Coordination Across Open Source Software Communities: Findings from the Rails Ecosystem

Alexander Eck

University of St.Gallen, Institute of Information Management, St.Gallen, Switzerland
alexander.eck@unisg.ch

Abstract. While coordination of work within open source software (OSS) communities is well-researched, it is virtually unknown how work is coordinated across community boundaries. However, as OSS projects are often part of a larger digital ecosystem of interdependent artifacts and communities, cross-community coordination is a pertinent topic. We turn to the ecosystem around Ruby on Rails to empirically explore this research gap. To this end, we scrutinize 96 coordination episodes among five interrelated OSS projects and identify four cross-community coordination mechanisms: adaptation, upgrading, positioning, and departure. Each mechanism describes a distinct and stable arrangement to integrate contributions across community borders. After presenting our findings, we reason about the significance of the results on explaining generative change in digital ecosystems.

Keywords: cross-community coordination, open source software, digital ecosystems, generative change, case study

1 Introduction

The open source software (OSS) model of designing and changing complex artifacts based on a publicly accessible codebase [1] has created new forms of coordinating which have been investigated thoroughly during the last two decades [2]. Despite this work, little is known about coordination *across* communities of interdependent artifacts, as opposed to coordination *within* the community around an individual OSS artifact [3]. Cross-community coordination is of practical importance, because many OSS artifacts are coupled with other artifacts to form larger digital ecosystems and as such their communities have to interact [4]. One example is the ecosystem around *Ruby on Rails*, a web application framework. Thousands of add-on modules extend utility of this artifact [5]. Cross-community coordination is also of theoretical relevance, as it provides an apt setting to study the phenomenon of generative change, that is unanticipated change produced by contributions from broad and varied audiences [6], which is believed to drive innovation in the digital age [7]. Therefore, we ask:

How do open source software communities coordinate with other communities to change their respective artifacts within a shared ecosystem?

OSS communities usually introduce change through granular, incremental steps [8]. We harness this mode of working to empirically inquire coordination episodes. By recognizing patterns among these granular episodes, we identify mechanisms with which actors from different communities integrate their respective contributions – that is, we identify cross-community coordination mechanisms that lead to change, or potential for change, in OSS artifacts.

Our findings from five communities that are part of the *Ruby on Rails* ecosystem suggest four mechanisms. First, cross-community coordination reflects the hierarchy between artifacts. If one artifact serves as the basis of the second artifact, it is likely that the community of the latter readily seeks to *adapt* to any relevant change of the outside artifact. Second, communities anticipate fundamental changes to an artifact they depend on by contemplating possible consequences and *upgrading* their artifact accordingly. Third, cross-community coordination often revolves around locating and eliminating a design flaw that is identified through artifact coupling. The communities work together to *position* the appropriate location for fixing the design flaw. Fourth, members of an outside community may argue for a *departure* from existing artifact design, based on experiences within the context of their own artifact.

The remainder of this paper is organized as follows. We review the relevant literature on coordination across open source software communities. We then present the research design and findings of the empirical study, followed by a discussion how the findings fit into the broader topic of generative change in digital ecosystems. The paper concludes with a summary of main limitations and directions for further research.

2 Background

2.1 Coordination, Coordination Episodes, and Coordination Mechanisms

Coordination refers to the extent to which actors that need to integrate their respective contributions due to interdependencies do so consistently and coherently [9]. A major purpose of an *OSS project*, i.e. an OSS artifact and the community supporting it, is to advance artifact design via source code contributions [10]. Therefore, a large part of coordination consists of managing change in software artifacts through resolving interdependencies that emanate from source code contributions [11]. Consequently, this paper investigates coordination episodes during which possible source code contributions are discussed and integrated. A *coordination episode* is a logically connected series of activities with a trigger-activities-resolution structure [12]: a *trigger* that prompts coordination need; sequential *coordination activities* between actors during which knowledge is exchanged and source code contributions are discussed; and an eventual *resolution* whether and what kind of change is implemented. We can expect that over time, OSS communities develop stable arrangements to coordinate, or distinct *coordination mechanisms* [13]. It is reasonable to believe that coordination mechanisms exist also when two OSS communities repeatedly interact to integrate contributions across community boundaries, which makes it possible to identify mechanisms for cross-community coordination.

2.2 Coordination Mechanisms Across Communities of Open Source Software

Often, a software artifact is coupled with other artifacts to enable and augment its own capabilities [e.g., 14]. As such, it is typically part of a larger *digital ecosystem*, or a set of interdependent, co-evolving artifacts and the communities supporting them [4]. Artifacts belonging to an active ecosystem commonly change, which induces the regular need to integrate contributions originating from outside communities. Such *cross-community coordination* is difficult, because the OSS projects making up a digital ecosystem are heterogeneous and distributed, both in terms of resources and in terms of control [cf. 15]. For example, social practices how artifacts are changed may differ between communities, and whether an outside artifact changes so that depending artifacts are forced to be adapted is ultimately beyond the control the focal community.

Extant research on cross-community coordination mainly highlights the role of boundary spanners, i.e. actors who participate in multiple projects and who are capable to integrate heterogeneous contributions and reconcile dependencies between distributed artifacts [16–19]. Hence, boundary spanners coordinate by leveraging their personal knowledge, not through processes that span actors in disparate communities.

Beyond the attention given to the boundary spanning role, there is little research on cross-community coordination. This might be attributed to the property of software artifacts to be loosely coupled via defined interfaces [cf. 20]. Because interfaces codify the rules and protocols of exchange, they commonly avoid the need for direct interaction [21]. Yet, interfaces neither resolve all interdependencies nor do they remain unchanged over time [cf. 22]. For those situations in which coordination is necessary, we found weak cues for two distinct coordination mechanisms in the literature, which we label *adaptation* and *departure*. First, a community may adapt its artifact when an interdependent outside artifact changes, which is a rather transactional mechanism: a community accepts the change produced by another community and adapts accordingly [22–24]. Second, a more involved process is to advocate a departure from existing artifact design. This mechanism exploits the affordance of OSS communities to receive contributions from outsiders: a community influences another one to deviate from its existing artifact design and implement a change, usually one that helps the outsiders themselves [22]. Overall, our knowledge of cross-community coordination is limited. In the following, we report on an exploratory case study, addressing this research gap.

3 Research Design and Methods

We conducted an explanatory multiple case study of cross-community coordination in the digital ecosystem around Ruby on Rails (or simply *Rails*), an artifact that simplifies development of web applications written in the Ruby programming language. The goal of this examination was to describe and explain how disparate communities coordinate to change their OSS artifacts in light of interdependencies. Our choice of the Rails ecosystem was based on three main considerations. First, Rails was designed to be coupled with other artifacts written in the Ruby programming language – *Ruby gems* in jargon – which created the potential for cross-community coordination needs. Second, with its history reaching back to 2004 [25], both the codebase and the community of

Rails, and in extension many related projects, can be assumed to be mature. This was important to us, because we did not intend to trace the dynamics of how coordination mechanisms emerge, but rather identify established coordination mechanisms. Third, through reconstructing many digital ecosystems, including the Rails ecosystem [26], we noticed that the Rails artifact maintained an unusually large number of couplings with other artifacts. This made Rails an ideal starting point to investigate cross-community coordination in OSS settings.

3.1 Data Collection

We selected a sample of five OSS projects that are part of the Rails ecosystem, listed in Table 1, based on four major considerations. First, the project had to be hosted on Github, a popular service for collaborative OSS development, as we collected empirical data from there. Second, we wanted to ensure that the projects had many contributors, measured by *Github forks* [27], to increase the chances of observing coordination. Third, we chose popular projects, measured by *Github stars* accumulated [28]. As an added benefit, these criteria ensured that the selected projects were relatively mature, because it takes time to accumulate a sizable number of forks and stars.

Table 1. Artifacts included in the case study

Artifact	*Description*	*Forks*	*Stars*	*Ep*	*First activity*	*Last activity*
Ruby on Rails (04 Nov 2008)	Framework for web application development	13,073	32,195	74	24 Aug 2009	06 Jul 2016
Devise (16 Sep 2009)	Provides advanced authentication functionality	3,499	15,576	38	05 Apr 2010	29 Jun 2016
Activeadmin (15 Apr 2010)	Simplifies creation of admin interfaces for web apps	2,497	6,763	41	01 Aug 2011	11 Jul 2016
Kaminari (06 Feb 2011)	Provides pagination functionality for web apps	779	5,909	17	04 Mar 2011	08 Jan 2016
Formtastic (07 Apr 2008)	Domain-specific language for designing web-based forms	604	4,894	22	24 Aug 2009	11 Jul 2016

Artifact	*Artifact name (in brackets: day artifact became available on Github)*
Forks	*Number of forks, a measure of contributing community size (as of 15 Jul 2016)*
Stars	*Number of stars, a measure of artifact popularity (as of 15 Jul 2016)*
Ep	*Total number of episodes with artifact participation (after exclusion criteria)*
First/last activity	*Start day of first episode and end day of last episode with artifact participation*

Fourth, based on our knowledge of projects belonging to the Rails ecosystem [26] and employing self-written software, we systematically searched the discussion threads of projects in the Rails ecosystem, with data ranging from 29 Oct 2007 to 01 Jul 2016. Specifically, we captured discussions during which an outside project was referenced, and took this referencing as cue that two artifacts were coupled, which potentially led to the need of repeated cross-community coordination [3]. Consider this example: In June 2011, an issue on the *Devise* discussion board triggered a coordination episode: *"I have just updated devise 1.4.0 => 1.4.1 in my project, and now it throws an exception"*. After some causal theorizing, the root cause was identified: *"it's broken by* *this commit in Rails: rails@0ca69ca"*. Notably, a reference to Rails (that follows a well-defined syntax) was included. Less than two hours after the initial issue description, Devise was adapted to accommodate the change made by the Rails community, which resolved the episode.

We counted the number of outside references, and retained pairs of projects that shared many references. For example, we counted 109 instances in which either Rails or Devise referenced the other project. In total, the five OSS projects that we included in the case study shared 394 references between them. We then manually inspected the full discussion threads that contained those references. A discussion thread on Github typically revolves around a specific and usually self-contained *issue* for which coordination is needed to sort it out [29]. In our data sample, we observed that an issue discussion may spill over to another project, or that it may span multiple threads within the same project. Therefore, we followed the activities pertaining an issue even if it spanned multiple discussion threads and grouped them together. Overall, we collected 245 grouped discussion threads, which we regarded as tentative empirical evidence for cross-community coordination episodes, our unit of analysis. To ensure that the data indeed represented episodes of cross-community coordination, we inductively elaborated a list of exclusion criteria. For example, we excluded an episode if it was still ongoing at time of data collection. After applying the exclusion criteria, we were left with 96 episodes of cross-community coordination.

3.2 Construct Operationalization

The selected episodes served as empirical basis for analyzing how cross-community coordination is carried out in the Rails ecosystem and what effect it has on the artifacts involved. Our unit of analysis were individual coordination episodes. Therefore, we needed to operationalize the trigger of an episode, individual coordination activities, and the eventual episode resolution, as summarized in Table 2.

To operationalize the *trigger* construct we relied on open coding and axial grouping [30], eliciting five possible triggers from our data.

As for *coordination activities*, we drew upon the coding book of [29], which distinguishes between knowledge integration and direct implementation. *Knowledge integration* comprises activities needed to compile and integrate knowledge among the involved actors. *Direct implementation* covers activities needed to create and evaluate source code that may change the artifact. Just as in [29] our data set showed other activities as well, such as acknowledgment (e.g., *"You, sir, are awesome."*) which we did not analyze as they were not material to our research question.

Each coordination activity was performed by an actor, whose *actor status* we classified as either insider or outsider following a simple heuristic: We counted how often an actor contributed to each community across our full data set and made her member of the community to which she contributed most. Then we compared assigned membership of this actor (e.g., Rails) with the community in which she performed a coordination activity (e.g., Devise). Matching pairs were coded as *insider*, mismatches were coded as *outsider*.

Table 2. Operationalization of constructs and number of data points per construct

Construct	*Operationalization*	*Count*
Trigger	Identified trigger: issue, not offering a possible root cause (N=16); issue, offering a possible root cause (49); anticipated outside change may require change in own artifact (18); outside artifact offers new functionality (3); change proposal (10).	96
Coordination activity	Discussion entry that was coded as either knowledge integration (709) or direct implementation (599).	1,308
Actor status	Actor was either insider (1,178) or outsider (130) of the community in which she performed the coordination activity.	1,308
Resolution	Identified resolution: business rules change (47); software properties change (29); workaround found (5); no change (15).	96
Changed artifact	Episode was resolved by changing: …upstream artifact (16); …downstream artifact (53); …both artifacts (7); none (20).	96
Episode length	Number of activities in a coordination episode.	96
Episode duration	Duration of coordination episode.	96
Actors involved	Number of different actors who participated in an episode.	96
Files involved	Number of files for which source code changes were discussed.	96
KI ratio	Share of knowledge integration activities per episode.	96
OA ratio	Share of activities performed by outsiders per episode.	96

KI: knowledge integration
OA: outsider activity

A coordination episode in our context resolves by (not) implementing a source code change. To describe the *resolution* in more detail we were interested to learn also about the kind of (non-)change that resolved the episode. Therefore, we drew upon [31], who distinguish two types of source code change: change of *business rules* are changes to the behavior of a software artifact such as fixing a design flaw, whereas change of *software properties* alters nonfunctional metrics of a software artifact such as increasing the readability of source code. Furthermore, we knew from prior literature that agreeing on a *workaround*, i.e. a temporary and localized solution that does not change the artifact, was yet another way to resolve a coordination episode [32].

In addition, we captured which of the two artifacts involved in a coordination episode were changed: upstream, downstream, both, or none, following common terms in software engineering [e.g., 22]. An *upstream* artifact is one on which others depend, while the depending artifact is called *downstream*. From manual inspection of the five artifacts in our case study we established the upstream/downstream relationships, depicted in Figure 1. For example, Rails is an upstream artifact to Devise. By analogy, the Rails community is upstream to the Devise community.

We elicited distinct cross-community coordination mechanisms through identifying patterns across coordination episodes. To support this process we selected six additional constructs that characterized the episodes as a whole [cf. 12]. *Episode length* captures the number of coded activities in a coordination episode. *Episode duration* measures how much time passed between the first and the last activity in a coordination episode. *Actors involved* counts how many different actors were involved in an episode. *Files* *involved* counts the number of files for which source code changes were discussed during the episode. *Knowledge integration ratio* sets the number of knowledge integration activities in relation to the number of all activities in an episode. Finally, *outsider activity ratio* measures the percentage of activities that outsiders performed.

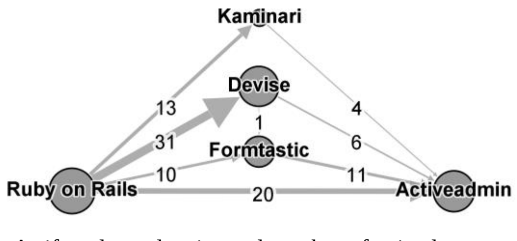


Figure 1. Artifact dependencies and number of episodes per artifact pair

3.3 Data Analysis

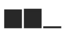
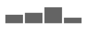
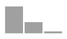
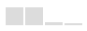
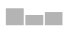
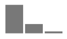
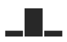
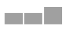
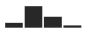
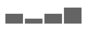
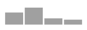
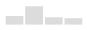
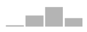
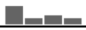
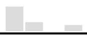
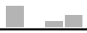
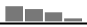
Data analysis consisted of two logical stages of coordination episode coding and coordination mechanism identification, through which we moved back and forth as we learnt more about the empirical setting. First, we *coded coordination episodes* according to the scheme summarized in Table 2. It struck us how straightforward the coding scheme could be applied. For example, when source code was changed it was easy to see whether functionality was added or fixed, readability was increased, etc. We attribute this to two reasons: In OSS communities it is customary to carry out fine-grained activities so that other community members can easily comprehend them [8]. In addition, the coordination episodes were usually focused on solving the specific issue at hand and did not divert into other directions. This demonstrated a certain level of discipline across the studied communities, which was an indicator that stable arrangements to coordinate, i.e. coordination mechanisms, existed.

Second, we *identified coordination mechanisms* in cross-community settings by seeking commonalities and differences between the individual episodes and matching them with preliminary coordination mechanism descriptions. We would switch to this stage of analysis whenever we identified tentative patterns or peculiarities that were potentially helpful to meet our research objective. As prior research mentioned *adaptation* and *departure* mechanisms, these were our natural starting points for candidate mechanisms, for which we wrote initial descriptions. We found evidence for these mechanisms and discovered two additional coordination mechanisms, which we eventually labeled *upgrading* and *positioning*.

4 Findings: Mechanisms of Cross-Community Coordination

Through analyzing 96 episodes of cross-community coordination, our empirical analysis unearthed four mechanisms that could comprehensively explain the observed coordination work. Summary statistics for each mechanism is provided with Table 3. In what follows, we show how each mechanism describes a different arrangement of cross-community coordination.

Table 3. Cross-community coordination mechanisms

	Trigger		*Activities*						*Resolution*		
Mechanism	*N*	*I*	*Length*	*Duration*	*Actors*	*Files*	*KI ratio*	*OA ratio*	*C*	*B*	*U*
Adaptation	43	72%							91%	42%	0%
Upgrading	7	14%							86%	43%	0%
Positioning	33	100%							91%	76%	67%
Departure	13	0%							8%	8%	8%

N	Number of episodes attributed to mechanism
I	Percentage of episodes that were triggered by an issue (with or without root cause analysis)
Length	Number of activities per episode (histogram categories: 2-5; 6-20; 21-50; 51-92)
Duration	Duration per episode in days (<1; 1-10; 10-100; 100-1,297)
Actors	Number of different actors involved per episode (2-3; 4-8; 9-16; 17-24)
Files	Number of source code files involved per episode (<2; 2-4; 5-9; 10-87)
KI ratio	Share of knowledge integration activities per episode (<25%; 25%-50%; 50%-75%, >75%)
OA ratio	Share of outsider activities per episode (<5%; 5%-50%; 50%-95%, >95%)
C	Percentage of episodes that resolved with an artifact change
B	Percentage of episodes that resolved with a change of business rules
U	Percentage of episodes with change in upstream artifact

4.1 The Adaptation Mechanism

Our data analysis shows how one community seeks to adapt its artifact to changes in an upstream artifact as to restore functionality or utilize new functionality, which we define as *adaptation* mechanism. Typically, a coordination episode of this type is triggered by a malfunction, and its root cause is found to be a recent change in an upstream artifact. This instigates change in the downstream artifact, which resolves the initially reported issue. The adaptation mechanism leverages hierarchy of artifact couplings as its logic to coordinate work across communities. The upstream community introduces changes without reaching out to downstream communities. Instead, it assumes that downstream artifacts will be adapted to restore compatibility. This is an efficient arrangement for the upstream community, because it mitigates the costs of verbal exchange with outside communities. It is also efficient for the downstream communities, given that most adaptation episodes resolve after few activities and involve few actors. Furthermore, the low number of files involved in an adaptation episode indicates that the necessary adaptations are rather small, which in turn implies that the changes to the upstream artifact cannot be extensive as well.

4.2 The Upgrading Mechanism

Our data analysis reveals how one community seeks to understand major changes in an upstream artifact and how it upgrades the downstream artifact to appropriate these changes, which we define as *upgrading* mechanism. Typically, a coordination episode of this type is triggered by upcoming major changes (e.g., changes to interfaces) that the community of an upstream artifact announced. After discussing the implications for its own artifact, the downstream community systematically changes its artifact to account for these implications. The upgrading mechanism resembles the adaptation mechanism, as it follows the same logic of artifact hierarchy. However, it is distinct in that coordination activities typically commence in anticipation of change to the upstream artifact, as opposed to a usually reactive adaptation episode. This mechanism is efficient for the upstream community, as indicated by the very low share of outsider activities. Thus, the coordination burden is mostly on the downstream community. Not only must it implement many changes to its artifact, it also has to spend effort on collating and integrating knowledge, as evidenced by the comparatively high knowledge integration ratio of most upgrading episodes.

4.3 The Positioning Mechanism

Our data analysis surfaces how two communities seek to track down an issue observed from coupling their respective artifacts and introduce change at the position they jointly agree on, which we define as *positioning* mechanism. Typically, a coordination episode of this type is triggered by an issue that is observed from the combination of upstream and downstream artifacts. Both communities then investigate why the issue happens, debate which artifact should be changed and implement the changes deemed appropriate. The positioning mechanism is distinct as it always sees one community reaching out to another due to an issue it cannot solve on its own. Creating a shared understanding between the two communities is a major concern, suggested by the comparatively high knowledge integration ratio combined with a high outsider activities ratio. A positioning episode mainly increases artifact quality, because a previously unknown issue is identified and resolved: 76% of all changes were to the functionality of the artifact, usually fixing a design flaw. Remarkably, the main beneficiary is the upstream artifact, as 67% of all positioning episodes in our data resolved with changes to the upstream artifact. This suggests that a growing pool of downstream artifacts effectively increases the number of collaborators who contribute with technical expertise to the upstream artifact, which extends the community borders.

4.4 The Departure Mechanism

Our data analysis exposes how an actor, based on experiences made with a coupled artifact, proposes a change and the focal community decides whether and how to implement the change despite it being a departure from existing design, which we define as *departure* mechanism. Typically, a coordination activity of this type has few activities, involves just a couple of actors, and no or just one source code file. Usually, the initially proposed departure of artifact design is not accepted by the focal community. The departure mechanism incorporates what is commonly considered a hallmark of open source software: anybody may modify an OSS artifact and see this change become part of the 'official release' if the community deems it beneficial [3]. However, in our case study of five rather mature OSS projects, fewer than 15% of all episodes could be attributed to the departure mechanism. What is more, all episodes but one resolved with a rejection of the proposed change by the focal community, and typically did so after few coordination activities. This suggests that OSS communities managing mature artifacts are reluctant to depart from existing artifact design.

5 Discussion: Coordination as Means of Generative Change

By analyzing how the communities of five interdependent OSS projects in the Rails ecosystem resolve coordination needs stemming from changes in coupled artifacts, we identified four cross-community coordination mechanisms, namely *adaptation*, *upgrading*, *positioning*, and *departure*. Each mechanism describes a mode of integrating contributions from coupled OSS projects that instigate change in the focal artifact. Thus, cross-community coordination organizes the co-evolutionary production of change caused by broad and varied audiences absent of central control, that is cross-community coordination is a means of *generative change* in a digital ecosystem [6].

The strength of a generative system lies in its capacity to allow contributions from heterogeneous actors, and to turn these contributions into productive changes. Because they are the result of a creative dialogue between insiders and outsiders, some of these changes are unanticipated and innovative [6]. Due to perpetual incompleteness of digital artifacts [33] and their ability to be coupled with other incomplete artifacts to form new combinations and variations, digital ecosystems produce constant co-evolutionary changes. Arguably, this dynamic is a main driver of digital innovation [7].

To this end, the identified cross-community coordination mechanisms help us better explain and rationalize digital innovation processes in complex digital ecosystems, particularly in an OSS setting. The adaptation and upgrading mechanisms show how upstream OSS projects effectively influence the evolutionary trajectory of downstream projects. In turn, the positioning mechanism demonstrates how autonomous, yet interdependent OSS communities create synchronization opportunities to sort out design flaws for mutual benefit. Finally, the departure mechanism suggests that mature OSS projects have a strong sense of direction, which reduces the propensity of radical design changes. Overall, the existence of coordination mechanisms that operate across OSS communities are further proof of the self-organizing capability of heterogeneous systems characterized by distributed resources and distributed control [15].

6 Limitations and Conclusion

The contributions presented here are naturally limited by our research approach and the scope of our data, which points at opportunities for future research. First, we cannot claim generalizability of the findings beyond the studied context. Therefore, additional research that studies different empirical contexts might complement our results. Second, stemming from our selection of mature communities, we did not regard path dependencies between the individual episodes. Future research might identify dynamic aspects of cross-community coordination. Third, we did not observe coordination activities directly, but derived them from verbalized contributions on discussion boards.

Hence, future research could employ ethnographic methods to draw a richer picture of the micro-foundations of cross-community coordination. Fourth, we obtained data only from the discussion boards on Github. Other tools to coordinate work are available, such as email, chat, and (virtual) whiteboards. As such, further examinations could tap into alternative data sources to identify coordination mechanisms that we did not observe.

In summary, much of the existing research on cross-community coordination assumes that boundary spanners and loose coupling are sufficient to resolve interdependencies between OSS projects. In this paper, we employed an explanative case study to elicit four mechanisms for coordinating work across interdependent OSS communities, which is the main accomplishment of this paper. In addition, we argued that cross-community coordination mechanisms are means of generative change in digital ecosystems, and thus help us better explain processes of digital innovation in heterogeneous systems.

References

1. Raymond, E.: The Cathedral and the Bazaar. Know Techn Pol 12, 23–49 (1999)
2. Crowston, K., Wei, K., Howison, J., Wiggins, A.: Free/Libre Open-Source Software Development: What We Know and What We Do not Know. ACM Computing Surveys 44, Article 7 (2012)
3. Faraj, S., von Krogh, G., Monteiro, E., Lakhani, K.R.: Special Section Introduction—Online Community as Space for Knowledge Flows. Information Systems Research 27, 668–684 (2016)
4. Selander, L., Henfridsson, O., Svahn, F.: Capability Search and Redeem Across Digital Ecosystems. J Inf Technol 28, 183–197 (2013)
5. RubyGems: Gems, https://rubygems.org/gems (Accessed: August 12, 2017)
6. Zittrain, J.L.: The Future of the Internet and How to Stop It. Yale University Press, New Haven, CT (2008)
7. Yoo, Y.: The Tables Have Turned: How Can the Information Systems Field Contribute to Technology and Innovation Management Research? JAIS 14, 227–236 (2013)
8. Howison, J., Crowston, K.: Collaboration Through Open Superposition: A Theory of the Open Source Way. MIS Quarterly 38, 29–50 (2014)
9. Faraj, S., Xiao, Y.: Coordination in Fast-Response Organizations. Management Science 52, 1155–1169 (2006)
10. Grewal, R., Lilien, G.L., Mallapragada, G.: Location, Location, Location: How Network Embeddedness Affects Project Success in Open Source Systems. Management Science 52, 1043–1056 (2006)
11. Crowston, K., Li, Q., Wei, K., Eseryel, U.Y., Howison, J.: Self-organization of Teams for Free/Libre Open Source Software Development. Information and Software Technology 49, 564–575 (2007)
12. Annabi, H., Crowston, K., Heckman, R.: Depicting What Really Matters: Using Episodes to Study Latent Phenomenon. ICIS 2008 Proceedings (2008)
13. Okhuysen, G.A., Bechky, B.A.: Coordination in Organizations: An Integrative Perspective. The Academy of Management Annals 3, 463–502 (2009)
14. Haefliger, S., von Krogh, G., Spaeth, S.: Code Reuse in Open Source Software. Management Science 54, 180–193 (2008)

15. Lyytinen, K., Yoo, Y., Boland, R.J.: Digital Product Innovation Within Four Classes of Innovation Networks. Information Systems Journal 25, 47–75 (2016)
16. Daniel, S., Stewart, K.: Open Source Project Success: Resource Access, Flow, and Integration. The Journal of Strategic Information Systems 25, 159–176 (2016)
17. Singh, P.V.: The Small-World Effect: The Influence of Macro-Level Properties of Developer Collaboration Networks on Open-Source Project Success. ACM Transactions on Software Engineering and Methodology (TOSEM) 20, Article 6 (2010)
18. Vasilescu, B., Blincoe, Kelly, Xuan, Qi, Casalnuovo, C., Damian, D., Devanbu, P., Filkov, V.: The Sky is Not the Limit: Multitasking on GitHub Projects. ICSE 2016 Proceedings, 994–1005 (2016)
19. Weiss, M., Moroiu, G., Zhao, P.: Evolution of Open Source Communities. OSS 2006 Proceedings, 21–32 (2006)
20. Baldwin, C.Y., Clark, K.B.: Design Rules: The Power of Modularity. MIT Press, Cambridge, Mass. (2000)
21. Ghazawneh, A., Henfridsson, O.: Balancing Platform Control and External Contribution in Third-party Development: The Boundary Resources Model. Information Systems Journal 23, 173–192 (2013)
22. Bogart, C., Kästner, C., Herbsleb, J., Thung, F.: How to Break an API: Cost Negotiation and Community Values in Three Software Ecosystems. FSE 2016 Proceedings, 109–120 (2016)
23. Decan, A., Mens, T., Claes, M., Grosjean, P.: When GitHub Meets CRAN: An Analysis of Inter-Repository Package Dependency Problems. SANER 2016 Proceedings, 493–504 (2016)
24. Gonzalez-Barahona, J.M., Robles, G., Michlmayr, M., Amor, J.J., German, D.M.: Macro-level Software Evolution: A Case Study of a Large Software Compilation. Empir Software Eng 14, 262–285 (2009)
25. Hansson, D.H.: Ruby on Rails, http://rubyonrails.org/ (Accessed: August 10, 2017)
26. Eck, A., Uebernickel, F.: Reconstructing Open Source Software Ecosystems: Finding Structure in Digital Traces. ICIS 2016 Proceedings (2016)
27. Biazzini, M., Baudry, B.: May the Fork Be with You: Novel Metrics to Analyze Collaboration on GitHub. WETSoM 2014 Proceedings, 37–43 (2014)
28. Borges, H., Valente, M.T., Hora, A., Coelho, J.: On the Popularity of GitHub Applications: A Preliminary Note (2015)
29. Lindberg, A., Berente, N., Gaskin, J., Lyytinen, K.: Coordinating Interdependencies in Online Communities: A Study of an Open Source Software Project. Information Systems Research 27, 751–772 (2016)
30. Corbin, J.M., Strauss, A.L.: Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Los Angeles, Calif (2008)
31. Chapin, N., Hale, J.E., Khan, K.M., Ramil, J.F., Tan, W.-G.: Types of Software Evolution and Software Maintenance. Journal of Software Maintenance and Evolution: Research and Practice 13, 3–30 (2001)
32. Ma, W., Chen, L., Zhang, X., Zhou, Y., Xu, B.: How Do Developers Fix Cross-project Correlated Bugs? A Case Study on the GitHub Scientific Python Ecosystem. ICSE 2017 Proceedings, 381–392 (2017)
33. Garud, R., Jain, S., Tuertscher, P.: Incomplete by Design and Designing for Incompleteness. Organization Studies 29, 351–371 (2008)