

Coordination Dynamics in Free/Libre Open Source Software Development

Andrea Wiggins

Abstract Successful free/libre open source software (FLOSS) projects demonstrate an alternative model of software production that may yield novel approaches for application to other forms of virtual work. Coordination, or the activities through which interdependencies are managed (Malone and Crowston, 1994), is a key challenge in software development, and FLOSS practices seem to offer particular promise for improving complex distributed work efforts in other contexts (Crowston et al, 2005). A variety of coordination mechanisms observed in FLOSS can be evaluated for potential application to managing dependencies in mainstream software development and other forms of decentralized but interdependent virtual work, such as collaborative writing. This prospectus outlines an investigation into the dynamics of two specific coordination mechanisms used in FLOSS development, modularity and self-assignment to tasks, which will provide a way to monitor project health, as well as evaluating the common claim that modularity enables scalable self-assignment to tasks as projects grow. A longitudinal design is proposed to observe the effects of project growth on coordination, employing a multiple case study methodology based on archival secondary data from FLOSS repositories. Integrated quantitative and qualitative analysis will incorporate qualitative narratives of the coordination dynamics in each project to contextualize the correlational analysis of project statistics, code metrics, and content analysis coding.

Key words: open source software, coordination, dynamics, self-organization

1 Introduction

Successful free/libre open source software (FLOSS) projects demonstrate an alternative model of software production that may yield effective new approaches for ap-

Andrea Wiggins
Syracuse University, 337 Hinds Hall, Syracuse, NY 13244 USA e-mail: awiggins@syr.edu

plication to other forms of distributed work. Coordination, in the form of activities through which interdependencies are managed (Malone and Crowston, 1994), is a key challenge in virtual work and FLOSS practices seem to offer particular promise for insight into improving distributed work efforts in other contexts (Crowston et al, 2005). Software development is often characterized as uncertain, interdependent, and highly coordinated work; Yamauchi et al (2000) claim that the difficulty of software development comes from the frailty of coordination in large complex software products, and believe that the relatively high level of uncertainty requires ongoing flexible coordination. However, as an increasing number of studies note, FLOSS development demonstrates successful distributed software development through coordination mechanisms that strategically minimize interdependency (Mockus et al, 2002; Dinh-Trong and Bieman, 2005; Jensen and Scacchi, 2005).

A variety of coordination mechanisms observed in FLOSS can be evaluated for potential application to managing dependencies in mainstream software development and other forms of decentralized but interdependent virtual work, such as collaborative writing. This prospectus outlines an investigation into the dynamics of two specific coordination mechanisms used in FLOSS development, modularity and self-assignment to tasks, which will provide a way to monitor project health, as well as evaluating the common claim that modularity enables scalable self-assignment to tasks as projects grow.

1.1 Motivation

Coordination requirements in virtual teams can be significantly different from their traditional counterparts (Maznevski and Chudoba, 2000; Hinds and McGrath, 2006; Bell and Kozlowski, 2002; Watson-Manheim et al, 2002). For example, while some virtual teams may require greater coordination effort for synchronous activity, often around other coordination mechanisms such as design and planning, FLOSS teams show a relatively low simultaneity constraint, and their work practices have been structured to virtually eliminate the need for synchronous participation (Jensen and Scacchi, 2005).

The voluntary and often decentralized nature of the work in community-based FLOSS projects also affects their coordination efforts (Mockus et al, 2002; Dinh-Trong and Bieman, 2005). In contrast to most proprietary software development practices, which apply more bureaucratic and hierarchical approaches of managing assignment of tasks, self-assignment to tasks is a dominant coordination mechanism for managing the task assignment resource allocation dependency (Crowston et al, 2005; Mockus et al, 2002). This aspect of community-based FLOSS projects demonstrates a mode of production that is incongruous with the definition of coordination employed by Sagers (2004), which measured ability of project leaders to know where expertise was located within the project. Such a measure seems to assume that the role of leaders is to assign tasks, and while this assumption may

hold to some extent in very large FLOSS projects, it is unlikely to be an appropriate expectation for many community-based FLOSS projects.

1.2 General research question

Researchers are now investigating the role of coordination practices in FLOSS development, with studies typically focusing on a subset of specific coordination practices in detail, e.g. coordination in bug-fixing (Crowston and Scozzi, 2004), or restricted commit access (Sagers, 2004). Such research indicates that such coordination mechanisms are unlikely to be applied in isolation, but are rather employed in combination. I hypothesize that these combinations of coordination mechanisms, or coordination strategies, may represent common good solutions to general organizing problems in FLOSS projects. This leads to the corollary hypothesis that as FLOSS projects grow and mature, their stages of development will be reflected in the dynamics of the coordination strategies they employ over time. FLOSS project growth therefore presents an opportunity to apply a sociotechnical perspective to investigating the relationships between the dynamics of project development and coordination strategies, leading to the general research question:

How does project growth affect the social and technical structuring of work through coordination strategies?

1.3 Specific research question

While there are almost infinite variations of possible combinations of coordination mechanisms that may represent coordination strategies, the cultural and operational contexts of FLOSS development renders it most conducive to applying a limited variety of canonical coordination strategies. The close link between social and technical aspects of software development suggest that common coordination strategies will likely include both social and technical coordination mechanisms that shape the way the work is done. Technical coordination structures emerge as natural outgrowths of software development's work processes, intended or not; one of the clearest instantiations of technical coordination is evident in source code modularity. Deriving from the general research question, a more specific research question examines the relationship between growth and coordination strategies:

How do the dynamics of the relationship between the size of the core committer group and size of the code base affect technical coordination through code modularity and social coordination through self-assignment to tasks in community-based FLOSS projects?

This research question investigates the changing coordination strategies through interdependent social and technical coordination mechanisms, upon which project

growth is expected to have meaningful consequences. Code modularity represents an often-mentioned technical coordination mechanism which affects the structure of work and manages both task interdependency and task/subtask decomposition. According to Lehman's laws of software evolution, complexity increases in an evolving system unless efforts are made to mitigate it, which suggests that code complexity is a reasonable indicator of coordination effort. Self-assignment represents a social coordination mechanism to manage task assignment, the complexity of which is affected by project growth and the technical structure of the work (modularity). Project growth has a variety of potential operationalizations, and for the purpose of linking the concept of project growth to the selected coordination mechanisms, the number of committers and the size of the code base are measures of growth that are theorized to have a direct effect on both code modularity and task assignment (Mockus et al, 2002). The expected interrelationships between project growth, modularity, and self-assignment form a set of propositions:

Proposition 1: As the size of code base and core committer group increase, code modularity increases.

Proposition 2: As the size of code base and core committer group increase, self-assignment to tasks increases.

Proposition 3: As code modularity increases, self-assignment to tasks increases.

1.4 Conceptual framework

Coordination strategies manage dependencies by moderating the effects of scale, uncertainty and interdependency on tasks. The conceptual model in Figure 1 shows the hypothesized relationships that this design will test. The size of the active committer group and size of code base are variables that indicate the scale of coordination requirements; more effort will be required to coordinate a larger number of contributors and also to manage a growing code base. Code modularity is a strategic approach to structuring the work in a way that both reduces task interdependence and moderates task/subtask decomposition, addressing some aspects of uncertainty. As projects grow, self-assignment is expected to be a prevalent task assignment coordination mechanism.

The size of the code and committer community are already in operationalized forms in this conceptual model and reflect both social and technical evidence of project growth; modularity is equated with code complexity and interdependency measures from software code metrics, while task self-assignment will be determined through content analysis coding.

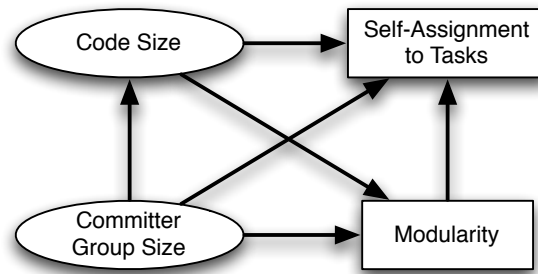


Fig. 1 Conceptual model of hypothesized relationships between the size of the code base, the size of the committer group, code modularity, and self-assignment to tasks.

2 Methods

Evaluating these propositions requires a longitudinal design in order to observe the effects of project growth on coordination. A multiple case study methodology will be employed, using archival secondary data from FLOSS repositories. Integrated quantitative and qualitative analysis will incorporate qualitative narratives of the coordination dynamics in each project to contextualize the correlational analysis of project statistics, code metrics, and content analysis coding.

2.1 Case selection

Due to the intensive effort required for content analysis coding, which is amplified by longitudinal observation sampling, three projects will be selected for comparison. Selection criteria include availability of data in the requisite repositories (see Section 2.2), project membership of at least 10 developers listed on the project's web site at the time of selection to allow for nontrivial coordination, with at least three releases and a project lifespan of at least 18 months at the time of data collection to provide sufficient data for multiple observations over time. While 18 months is the minimum project lifespan that would allow sufficient data for analysis, it is more likely that projects with at least 10 developers listed as affiliated with the project will have significantly longer histories than this minimum.

The selected cases will be projects that produce the same type of software, as product type may have an effect on code architecture that could affect the validity of comparisons of code modularity¹. The cases will be selected from moderately complex products that have a single primary package (although they may have ancillary

¹ An expert will be consulted to determine whether code modularity comparisons will also require that the projects use the same programming language. If this is the case, the strategy may shift to hold programming language constant while product type may vary.

packages) to disambiguate longitudinal observation sampling. In addition, projects with a continuous release cycle, while interesting cases of coordination strategy worthy of study in their own right, will be excluded because their coordination practices present a very different project management strategy than this research intends to investigate. Within the product type for case selection, the cases will additionally be selected from among the most downloaded software, with preference given to those that show evidence of ongoing and recent effort in the project's primary communication channels. Combined with the developer membership threshold and project release record and lifespan criteria, these projects are considered successful according to the criteria from Crowston et al (2003). Successful projects are desired for case selection as they are more likely to demonstrate effective coordination practices which have evolved over time.

2.2 Data

Observations will be sampled in a similar fashion to the method used by Crowston et al (2005); for each software release date for the primary package in the selected projects, developer email list messages will be collected for the preceding three weeks and subsequent week, based on the reasoning that more coordination will occur around a release. The combined case selection and data sampling requirements will therefore require a minimum sample of twelve weeks of email for each project, although some adaptation may be required if project release timing generates overlapping sample windows. Although the time period for observations is reduced to four weeks, this selection mechanism is expected to yield a total corpus several thousand messages for coding due to the multiple observations over time for each project.

If available through a repository, the email messages will be extracted from these sources and prepared for analysis; if the mailing lists are not already available in a usable form, a relatively small amount of additional time will be required to spider and parse the mailing list archives. Deductive content analysis of these messages will apply the task assignment mechanism coding scheme from Crowston et al (2005) to determine the relative frequency of self-assignment for each period of observations, extending the prior research by examining a new project sample and looking at changes in task assignment mechanisms over time.

Email lists are the chosen data source for several reasons, foremost because they are a primary venue for project communication and coordination. Tracker activity could also be useful to evaluate, but different analysis methods may be more appropriate for these data, and the results would be difficult to compare to prior research. IRC and CVS may also offer indicators of coordination activity, but IRC transcripts are not readily available and may also require substantially different ethical considerations for use as a data source. While coordination may be evident in commit comments, it is not clear that these data would represent assignment to tasks so much as task completion. Therefore although evidence of coordination may be

present in multiple communication venues, the scope of this study will be limited to analysis of email, as each venue has distinct characteristics and dynamics that merit differentiated analysis approaches (Wiggins et al, 2008).

Project statistics and code metrics will be retrieved from FLOSS repositories, and will be used for correlational analysis, along with the results of the content analysis coding. Project statistics will include the number of developers listed on the home page at the time of the release and the number of active contributors to the project (individuals posting to mailing lists, forums, trackers, or committing code) during the four week period around the release. These data will come from FLOSSmole and SRDA. Appropriate code metrics for evaluating modularity and code complexity will be selected by consulting the literature to choose the most appropriate measure available from FLOSSmetrics; code size counts (non-comment lines of code) will also be obtained.

The source data will already exist as public documents on the Internet, and as such are expected to be exempt from IRB review. However, to address the potential ethical issues arising from aggregating and reporting this information, all identities will be anonymized. In addition, direct quotations will be paraphrased such that they are altered in form but not content, to obfuscate the otherwise easy use of search engines to identify individuals by querying quoted phrases.

2.3 Analysis

The analysis will combine complementary statistical and qualitative elements for the final findings. Rank correlations and any other appropriate statistical tests for time series data will be applied to evaluate the relationships proposed by the propositions in Section 1.3. The results of the content analysis will be compared to those reported in Crowston et al (2005); the results are expected to be consistent with the prior findings for the first release, but demonstrate variation from the established result for later releases in each project. In addition, metrics for the scale of coordination effort, such as number of lines of code in the repository divided by the number of active committers, will be examined for potential application as a compound measure of degree of coordination required based on project growth.

The qualitative portion of the analysis will contextualize these relatively simplistic measures by developing narratives of the coordination dynamics in each project, highlighting potential causal factors for changes in task assignment mechanisms. Telling the stories of the evolution of work coordination practices will provide a richer frame for understanding and interpreting the statistical results, and will include nuance and detail that cannot be captured in summaries of content analysis coding or tables of correlation coefficients. These narratives will allow a more holistic and thorough comparison between the cases, and will improve the face validity of the statistical results.

2.4 Design compromises

Most research design decisions result in compromises between costs, difficulty and quality. The primary cost in this design is the intensive work required for content analysis coding for multiple release periods across multiple cases. This analysis method therefore limits the number of projects that can be sampled, and the sampling methods also use a smaller selection of messages for each release by comparison to prior work, through shrinking the size of the window around each release for which data will be collected and coded. These choices are intended to limit the scope of the research just enough to make it possible to complete the work in the allotted time frame, while minimizing impact on quality.

In order to improve the reliability of results, and also make it possible to complete the content analysis coding within a year, two raters will double-code messages until acceptable inter-rater reliability is achieved (at least 0.8) and will then code the data separately. Coding quality will be subject to ongoing evaluation through periodic reliability checks on randomly sampled double-coded messages to ensure that inter-rater reliability is maintained as coding progresses. In addition, careful project selection will help improve results: checking for data availability for all of the sources of data required for the full analysis and an initial data quality review will prevent late discovery of unforeseen data quality challenges.

2.4.1 Scaling content analysis

Automated coding software will be tested on the data, but the wide variation in expressions of task assignment makes this content analysis coding poorly suited to fully automated coding methods. Using an established coding schema suited to this precise context and medium of communication, however, will reduce the difficulty of the content analysis coding. It should also improve the comparability and quality of the results.

A “human-in-the-loop” semi-automated coding process will also be tested. In this process, once the coders have achieved suitable levels of agreement during an initial round of double-coding, simple text matching rules are generated and applied to achieve a high recall result set for messages that include elements often observed in coded messages. For example, searching on the phrase “I will”, and common variants, would retrieve many instances that are likely to be coded as self-assignment.

A high false positive rate is expected and desirable for this purpose, as the results would be independently reviewed by the coders for rapid verification of whether a task assignment mechanism is present in the message. This type of binary decision is faster than reviewing every message, and the risk of false positives is significantly ameliorated by the process of human review. The rate of false negatives can be tested by comparing the text retrieval rule results for the initial set of coded messages to the human-generated “gold standard” to demonstrate the recall and precision; this information will help determine whether to apply the semi-automated coding process for each coding category. If content analysis codes are applied as mutually exclusive

Acquire and prep data for analysis	Content analysis coding to agreement	Test semi-automated coding	Content analysis coding	Integrated analysis & writing
1 month	2 months	1 month	6 months	2 months

Fig. 2 Timeline for research completion.

message-level codes, then it would be possible to further optimize the process; although this method may introduce additional potential biases, the compromise may be acceptable in exchange for the concurrent potential to significantly increase the scale of analysis.

2.5 Timeline

The timeline for this research, illustrated in Figure 2, covers one year. Approximately three to four weeks will be spent on data collection and preparation for analysis at the beginning of the project; the required data should be relatively easy to obtain. Content analysis coding will occupy the following 9 months, with the initial coding phase expected to take approximately 2 months. Initial coding will cover approximately one-third of the initial sample of messages; these will be randomly selected from all projects. A subsequent period of one month will allow a trial of the human-in-the-loop semi-automated process; if this method is judged sufficiently rigorous and more efficient than exhaustive coding, it will be adopted so that additional cases can be sampled. Statistical analysis of coded data will follow the initial phase of coding and at the completion of the coding. The remaining two months will be spent on integrated analysis of the content coding results; correlational analysis of the project statistics, code metrics, and content coding; and construction of qualitative coordination narratives.

3 Threats to validity

Case selection bias and measurement error pose the strongest threats to validity in this design, among those shown in Table 1. Selection bias is somewhat ameliorated by purposive sampling, which selects cases according to criteria that maximize the likelihood for observing the phenomena of interest (e.g., data availability, release-

based sampling), and in this case, broad generalizability of the sample is not the intent of the study. Instead, the goal is to gain insight into FLOSS software development strategies as a potential source of novel approaches for adoption in other distributed work contexts.

Table 1 Threats to validity, potential effects, and ways the design addresses them.

Type of threat	Effect	Countermeasures
Selection	Limited sample for comparison reduces generalizability. Availability bias from using repository data systematically excludes some projects from consideration.	Careful case selection; test of semi-automated content analysis to expand sample size.
Experimenter bias	Use of archival data means neither demand nor expectance are likely to be a problem.	n/a
Measurement / Instrumentation	Project statistics and code metrics are reductive and abstracted. Content analysis for task assignment mechanisms is may not be reliable.	Project growth measures and code metrics are direct operationalizations of theorized concepts. Content analysis uses established schema appropriate to the study and uses a strategy to ensure inter-rater reliability.
Longitudinal	History effects may arise from external factors influencing code modularity and self-assignment.	Any evidence of external factors encountered during content analysis coding will be separately coded and included in the project narratives.
Mortality	Case selection and use of archival data minimizes potential mortality effects.	n/a
External validity / Generalizability	May only be generalizable to narrow range of FLOSS projects; limited sample size.	Purpose is to identify FLOSS-specific coordination dynamics and strategies, so the tradeoff is acceptable.

3.1 External validity

External validity indicates the degree to which the findings are applicable more broadly; this study has limited external validity, as the context of the study of coordination utilizes concepts and operationalizations specific to software development. Generalizability may be further limited by the data sources FLOSS research repos-

itories based on public data, which are able to provide data from a wide array of available sources, but are by no means comprehensive. As such, the realistic extent of generalizability of the results is to FLOSS projects having similar characteristics with respect to size, success, product complexity, and technical practices. The results may have some bearing on distributed software development outside of FLOSS, but are expected to generate insight into novel approaches from community-based FLOSS teams to apply in mainstream software development, and so significant differences between FLOSS teams and other software development teams are likely to be present. However, abstracting the variables that represent the scale of coordination effort required to complete work could potentially allow the application of the conceptual framework to other contexts that share similarly low simultaneity constraints and relatively high task assignment resource allocation requirements.

3.2 Internal validity

The most significant problem for internal validity is the potential for measurement error. The design attempts to balance simple, direct measures of structural characteristics with more nuanced, content-based evaluation of project email archives. While the interval of observations for each time series period provides only single, summative measures of the technical coordination structures, the content analysis coding will provide more detailed and finely grained observations. Code metrics such as modularity and lines of code are objective, and their quality and efficacy can be evaluated and validated with existing algorithms. The number of committers on a project can be measured in multiple ways, so the study will collect two measures to compare for use in the final analysis: the number of developers listed in the project description (or alternately, the number of developers with CVS access, if the data are available) and the number of active committers, who are the individuals committing code during the period of time between releases.

In order to monitor the potential for the use of restriction of commit access, which is a coordination mechanism that would influence interpretation, the ratio of developers to active contributors will be calculated for each release time period prior to conducting content analysis. As mentioned in Section 2.2, active contributors are identified by participation in any email list, forum, tracker, or code repository. If restricting commit access is a practice that emerges as a significant factor that may threaten the validity of the interpretation of the number of committers to other variables, a new case can be selected if the effect is isolated, or in the event that two or three of the cases show evidence of restricting commit access, the theoretical model may require revision to include the additional variable.

Finally, the validity of the content analysis coding will be improved through the efforts of two independent content analysis coders, who will use the strategy outlined in Section 2.4 to manage the ongoing reliability of the coding. In addition, the design utilizes an existing coding schema drawn from the literature, which lends

validity through the direct applicability of the task assignment mechanism schema from Crowston et al (2005) to the purposes of the proposed study.

3.3 Expected contributions

The propositions in Section 1.3 outline the expected outcomes for the correlational analysis. This research advances a more process-oriented theory that explains coordination strategies as an outcome the scale and interdependency of the work, however, so certain time series trends are expected to emerge in the projects selected for case study comparison. At the first release, the projects are expected to demonstrate similar proportions of task assignment mechanisms to those previously observed, with self-assignment making up just over 50% of the task assignment instances in the sample.

Over time, the size of the code base is expected to rise more rapidly than the number of developers, but as the number of developers increases, so should code modularity. These effects would generate conditions that are theorized to be more favorable to self-assignment, so if the projects are successfully managing the increasing scale of their activities, the frequency of self-assignment should at least remain the same, but would ideally increase. It is also possible that there are transition points at which coordination strategies change to adapt to a more effective set of mechanisms for the current structure of interdependencies; these might be preceded by fluctuations in task assignment mechanisms as the project seeks a new state of equilibrium with respect to coordination efforts.

This research will extend the current scholarship on coordination in FLOSS teams by implementing a dynamic analysis of coordination mechanisms which are theorized to be closely linked. The study will evaluate the common claims of researchers about the relationship between code modularity, community size, and self-assignment to tasks for a small selection of projects. The longitudinal aspect of the design will permit the observation of effects of project growth on coordination through both technical and social structures.

In addition, the study is expected to reproduce prior results on task assignment at the first release of a software package, and will make a novel contribution by examining changes to task assignment over time in successful FLOSS projects. Taken together, the results for the individual cases will provide the basis for a general model of FLOSS coordination dynamics, to which other projects can then be compared to evaluate potential for ongoing project success. Finally, the goal of the research is to develop insights into FLOSS coordination that may prove appropriate for application to distributed coordination problems in mainstream software development. This pragmatic contribution would help distributed software development teams in a variety of contexts to plan optimal coordination strategies for managing interdependencies on an increasing scale of complexity.

References

- Bell B, Kozlowski S (2002) A typology of virtual teams: Implications for effective leadership. *Group & Organization Management* 27(1):14–49
- Crowston K, Scozzi B (2004) Coordination practices within floss development teams: The bug fixing process. *Computer Supported Activity Coordination*
- Crowston K, Annabi H, Howison J (2003) Defining Open Source Software Project Success. In: *Proceedings of the 24th International Conference on Information Systems (ICIS 2003)*
- Crowston K, Wei K, Li Q, Eseryel U, Howison J (2005) Coordination of free/libre open source software development. In: *Proceedings of the International Conference on Information Systems (ICIS 2005)*, Las Vegas, NV, USA
- Dinh-Trong T, Bieman J (2005) The FreeBSD Project: A Replication Case Study of Open Source Development. *IEEE Transactions on Software Engineering* pp 481–494
- Hinds P, McGrath C (2006) Structures that work: social structure, work structure and coordination ease in geographically distributed teams. In: *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, ACM Press New York, NY, USA, pp 343–352
- Jensen C, Scacchi W (2005) Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans. org Open Source Software Development Community. In: *System Sciences, 2005. HICSS'05. Proceedings of the 38th Annual Hawaii International Conference on*, pp 196b–196b
- Malone T, Crowston K (1994) The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR)* 26(1):87–119
- Maznevski M, Chudoba K (2000) Bridging Space Over Time: Global Virtual Team Dynamics and Effectiveness. *Organization Science* 11(5):473–492
- Mockus A, Fielding R, Herbsleb J (2002) Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11(3):309–346
- Sagers G (2004) The influence of network governance factors on success in open source software development projects. In: *Proceedings of International Conference of Information Systems*, Washington, DC
- Watson-Manheim M, Chudoba K, Crowston K (2002) Discontinuities and continuities: a new way to understand virtual work. *Information Technology & People* 15(3):191–209
- Wiggins A, Howison J, Crowston K (2008) Social Dynamics of FLOSS Team Communication across Channels. In: *Proceedings of the Fourth International Conference on Open Source Systems: Open Source Development, Communities and Quality*, pp 131–142
- Yamauchi Y, Yokozawa M, Shinohara T, Ishida T (2000) Collaboration with Lean Media: how open-source software succeeds. In: *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, ACM New York, NY, USA, pp 329–338