a situation could indicate that the peripheral members of the group did not feel ownership of the project, with negative implications for their future as potential core members. We therefore hypothesize that:

H3a: Successful projects will have a higher level of expressions of belonging to the team by core members than unsuccessful projects.

H3b: Successful projects will have a higher level of expressions of belonging to the team by peripheral members than unsuccessful projects.

Finally, as noted above, the periphery is the source of new core members. Active peripheral members may be invited to become project committers, thus joining the core. We therefore expect that peripheral members who become core members will be as active as core members in communicating even before being officially elected as core members (that is, that they are selected based on their activity, rather than their activity being driven by their committer status). We therefore hypothesize that:

H4: Peripheral members who later become core members will resemble core members in their communication behaviours.

5 Materials and methods

5.1 Setting

Scialdone et al. [18] and Wei et al. [19] studied only a few projects and noted problems making comparisons across projects that can be quite diverse. To address this concern, in this paper we studied a larger number of projects (74 in total) that all operated within a common

framework at a similar stage of development. Specifically, we studied projects in the Apache Software Foundation (ASF) Incubator. The ASF is an umbrella organization including more than 60 free/libre open source software (FLOSS) development projects. The ASF's apparent success in managing FLOSS projects has made it a frequently mentioned model for these efforts, though often without a deep understanding of the factors behind that success.

The ASF Incubator's purpose is to mentor new projects to the point where they can successfully join the ASF. Projects are invited to join the Incubator based on an application and support from a sponsor (a member of the ASF). Accepted projects (known as Podlings) receive support from one or more mentors, who help guide the Podlings through the steps necessary to become a full-fledged ASF project. It should be noted that projects may already be well established when they apply to join the Apache Foundation.

The incubation process has several goals, including fulfillment of legal and infrastructural requirements and development of relationships with other ASF projects, but the main goal is to develop effective software development communities, which Podlings must demonstrate to graduate from the Incubator. The Apache Incubator specifically promotes diverse participation in development projects to improve the long-term viability of the project community and ensure requisite diversity of intellectual resources. The time projects spend in incubation varies widely, from as little as two months to nearly five years, indicating significant diversity in the efforts required for Podlings to become viable projects. The primary reason that projects are retired from the Incubator (rather than graduated) is a lack of community development that stalls progress. Volume of communication is not a graduation criterion because according to Apache website, "the incubation period normally serves to estimate whether or not: the project is able to increase the diversity of its committer base and to play with the meritocratic rules of the foundation" [http://www.apache.org/foundation/how-it-works.html].

5.2 Data Collection and Processing

In FLOSS settings, collaborative work primarily takes place by means of asynchronous computer-mediated communication such as email lists and discussion fora [7]. This practice is common for Apache projects because asynchronous communication is a general requirement for groups that are so geographically distributed as to cover all time zones (it is normally the case for the various Apache communities) [see http://www.apache.org/foundation/how-it-works.html]. Also, asynchronous communication allows archives to be created and it is more tolerant of the volunteer nature of the various communities [see http://www.apache.org/foundation/how-it-works.html]. ASF community norms strongly support transparency and broad participation, which is accomplished via electronic communications, such that even collocated participants are expected to document conversations in the online record, i.e., the email discussion lists. We therefore drew our data from messages on the developers' mailing list for each project.

A Perl script was used to collect messages in html format from the email archive website markmail.org. We discarded any messages sent after the Podling either graduated or retired from the ASF Incubator, as many of the projects apparently used the same email list even after graduation. We are analyzing emails for one list per project, so we did not face the issue of cross-posted messages appearing multiple times in the corpus for a project. We did not otherwise check for duplicate messages. After the dataset was collected, relevant data (sender and message contents) were extracted from the html files representing each message thread and other sources. We manually reviewed the sender addresses to identify non-human message senders (e.g., messages from bug reporting or continuous integration systems). Messages from these senders were removed from the analysis.

5.2.1 Dependent variable: Success

As noted above, the dependent variable for H1 and H3, project success in building a community, was determined by whether the project had graduated (success) or been retired (not success) from the incubator. Graduation was determined based on the list of projects maintained by the Apache Incubator and available on the Apache website. The dataset includes email messages for 24 retired and 50 graduated Podlings. The data set also included messages for some projects still in incubation and some with unknown status; these were not used in the analysis.

As a check on the validity of graduation from the Incubator as a measure of success in building a community, we compared the number of active developers in graduated and retired projects (active developers were those who had participated on the mailing list). The results are shown in Table 1 and Figure 1. As the Table shows, the median graduated project had more than twice as many developers active on the mailing list as did retired projects. To check the significance of this difference we applied a Wilcoxon rank-sum test (also known as Mann-Whitney U), chosen because the data are not normally distributed. (A Wilcoxon test is used for all tests of significance reported in this paper.) The test shows that the difference in the number of developers between graduated and retired projects is statistically significant, $p=0.000$. The effect size of this difference, r, was 0.33, between small and medium. Furthermore, only graduated projects had future core members who posted on the mailing list during incubation. (By future core members, we mean committers who were not in the list of committers at the start of the data collection but were elected later as documented in an announcement to the mailing list.) These results provide evidence for the validity of our choice of graduation as a measure of success in building a project community.

5.2.2 Core vs. periphery

For all hypotheses, we distinguish between core and peripheral members. Crowston et al. [20] suggested three methods to identify core and peripheral members in FLOSS teams: relying on project-reported formal roles, analysis of distribution of contributions based on Bradford's Law of Scatter and core-and-periphery analysis of the network formed by developer communications. Their analysis showed that all three measures were highly correlated, but that relying on project-reported roles was the most accurate. [21] suggested identifying core developers by examining contributions to core modules, but this approach requires analysis of the code base. Therefore, in this study, we identified a message sender as a core member if the sender's name was on the list of project committers on the project website. If we did not find a match in the list of committers, then the sender was labeled as a peripheral member.

The list of committers we used included names but not email addresses. Therefore, we developed a matching algorithm to take account of the different ways that names appear in email messages. Specifically, we checked for matches with different capitalization, with and without a middle name and with the first and family names reversed. We attributed to the developer all messages that matched the given name, regardless of the specific email address used. All messages were attributed to either a known developer from the website, or to a non-core developer; no messages were discarded. There are only a few hundred developers, with distinctive names, so we did not need to apply advanced name disambiguation heuristics that have been created to handle thousands of identical names (e.g., on publication author lists). We simply needed to pick up variations in names created by the email software (e.g., order of names, inclusion of middle initial).

We also looked for evidence of new committers joining a project during incubation. Projects that join the Apache Incubator start with an initial set of committers. New committers

are elected by the current committers. When a new committer is elected, an announcement is made to the mailing list. We therefore wrote a program to search the email message archive for these announcement messages. Announcement messages are those that have "[ANNOUNCE] OR [ANN] + committer" in their subject. Matching was done as follows: first name + family name, first name + middle name + family name. We found 19 projects in which new committers had been elected, for a total of 83 new committers. Only one name from our list of developers was among the names from the announcements. Interestingly, these elections all happened after the projects had graduated or retired from incubation. Some, but not all new committers participated in the mailing lists before the projects left the incubator. We labelled messages from these members as from "future core".

5.2.3 Expressions of belonging to the team

In this paper, we examine one factor identified by Scialdone et al. [18] in their investigation of how core and peripheral members use language to create "intimacy among team members" thus "building solidarity in teams". Specifically, for H3 and H4, we examined the use of inclusive pronouns as one way that team members build a sense of belong to the group. Scialdone et al. [18] noted that such use of inclusive pronouns is "consistent with Bagozzi and Dholakia [22]'s argument about the importance of we-intention in Linux user groups, i.e., when individuals think themselves as 'us' or 'we' and so attempt to act in a joint way".

Inclusive pronouns were defined as:

> *reference to the team using an inclusive pronoun. If we see "we" or "us" or "our", and it refers to the group, then it is Inclusive Reference. Not if "we" or "us" or "our" refer to another group that the speaker is a member of.*

That is, the sentences were judged on two criteria: 1) whether there are language cues for inclusive reference (a pronoun), as specified in the definition above and 2) if these cues refer to

the current group rather than to another group. To judge the second criteria may require reviewing the sentence in the context of the whole conversation. This usage is only one of the many indicators of group maintenance studied by Scialdone et al. [18] and Wei et al. [19], but it is interesting and tractable for analysis.

To handle the large volume of messages drawn from many projects, we applied NLP techniques as suggested (but not implemented) by previous research. Specifically, we used a machine-learning (ML) approach, where an algorithm learns to classify sentences from a corpus of human-analyzed data. Sentences were chosen as the unit for the NLP analysis instead of the thematic units more typically used in human analysis, because sentences can be more easily identified for machine learning. We expected that the NLP would have no problem handling the first part of the definition, but that the second (whether the pronoun refers to the project or some other group) would pose challenges.

Training data were obtained from the SOCQA (Socio-computational Qualitative Analysis) project at the Syracuse University (http://socqa.org/) [23; 24]. The training data consists of 10,841 sentences drawn from two Apache projects, SpamAssassin and Avalon. Trained annotators manually annotated each sentence as to whether it included an inclusive pronoun (per the above definition) or not and cross-checked their results to ensure reliability. The distribution of the classes in the training data is shown in Table 2. Note that the sample is unbalanced (there are many fewer sentences with inclusive pronouns than without).

A standard vector space model was used to transform text into vectors. In particular, term-frequency inverse-document frequency (TF-IDF) with frequency and presence (binary) term-document matrices were used. As features for the ML, we used bag of words, experimenting with unigrams, bigrams and trigrams. We used the default stop word list from the Python nltk package. No stemming was performed because stemming would not change the

pronouns themselves. As well, the predictive value of our models was already very good, so we did not expect stemming to improve it significantly. Naïve Bayes (MNB), k Nearest Neighbors (KNN) and Support Vector Machines (SVM) algorithms (Python LibSVM implementation) were trained and applied to predict the class of the sentences, i.e., whether a sentence has inclusive pronoun or not.

Ten-fold cross-validation was used to evaluate the classifier's performance on the training data. Results are shown in Table 3. The results (accuracy, the proportion of correctly classified instances) show that though all three approaches gave reasonable performance, SVM outperforms other methods. The Linear SVM model was therefore selected for further use. The trained SVM model significantly outperforms a majority vote rule baseline (classify all examples as the majority class), which provides an accuracy of 0.87. We experimented with tuning SVM parameters such as minimal term frequency, etc. but did not find settings that affected the accuracy, so we used the default settings.

To further evaluate model performance, the model was applied to new data and the results checked by a trained annotator (one of the annotators of the training data set). Specifically, we used the model to code 200 sentences (10 sentences randomly selected from 5 projects each in the "graduated", "in incubator", "retired" and "unknown" classes of projects). The human annotator annotated the same sentences and we compared the results. The Cohen kappa (agreement corrected for chance agreement) for the human vs. machine annotation was 88.6%, which is higher than the frequently-applied threshold for human analysis of 80% agreement. In other words, the ML model performed at least as well as a second human analyst would be expected to do.

Surprisingly, when we examined the results, we found no cases where a predicted inclusive reference refers to another group, suggesting that the ML had managed to learn the

second criterion. Two sentences that the model misclassified are illustrative of limitations of the approach:

> *It looks like it requires work with "our @patterns" in lib/path.pmI looked at the path.pm for www.apache.org and it is a clue.*

The actual class is "no" but the classifier marked it as "yes" because the inclusive pronoun "our" was included in the sentence, though in quotation marks as part of a code snippet. The human coder knew to ignore this section, but there were no features to enable the ML to learn to do so.

> *Could also clarify download URLs for third-party dependencies wecan't ship.*

In this sentence, the actual class is "yes" but the model marked the sentence as "no" due to the error in spelling (no space after "we"). The human annotator ignored the error, but there were not enough examples of such errors for the ML to learn to do so.

Despite such limitations, the benefit of being able to handle large volumes of email more than makes up for the possible slight loss in reliability of coding, especially considering that human coders are also not perfectly reliable.

6 Findings

In this section, we discuss in turn the findings from our study, examining support for each hypothesis, deferring discussion of the results to the following section. Hypothesis 1 was that successful projects would have more communication. Table 4 shows the median of the total messages, by project status and developer role of the sender. Note that because the distribution of the count of messages sent is skewed, we report medians and significance tests are done with a non-parametric test that does not make distributional assumptions. Figure 2 provides a violin plot of this data. A violin plot is like a box plot, but includes a kernel density plot for the data, thus showing the distribution in more detail. As shown in Table 4 and Figure 2, Hypothesis 1 is

strongly supported, as graduated projects have many times more messages sent during the incubation process than retired projects (p=0.000, r=0.34). (r is the effect size of the difference.)

Hypotheses 1a and 1b were that core and peripheral members respectively would communicate more in successful projects than in unsuccessful projects. Table 4 shows the median of the total number of messages sent in a project, by project status and developer role. The differences in Tables 4 and 5 show that these hypotheses are supported (p=0.000, r=0.34 for core and p=0.001, r=0.26 for peripheral members for total message count in graduated vs. retired projects. The tests exclude future core members.

However, as we noted above, graduated projects have more developers and so would be expected to have more communication for that reason. To control for the number of developers, Table 5 shows the median of the median number of messages per developer by project status and developer role. There was not a significant difference in the median number of messages sent between graduated and retired projects for either core or peripheral developers (p=0.62 and p=0.44 respectively).

Hypothesis 2 was that core members would communicate more than peripheral members. From Table 4, we can see that in total core members do send more messages than peripheral members in graduated projects, though the total is about the same in retired projects. However, there are fewer core members, so the median core developer in a project sends many more messages than the median peripheral developer, as shown in Table 5 and Figure 3 (p=0.000, r=0.49).

We now turn from the volume of messages to consider the content of the messages. Hypothesis 3 was that core members would use more inclusive pronouns than peripheral members. Table 6 and Figure 4 shows the median number of messages sent by developers that

included an inclusive pronoun. The table shows that core developers do send more messages with inclusive pronouns in both graduated and retired projects (p=0.000, r=0.64).

To control for the fact that core developers send more messages in general, we computed the percentage of messages that include an inclusive pronoun, as shown in Table 7 and Figure 5. From this table, we can see that the median percentage of messages sent by core developers that include an inclusive pronoun is higher than for peripheral members (p=0.000, r=0.60).

Hypotheses 3a & b were that there would be more use of inclusive pronouns by core and peripheral members respectively in successful projects compared to unsuccessful projects. However, from Table 6, we can see that usage is nearly the same, so this hypothesis is not supported (p=0.94 for core members; p=0.19 for non-core members). When considered as a percentage of messages, again we find no significant difference between graduated and retired project (p=0.45 for core developers; p=0.10 for peripheral developers). Accordingly, neither hypothesis is supported.

Finally, hypothesis 4 was that future core developers would behave the same as core developers. Surprisingly, the analyses above suggest that the median future core developer actually sends more messages (Table 5) but appear like core developers in use of inclusive pronouns (Table 6). Overall, Hypothesis 4 is supported.

7 Discussion

In general, our data suggest that successful projects (i.e., those that successfully graduated from incubation) have more members and a correspondingly large volume of communication, suggesting an active community. Given that community development is one of the goals of the Apache incubation process, this outcome should be expected. Also as expected, core members contribute more than non-core developers. Nevertheless, there is a high volume of

messages for both core and peripheral members, suggesting that both roles play a part in projects.

As expected, core members do display greater expressions of belonging to the team as expressed in the use of inclusive pronouns. This finding supports Scialdone et al.'s [18] hypothesis that "peripheral members in general do not feel as comfortable expressing a sense of belonging within their groups", which is consistent with the notion of peripheral members as being less connected to projects. However, counter to our expectations, the use of inclusive pronouns did not distinguish successful and unsuccessful projects. This finding suggests that while it is true that the unsuccessful projects failed in growing their membership, there is no evidence that this failure was due to peripheral members of the group feeling less ownership of the project (or at least, not expressing ownership in their language) and so not moving in to the core.

It may be that median is not the important measure: perhaps projects need only a few committed peripheral members, not a high average level of commitment. It is noteworthy that only the graduated projects had peripheral members who moved into the core. In conclusion, while growing the community in general and the core in particular seems to be important, other explanations need to be sought for differences in success in attracting new core members.

7.1 Threats to validity

As with any study, there are possible threats to the validity of our conclusions. We cover in turn threats to construct validity, to internal validity and to external validity.

7.1.1 Construct validity

Construct validity concerns the ability of the measured data to represent the construct of interest. In the theory development section, we argued why the data we chose represent the

concepts of interest. Specifically, we argued above that graduation from the incubator is a good measure of project success, that project committer status is a good measure of core or peripheral status and that email use is a good measure of project communication. We further argued that use of inclusive pronouns reflects commitment of members to the project.

However, the data we used in this paper are all based on a single measure for each construct. While we do not believe that the measures are biased, it might increase construct validity if they were based on multiple sources of data. A further issue is that the current measures of two of the constructs are binary. It might provide more insight if we could develop a more nuanced measure of success to replace the graduated vs. retired measure we used or of developer status to replace core vs. periphery. Finally, it could be that some non-core developers have names identical to core developers and their messages are being included with the core developers. However, core and peripheral members behave quite differently in our analysis, so such possible misidentifications, if any, do not seem to have impacted our findings.

7.1.2 Threats to internal validity

Threats to internal validity are those that affect the conclusions drawn from the study by offering explanations for the outcome beside the independent variables. Many well-known threats to internal validity do not apply to a non-experimental study, e.g., history, maturation, instrumentation change or interaction of treatment and construct. The data for our study came from non-reactive observation, which rules out threats to internal validity that arise from the study itself influencing the behaviours of the participants, e.g., testing, experimenter demand, hypothesis guessing or resentful demoralization. In our study, we included the whole population of Apache Incubator projects, eliminating threats to internal validity that arise from selection bias or regression to the mean.

However, there is a possible threat to statistical conclusion validity. A major finding of the study was that the use of inclusive pronouns did not seem to explain the difference between successful and unsuccessful projects. However, while the p values for peripheral members are not significant, they are low, so it could be that this negative finding is due to an under-powered statistical test, specifically, the non-parametric test used due to concerns about the skewed distribution of the data.

7.1.3 Threats to external validity

Finally, threats to external validity concern the possibility to generalize from the study findings to other settings. The study included only projects in the Apache Incubator. The external validity of the findings could be assessed by analyzing projects other than Apache Incubator Podlings. For example, our focus on email is appropriate for Apache projects, but other projects with different policies may also use IRC, issue tracker or newer mechanisms (e.g., Slack, HipChat, Mattermost) as a communication channel, so those communications would need to be included. Furthermore, it could be that the periphery expressing ownership of the project is important later in the lifecycle of project, not in the earlier period captured during many incubation projects. Future studies could address projects at later stages.

Conclusions

The work presented here can be extended in many ways in future work. First, the ML NLP might be improved with a richer feature set, though as noted, the performance was already as good as would be expected from an additional human coder. Second, we can consider the effects of additional group maintenance behaviours suggested by Wei et al. [19]. The Syracuse SOCQA project has had some success applying ML NLP techniques to these codes, suggesting

that this analysis is feasible. Similarly, in this paper we have considered only communication behaviours. A more complete model of project success would take into account measure of development activities such as code commits or project topic, data for which are available online. Finally, research might consider the temporal aspects of the incubation process, e.g., entropy/mean of time response between the messages, to see how developer engagement evolves over time [e.g, 25].

The research might also be extended by developing practical uses of the analyses. First, it would be interesting to examine the first few months of a project for early signs that are predictive of its eventual outcome. Project leaders might be able to use such diagnostics to identify problems while there is still time to act. It might similarly be possible to predict which peripheral members will become core members from their individual actions. However, it is necessary to consider limits to the hypothesized impacts before using them to provide advice to nascent communities. For example, we hypothesized that more communication reflects a more developed community, but it could be that too much communication creates information overload and so has a negative impact. Focusing attention on particular peripheral members identified as potential future core members could create a self-fulfilling prophecy or discourage other peripheral members.

Despite its limitations, our research offers several advances over prior work. First, it examines a much large sample of projects than prior work examining core-peripheral communications and group maintenance behaviours. Second, it uses a more objective measure of project success, namely graduation from the ASF Incubator, as a measure of community development. Finally, it shows the viability of the application of NLP and ML techniques to processing large volumes of email messages, incorporating analysis of the content of messages, not just counts or network structure.

8 Declarations

Availability of data and materials: Raw project and email data are available from the Apache Software Foundation and MarkMail as noted in the paper. Training data for inclusive pronouns were provided by the Syracuse SOCQA project and can be requested from that project. Processed email and contributor data are available from the authors on request.

9 List of abbreviations

ASF Apache Software Foundation

FLOSS Free/Libre Open Source Software

KNN k Nearest Neighbors

MNB Naïve Bayes

SOCQA Socio-computational Qualitative Analysis

SVM Support Vector Machines

10 Ethics approval

The research uses pre-existing public data sources and so did not require ethics approval under United State human-subjects research regulations.

11 Funding

This research drew on data from the Syracuse University SOCQA project, which was partially supported by a grant from the US National Science Foundation Socio-computational Systems (SOCS) program, award 11–11107.

12 Competing interests

The authors declare that they have no competing interests.

13 Authors' contributions

The first author developed the hypotheses and study design, carried out the statistical analyses and wrote most the paper. The second author collected and processed the email data and developed the natural language processing used to classify the sentences.

14 Acknowledgements

A prior version of this paper was presented at the IFIP Working Group 2.13 OSS2016 Conference [26]. The current version includes additional data analysis done to identify peripheral members who became committers and their contributions, with corresponding changes to the methods, findings and discussion sections.

We thank the SOCQA Project (Nancy McCracken PI) for access to the coded sentences for training and Feifei Zhang for checking the coding results.

15 References

[1] Crowston K, Li Q, Wei K, Eseryel UY, Howison J (2007) Self-organization of teams for free/libre open source software development. *Information and Software Technology, 49*(6): 564–575. doi: 10.1016/j.infsof.2007.02.004.

[2] Crowston K, Howison J (2006) Assessing the health of open source communities. *IEEE Computer, 39*(5): 89–91. doi: 10.1109/MC.2006.152.

[3] Bonaccorsi A, Rossi C (2003) Why Open Source software can succeed. *Research Policy, 32*(7): 1243–1258. doi: 10.1016/S0048-7333(03)00051-9.

[4] Fang Y, Neufeld D (2009) Understanding sustained participation in open source software projects. *Journal of Management Information Systems, 25*(4): 9-50. doi: 10.2753/MIS0742-1222250401.

[5] Barcellini F, Détienne F, Burkhardt J-M (2014) A situated approach of roles and participation in Open Source Software Communities. *Human–Computer Interaction, 29*(3): 205-255. doi: 10.1080/07370024.2013.812409.

[6] Park JR (2007) Interpersonal and affective communication in synchronous online discourse. *Library Quarterly, 77*(2): 133-155. doi: 10.1086/517841

[7] Crowston K, Wei K, Howison J, Wiggins A (2012) Free/libre Open Source Software development: What we know and what we do not know. *ACM Computing Surveys, 44*(2): 7:1–7:35. doi: 10.1145/2089125.2089127.

[8] Amrit C, van Hillegersberg J (2010) Exploring the impact of socio-technical core-periphery structures in open source software development. *Journal of Information Technology, 25*(2): 216–229. doi: 10.1057/jit.2010.7.

[9] Jensen C, Scacchi W (2007) Role migration and advancement processes in OSSD Projects: A comparative case study. In *Proceedings of the International Conference on Software Engineering (ICSE)*, Minneapolis, MN, pp. 364–374. doi: 10.1109/ICSE.2007.74.

[10] Rullani F, Haefliger S (2013) The periphery on stage: The intra-organizational dynamics in online communities of creation. *Research Policy, 42*(4): 941-953. doi: 10.1016/j.respol.2012.10.008.

[11] Jergensen C, Sarma A, Wagstrom P (2011) The onion patch: Migration in open source ecosystems. In *Proceedings of the ACM SIGSOFT Symposium and the European Conference on Foundations of Software Engineering*, pp. 70–80. ACM. doi: 10.1145/2025113.2025127.

[12] Dahlander L, O'Mahony S (2011) Progressing to the center: Coordinating project work. *Organization Science, 22*(4): 961-979. doi: 10.1287/orsc.1100.0571.

[13] von Krogh G, Spaeth S, Lakhani KR (2003) Community, joining, and specialization in Open Source Software innovation: A case study. *Research Policy, 32*(7): 1217–1241. doi: 10.1016/S0048-7333(03)00050-7.

[14] Crowston K, Howison J, Annabi H (2006) Information systems success in Free and Open Source Software development: Theory and measures. *Software Process—Improvement and Practice, 11*(2): 123–148. doi: 10.1002/spip.259.

[15] Luthiger Stoll B (2005) Fun and software development. In *Proceedings of the International Conference on Open Source Systems*, Genova, Italy. Available from: http://www.aktion-hip.ch/texts/BLuthiger_Fun_SoftwareDevel_OSS2005.pdf.

[16] Toral S, Martínez-Torres M, Barrero F (2010) Analysis of virtual communities supporting OSS projects using social network analysis. *Information and Software Technology, 52*(3): 296-303. doi: 10.1016/j.infsof.2009.10.007.

[17] Park J-r (2008) Linguistic politeness and face-work in computer mediated communication, Part 2: An application of the theoretical framework. *Journal of the American Society for Information Science and Technology, 59*(14): 2199–2209. doi: 10.1002/asi.20926.

[18] Scialdone MJ, Heckman R, Crowston K (2009) Group maintenance behaviours of core and peripheral members of free/libre open source software teams. In *Proceedings of the*

Conference on Open Source Systems, Skövde, Sweden. Springer. doi: 10.1007/978-3-642-02032-2_26.

[19] Wei K, Crowston K, Li NL, Heckman R (2014) Understanding group maintenance behavior in Free/Libre Open-Source Software projects: The case of Fire and Gaim. *Information and Management, 51*(3): 297–309. doi: 10.1016/j.im.2014.02.001.

[20] Crowston K, Wei K, Li Q, Howison J (2006) Core and periphery in Free/Libre and Open Source software team communications. In *Proceedings of the Hawai'i International Conference on System System (HICSS-39)*, Kaua'i, Hawai'i. doi: 10.1109/HICSS.2006.101.

[21] Oliva GA, da Silva JT, Gerosa MA, Santana FWS, Werner CML, de Souza CRB, de Oliveira KCM (2015) Evolving the system's core: A case study on the identification and characterization of key developers in Apache Ant. *Computing And Informatics, Slovakia, 34*(3). Available from: http://www.cai.sk/ojs/index.php/cai/article/view/3225

[22] Bagozzi RP, Dholakia UM (2006) Open source software user communities: A study of participation in Linux user groups. *Management Science, 52*(7): 1099–1115. doi: 10.1287/mnsc.1060.0545.

[23] Yan JLS, McCracken N, Crowston K. (2014). *Design of an active learning system with human correction for content analysis*. Paper presented at the Workshop on Interactive Language Learning, Visualization, and Interfaces, 52nd Annual Meeting of the Association for Computational Linguistics, Baltimore, MD. Available from http://socqa.org/sites/crowston.syr.edu/files/ILLWorkshop.ACLFormat.04.28.14.final_.pdf

[24] Yan JLS, McCracken N, Crowston K (2014) Semi-automatic content analysis of qualitative data. In *Proceedings of the iConference*, Berlin, Germany. Available from: http://socqa.org/sites/crowston.syr.edu/files/iConference_Poster_Published.pdf.

[25] Zanetti MS, Scholtes I, Tessone CJ, Schweitzer F (2013) The rise and fall of a central contributor: Dynamics of social organization and performance in the GENTOO community. In *Proceedings of the 2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pp. 49-56. doi: 10.1109/CHASE.2013.6614731.

[26] Crowston K, Shamshurin I (2016) Core-Periphery communication and the success of Free/Libre Open Source Software projects. In *Proceedings of the IFIP International Conference on Open Source Systems (OSS2016)*, Gothenburg, Sweden, pp. 45–56. Springer. doi: 10.1007/978-3-319-39225-7_4.

16 Tables

Table 1. Median number of developers by project status and developer role

Project status	Core	Future core	Peripheral
Graduated	26	1	44
	(18.2)	(4.7)	(102.1)
Retired	12.5	0	23
	(9.0)		(17.9)

N = 74 projects. Standard deviations in parentheses.

Table 2. Distribution of classes in the training data

	#	%
Sentences with inclusive pronouns	1395	12.9
Sentences without inclusive pronouns	9446	87.1
Total	10841	

Table 3. Accuracy of 10-fold Cross-Validation on the Training Data

	Unigram	Bigram	Trigram
Naïve Bayes (MNB)	0.86	0.81	0.75
k Nearest Neighbors (KNN)	0.89	0.89	0.88
Support Vector Machines (SVM) (LinearSVC)	0.97	0.97	0.97

Table 4. Median of total project messages by project status and developer role

	Core	Future core	Peripheral
Graduated	5481.5	67	2203
	(8259)	(2137)	(7007)
Retired	917		814
	(1811)		(2023)

N = 74 projects. Standard deviations in parentheses.

Table 5. Median of median number of messages sent per developer by project status and developer role

	Core	Future core	Peripheral
Graduated	27.25	64.5	11
	(29.0)	(65.8)	(3.6)
Retired	20.25		13
	(44.4)		(18.6)

N = 74 projects. Standard deviations in parentheses.

Table 6. Median number of messages including an inclusive pronoun sent per developer by project status and developer role

	Core	Future core	Periphery
Graduated	2	2	0
	(1.9)	(4.81)	(0.3)
Retired	1.25		0
	(3.7)		(2.6)

N = 74 projects. Standard deviations in parentheses.

Table 7. Median percentage of messages that include an inclusive pronoun per developer by project status and developer role

	Core	Future core	Periphery
Graduated	4	2.25	0
	(4.0)	(7.0)	(0.8)
Retired	4		0
	(5.3)		(3.3)

N = 74 projects. Standard deviations in parentheses.

17 Figure captions

Figure 1. Violin plot of number of developers by project status and developer role.

Figure 2. Violin plot of total number of messages sent by project status and developer role.

Figure 3. Violin plot of mean number of messages sent per developer by project status and developer role.

Figure 4. Violin plot of mean number of messages sent that include an inclusive pronoun by project status and developer role.

Figure 5. Violin plot of mean percentage of messages sent that include an inclusive pronoun by project status and developer role.