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

Kevin Crowston[1(✉)] and Ivan Shamshurin[2]

[1] Syracuse University School of Information Studies, 348 Hinds Hall, Syracuse, NY 13244–4100, USA
crowston@syr.edu

[2] Syracuse University School of Information Studies, 337 Hinds Hall, Syracuse, NY 13244–4100, USA
ishamshu@syr.edu

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 by core (committer) and peripheral community members 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 find that core and peripheral members differ in their volume of contribution and in their use of inclusive pronouns, and that volume of communication is related to project success.

1 Introduction

Community-based Free/Libre Open Source Software (FLOSS) projects are developed and maintained by teams of individuals collaborating in globally-distributed environments [8]. The health of the developer community is critical for the performance of projects [7], but it is challenging to sustain a project with voluntary members over the long term [4, 11]. Social-relational issues have been seen as a key component of achieving design effectiveness [3] and enhancing online group involvement and collaboration [15]. 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 taken on different roles [5]. A widely accepted models of roles in community-based FLOSS teams is the core-periphery structure [1, 3, 12]. For example, Crowston and Howison [7] 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 [17] described periphery as a "cloud" of members that orbits around the core members of open source software development teams.

© IFIP International Federation for Information Processing 2016
Published by Springer International Publishing Switzerland 2016. All Rights Reserved
K. Crowston et al. (Eds.): OSS 2016, IFIP AICT 472, pp. 45–56, 2016.
DOI: 10.1007/978-3-319-39225-7_4

Generally speaking, access to core roles is based on technical skills demonstrated through the development tasks that the developer performs [13]. 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 [7]. Peripheral members, on the other hand, submit patches such as bug fixes (co-developers), which provides an opportunity to demonstrate skills and interest, or just provide use cases and bug reports or test new releases without contributing codes directly (active users), which requires less technical skill [7].

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 [17]. In addition, the periphery is the source of new core members [10, 20], so maintaining a strong periphery is important to the long-term success of a project. Amrit and van Hillegersberg [1] 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.

2 Theory and Hypotheses

To develop hypotheses for our study, we discuss in turn the dependent and independent variables in our study.

The dependent variable 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 [6]. 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 independent variables that predict success (i.e., success in building a developer community), we examine 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:

H2a: Successful projects will have a higher volume of communication by core members than unsuccessful projects.
H2b: 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 [14] found that core members make greater time commitment than peripheral members: core participants spend an average of 12 h per week, with project leaders averaging 14 h, and bug-fixers and otherwise active users, around 5 h per week. Similarly, using social network analysis, Toral et al. [19] 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:

H3: 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 [3]. However, developers do more than just coding [3]. 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 [16].

Scialdone et al. [18] and Wei et al. [21] 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. In this paper, we examine one factor they identified, investigating how core and peripheral members use language to create "intimacy among team members" thus "building solidarity in teams". Specifically, Scialdone et al. [18] found that core members of two teams used more inclusive pronouns (i.e., pronouns referring to the team) than did peripheral members. They interpreted this finding as meaning that "peripheral members in general do not feel as comfortable expressing a sense of belonging within their groups". We therefore hypothesize that:

H4: Core members will use more inclusive pronouns 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 inclusive pronouns. Such a situation could indicate that the peripheral members of the group do not feel ownership of the project, with negative implications for their future as potential core members. Scialdone et al. [18] noted that such use of inclusive pronouns is "consistent with Bagozzi and Dholakia [2]'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". A similar argument can be made for the importance of core member use of inclusive pronouns. We therefore hypothesize that:

H5a: Successful projects will have a higher usage of inclusive pronouns by core members than unsuccessful projects.
H5b: Successful projects will have a higher usage of inclusive pronouns by peripheral members than unsuccessful projects.

3 Methods

3.1 Setting

Scialdone et al. [18] and Wei et al. [21] studied only a few projects and noted problem making comparison 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 are able to 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.

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 in order 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.

3.2 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 [5]. 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 site http://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. After the dataset was collected, relevant data was extracted from the html files representing each message thread and other sources.

3.2.1 Dependent Variable: Success

The dependent variable, project success in building a community, was determined by whether the project had graduated (success) or been retired (not success) 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 for further analysis.

As a check on this measure of successful community development, we examined the number of developers active in the community (a more successful community has more developers). We considered as active members of the projects those who sent an email to the developer mailing list during incubation.

3.2.2 Core Vs. Periphery

Crowston et al. [9] 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 project social network. Their analysis showed that relying on project-reported roles was the most accurate. 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, then the sender was labeled as non-committer (peripheral member). we developed a matching algorithm to take into account the variety of ways that names appear in email message.

3.2.3 Inclusive Pronouns

As noted above, we examined the use of inclusive pronouns as one way that team members build a sense of belong to the group. 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 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 studied by Scialdone et al. [18] and Wei et al. [21], 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 already coded data. Sentences were chosen as the unit of coding instead of the thematic units more typically used in human coding, because sentences can be more easily identified for machine learning. Training data was obtained from the SOCQA (Socio-computational Qualitative Analysis) project at the Syracuse University (http://socqa.org/) [22, 23]. The training data consists of 10,841

sentences drawn from two Apache projects, SpamAssassin and Avalon. Trained annotators manually coded each sentence as to whether it included an inclusive pronoun (per the above definition) or not. The distribution of the classes in the training data is shown in Table 1 ("yes" means the sentence has an inclusive pronoun). Note that the sample is unbalanced.

Table 1. Distribution of classes in the training data

	#	%
"yes"	1395	12.9
"no"	9446	87.1
Total	10841	

As features for the ML, we used bag of words, experimenting with unigrams, bigrams and trigrams. 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. 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.

10-fold cross-validation was used to evaluate the classifier's performance on the training data. Results are shown in Table 2. The results show that though all three approaches gave reasonable performance, SVM outperformed other methods. The Linear SVM model was therefore selected for further use. 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.

Table 2. Results of 10-fold cross-validation on the training data

	Unigram	Bigram	Trigram
MNB	0.86	0.81	0.75
KNN	0.89	0.89	0.88
SVM (LinearSVC)	0.97	0.97	0.97

The random guess baseline for a binary classification task would give an accuracy of 0.5; a majority vote rule baseline (classify all examples to the majority class) provides an accuracy of 0.87. The trained SVM model significantly outperforms both. To further evaluate model performance, it 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 annotator coded the same sentences and we compared the results. The Cohen kappa (agreement corrected for chance agreement) for the human vs. machine coding was 88.6 %, which is higher than the frequently applied threshold of 80 % agreement. In other words, the ML model performed at least as well as a second human coder would be expected to do.

Examining the results, somewhat surprisingly, 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 quotes.

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

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.

4 Findings

In this section we discuss in turn the findings from our study, first validating the measure of success, then examining support for each hypothesis.

4.1 Membership

As a check on our measure of success (graduation from the Incubator), we compared the number of developers in graduated and retired projects (active developers were those who had participated on the mailing list). The results are shown in Table 3. As the table shows, graduated projects had more than twice as many developers active on the mailing list as did retired projects. The differences are so large than a statistical test of significance seems superfluous (for doubters, a Kruskal-Wallis test, chosen because the data are not normally distributed, shows a statistically significant difference in the number of developers between graduated and retired projects, $p = 0.001$). This result provides evidence for the validity of graduation as a measure of project community health.

Table 3. Mean number of developers by project status and developer role

Project status	Core	Peripheral
Graduated	31.6 (19.4)	82.2 (102.4)
Retired	13.9 (9.3)	25.4 (18.3)

N = 74. Standard deviations in parentheses.

Hypothesis 1 was that successful projects would have more communication. As shown in Table 4, this hypothesis is strongly supported, as graduated projects have many times more messages sent than retired projects during the incubation process ($p = 0.0001$).

Table 4. Mean number of project messages by project status and developer role

	Core	Peripheral
Graduated	8265 (8878)	7306 (8908)
Retired	1791 (1805)	1652 (2058)

N = 74. Standard deviations in parentheses.

Hypotheses 2a and 2b were that core and peripheral members respectively would communicate more in successful projects than in unsuccessful projects. The differences in Tables 4 and 5 show that these hypotheses are supported ($p = 0.0001$ for core and $p = 0.0001$ for peripheral members for overall message count in graduated vs. retired projects, and $p = 0.0011$ and $p = 0.0399$ for messages per developer).

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

	Core	Peripheral
Graduated	239 (191)	109 (119)
Retired	107 (200)	47 (92)

N = 74. Standard deviations in parentheses.

Hypothesis 3 was that core members would communicate more than peripheral members. From Table 4, we can see that in fact in total core and peripheral members send about the same volume of messages in both graduated and retired projects. However, there are fewer core members, so on average, each sends many more messages on average, as shown in Table 5 ($p = 0.0001$).

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

	Core	Periphery
Graduated	22 (18)	6 (5)
Retired	12 (8)	4 (5)

N = 74. Standard deviations in parentheses.

Hypothesis 4 was that core members would use more inclusive pronouns than peripheral members. Table 6 shows the 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 ($p = 0.0001$).

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

	Core	Periphery
Graduated	7.6 (3.4)	5.5 (2.2)
Retired	9.3 (5.)	5.3 (3.2)

N = 74. Standard deviations in parentheses.

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. From this table, we can see that the mean percentage of messages sent by core developers that include an inclusive pronoun is higher than for peripheral members ($p = 0.001$).

Hypotheses 5a and b were that there would be more use of inclusive pronouns by core and peripheral members respectively in successful projects. From Table 6, this hypothesis seems supported for core members at least, but note that successful projects have more communication overall. Examining Table 7 suggests that there is in fact slightly more proportional use of inclusive pronouns by core members in unsuccessful projects, but no difference in use by peripheral members. However, neither difference is significant using a KW test, meaning that Hypothesis 5 is not supported.

Finally, to assess which of the factors we examined are most predictive of projects success, we applied a stepwise logistic regression, predicting graduation from the various measures of communication developed (e.g., total number of message by developer role, mean number, percentage of message with inclusive pronouns). Our first regression identified only one factor as predictive, the number of core members. This result can be expected, as we argued above that the number of core members can also be viewed as a measure of community health. A regression without counts of members identified the total number and the mean number of messages sent by core members as predictive, with mean having a negative coefficient. (The R^2 for the regression was 33 %.) This combination of factors does not provide much insight as it is essentially a proxy for developer count: greatest when there are a lot of messages but not many messages per developer, i.e., when there are more developers.

5 Discussion

In general, our data suggest that successful projects (i.e., those that successfully built a community and graduated from incubation) have more members and a correspondingly large volume of communication, suggesting an active community. As expected, core

members contribute more, but overall, the message volume seems almost evenly split between core and peripheral members, suggesting that both roles play an important part in projects. These results demonstrate the importance of interaction between and the shared responsibilities of core and peripheral members.

As expected, core members do display somewhat greater ownership of the project, as expressed in the use of inclusive pronouns, but counter to our expectations, the use of inclusive pronouns did not distinguish successful and unsuccessful projects. A possible explanation for this result is a limitation in our data processing: we determined developer status (core or periphery) based on committer lists from the project website collected at the time of analysis. This process does not take into account the movement of developers from periphery to core (or less frequently, from core to periphery). It could be that in successful projects, active peripheral members (i.e., those using more inclusive pronouns) are invited to join the core, thus suppressing the average for peripheral members.

6 Conclusions

The work presented here can be extended in many ways in future work. First, as noted, developers may change status during the project. The results would be more accurate if they took into account the history of when developers became committers to correctly assign their status over time. Obtaining such historical data is challenging but not impossible. Second, the ML NLP might be improved with a richer feature set [24], though as noted, the performance was already as good as would be expected from an additional human coder. Third, it would be interesting to examine the first few months of a project for early signs that are predictive of its eventual outcome. Fourth, it might similarly be possible to predict which peripheral members will become core members from their individual actions. Fifth, we can consider the effects of additional group maintenance behaviours from Wei et al. [21]. The Syracuse SOCQA project has had some success applying ML NLP techniques to these codes, suggesting that this analysis is feasible. Sixth, it is necessary to consider limits to the hypothesized impacts. 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. Finally, 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.

Despite its limitations, our research offers several advances over prior work. First, it examines a much large sample of projects. 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.

Acknowledgements. We thank the SOCQA Project (Nancy McCracken PI) for access to the coded sentences for training and Feifei Zhang for checking the coding results. SOCQA was partially supported by a grant from the US National Science Foundation Socio-computational Systems (SOCS) program, award 11–11107.

References

1. Amrit, C., van Hillegersberg, J.: Exploring the impact of socio-technical core-periphery structures in open source software development. J. Inf. Technol. **25**(2), 216–229 (2010)
2. Bagozzi, R.P., Dholakia, U.M.: Open source software user communities: a study of participation in Linux user groups. Manage. Sci. **52**(7), 1099–1115 (2006)
3. Barcellini, F., Détienne, F., Burkhardt, J.-M.: A situated approach of roles and participation in open source software communities. Hum.-Comput. Interact. **29**(3), 205–255 (2014)
4. Bonaccorsi, A., Rossi, C.: Why F/OSS can succeed. Res. Policy **32**, 1243–1258 (2003)
5. Crowston, K., Wei, K., Howison, J., Wiggins, A.: Free/Libre open source software development: what we know and what we do not know. ACM Comput. Surv. **44**(2), Article 7 (2012)
6. Crowston, K., Howison, J., Annabi, H.: Information systems success in free and open source software development: theory and measures. Softw. Process Improv. Pract. **11**(2), 123–148 (2006)
7. Crowston, K., Howison, J.: Assessing the health of open source communities. IEEE Comput. **39**(5), 89–91 (2006)
8. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of teams for Free/Libre open source software development. Inf. Softw. Technol. **49**(6), 564–575 (2007)
9. Crowston, K., Wei, K., Li, Q., Howison, J.: 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) (2006)
10. Dahlander, L., O'Mahony, S.: Progressing to the center: coordinating project work. Organ. Sci. **22**(4), 961–979 (2011)
11. Fang, Y., Neufeld, D.: Understanding sustained participation in open source software projects. J. Manage. Inf. Syst. **25**(4), 9–50 (2009)
12. Jensen, C., Scacchi, W.: Role migration and advancement processes in OSSD projects: a comparative case study. In: Proceedings of the 29th International Conference on Software Engineering (ICSE), pp. 364–374 (2007)
13. Jergensen, C., Sarma, A., Wagstrom, P.: The onion patch: migration in open source ecosystems. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 70–80 (2011)
14. Luthiger Stoll, B.: Fun and software development. In: Proceedings of the First International Conference on Open Source Systems, Genova, Italy, 11–15 July 2005
15. Park, J.R.: Interpersonal and affective communication in synchronous online discourse. Libr. Q. **77**(2), 133–155 (2007)
16. Park, J.-R.: Linguistic politeness and face-work in computer mediated communication, part 2: an application of the theoretical framework. J. Am. Soc. Inf. Sci. Technol. **59**(14), 2199–2209 (2008)
17. Rullani, F., Haefliger, S.: The periphery on stage: the intra-organizational dynamics in online communities of creation. Res. Policy **42**(4), 941–953 (2013)

18. Scialdone, M.J., Heckman, R., Crowston, K.: Group maintenance behaviours of core and peripheral members of Free/Libre open source software teams. In: Proceedings of the IFIP WG 2.13 Working Conference on Open Source Systems, Skövde, Sweden, 3–6 June 2009
19. Toral, S.L., Martínez-Torres, M.R., Barrero, Federico: Analysis of virtual communities supporting OSS projects using social network analysis. Inf. Softw. Technol. **52**(3), 296–303 (2010)
20. von Krogh, G., Spaeth, S., Lakhani, K.R.: Community, joining, and specialization in open source software innovation: a case study. Res. Policy **32**(7), 1217–1241 (2003)
21. Wei, K., Crowston, K., Li, N.L., Heckman, R.: Understanding group maintenance behaviour in Free/Libre open-source software projects: the case of fire and gaim. Inf. Manage. **51**(3), 297–309 (2014)
22. Yan, J.L.S., McCracken, N., Crowston, K.: 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, June 2014. http://nlp.stanford.edu/events/illvi2014/papers/mccracken-illvi2014.pdf
23. Yan, J.L.S., McCracken, N., Crowston, K.: Semi-automatic content analysis of qualitative data. In: Proceedings of the iConference, Berlin, Germany, 4–7 Mar 2014
24. Yan, J.L.S., McCracken, N., Zhou, S., Crowston, K.: Optimizing features in active machine learning for complex qualitative content analysis. Paper Presented at the Workshop on Language Technologies and Computational Social Science, 52nd Annual Meeting of the Association for Computational Linguistics Baltimore, MD, June 2014