Università Ca’Foscari Venezia

Department of Management

Working Paper Series

Francesco Rullani, Markus C. Becker and Francesco Zirpoli

Coordination of joint search in distributed innovation processes. Lessons from the effects of initial code release in Open Source Software development

Working Paper n. 20/2013
October 2013

ISSN: 2239-2734

Coordination of joint search in distributed innovation processes:

Lessons from the effects of initial code release in Open Source Software development

Francesco Rullani
Department of Business and Management, LUISS Guido Carli
Viale Romania 32, 00198, Rome, Italy
Ph: +39 0685225935; Fax: +39 0685225949; frullani@luiss.it

Markus C. Becker
Strategic Organization Design Unit, University of Southern Denmark,
Campusvej 55, DK-5230 Odense M, Denmark
Ph: +45 65503316; Fax +45 61155129; mab@sod.dias.sdu.dk

Francesco Zirpoli
Department of Management, Ca' Foscari University
Cannaregio 873, 30121, Venice, Italy
Ph: +39 0412348786; Fax: +39 0412348701; fzirpoli@unive.it

This version: 14 October 2013

Abstract

This paper casts light on the role of initial code release for providing coordination of joint search processes, i.e., search processes that involve several agents who search together. We develop hypotheses about the role of initial code release for providing coordination, and for whether development projects remain active. We test these hypotheses on a dataset of 5703 open source software projects registered on SourceForge during a two-year period. We find that initial code release is indeed associated with improved coordination, and a higher chance that software development projects will actually release further code subsequently. We contribute to theory on coordination in joint search, common in distributed innovation settings.

Acknowledgements: We thank Carliss Baldwin, Siobhan O'Mahony, Kannan Srikanth, Keld Laursen, Toke Reichstein, Marcel Bogers, Volker Mahnke, Giulia Trombini, Lorenzo Zirulia, Matteo Ploner, and all the participants to the seminars at University of Lugano, University of Vienna, Ca' Foscari University of Venice, University of Milan, ETH Zurich, University of Jena and Max Plank Institute, to the conferences "Organizing for Internal and External Knowledge Creation and Innovation: The Role of Resources" (Copenhagen, October 2009), DRUID 2012, Academy of Management 2010 and 2013, and at the Elisir network (Electronic List in Innovation Studies In Rome). Francesco Rullani gratefully acknowledges the support by the program PRIN 2010-2011 CUP: B81J12002690008.

1. Introduction

The development of complex products, such as automobiles, aircraft or software, typically takes place in networks that draw on multiple actors such as suppliers, customers, or universities (von Hippel, 1988; Womack et al., 1990; Powell et al., 1996, Chesbrough, 2003). Considering innovation as a search process (Nelson & Winter, 1982; Fleming, 2001), developing innovative complex products therefore represents a case of joint search, i.e., multiple actors searching together (Bhardwaj, et al., 2006, Knudsen & Levinthal, 2007). From an organizational perspective, the main challenge in a joint search process is how to coordinate the involvement of many actors, i.e., to align their actions to achieve the intended objective (Heath & Staudenmayer, 2000). This challenge stems from interdependences between different actors involved in the search process (Thompson, 1967, Levinthal, 1997, Sosa et al., 2004). Empirical research shows that coordinating the different actors to achieve the intended objective can be a massive challenge in developing highly complex products such as cars (e.g., Zirpoli & Becker, 2011), aircraft (Brusoni et al., 2001) or software, especially open source software (e.g., Markus, 2007; David & Rullani, 2008).

While joint search processes are common, we know little about how to coordinate them. A stream of literature has shed much light on the search process and its difficulties, mostly by using simulation models (Levinthal, 1997; Fleming & Sorenson, 2004; Siggelkow & Rivkin, 2006). Much of the research on search, however, assumes that the searcher is an individual or a single firm (Knudsen & Levinthal (2007). As Knudsen & Levinthal (2007: 39) argue, most formal models of search tend to be 'remarkably nonorganizational'. Work on search processes has, thus, not shed much light on how to coordinate multiple individuals who are searching (Bhardwaj et al., 2006). Yet, coordinating the different searchers – i.e., aligning their actions (Gulati et al., 2005; Okhuysen & Bechky, 2009) – is essential for countering the risk that the different individuals disperse their search efforts by developing different solutions that cannot be integrated and thus, do not converge to the same solution (Heath & Staudenmayer, 2000). Dispersion of individual search efforts can therefore carry the risk of not reaching a complete, functioning solution to the problem (e.g., a product design). The gap in knowledge about how coordination is provided in joint search processes does, therefore, matter. The objective of the present article is to improve understanding of how coordination in joint search is achieved.

To contribute to understanding how joint search is coordinated, we selected the setting of open source software (OSS) development. From an organizational perspective, the development of open source software represents an extreme case with regard to coordination of joint search. First, the coordination challenge is particularly demanding: open source software development requires coordinating many independent developers who work in a self-organized fashion (von Hippel & von Krogh, 2003). Second, the stakes are very high: there is a high risk that developers split into different and potentially incompatible software (Dalle & Juillen, 2003; Narduzzo & Rossi, 2005). Third, the available coordination mechanisms are less powerful than in other settings: the set of available coordination mechanisms is restricted due to the lack of direct supervision and authority in a self-organized community without employment contracts, which by nature implies a bottom-up approach to coordination (von Krogh et al., 2012; David & Rullani, 2008; Lanzara & Morner, 2005). Furthermore, implementing many of the coordination mechanisms that are considered especially important in collaboration, such as common knowledge (Giuri et al., 2010) and coherence in aspects of the work setting that help make sense of the task and of communications from others (Crowston et al., 2007), is more demanding. The reason is that in OSS work is not assigned, there is no explicit system-level design, and no project schedule or list of deliverables (Markus, 2007; Mockus et al., 2002: Crowston et al., 2012).

Despite the great importance of coordination for open source software development, there are only few studies on coordination in OSS (Chua & Yeow, 2010: 840). How participants in open source software projects coordinate on what software processes, work practices, and organizational contexts they build on is still relatively under-researched (Chua & Yeow, 2010: 840) and not perfectly understood yet (von Krogh et al., 2012; cf. Crowston, 2007). The possibility of shedding light on the coordination of joint search by identifying how coordination of open source software is achieved has, thus, not been used to the fullest possible extent so far. The purpose of this paper is to tap into the potential of research on the coordination of open source software development to further our understanding of the coordination of joint search. Research on the coordination of open source software development appears promising for understanding the coordination of joint search as it offers the occasion for observing the phenomenon in a particularly clear instantiation (Eisenhardt & Graebner, 2007; Siggelkow, 2007) and thus, to learn about how coordination of joint search is provided in particularly difficult conditions (von Hippel & von Krogh, 2003).

Research on distributed innovation in general and on open source software in particular identifies two main mechanisms by which coordination between individuals is usually achieved: by modular product architecture (see Baldwin & Clark, 2000 for the general case and MacCormack et al., 2006; Baldwin & Clark, 2006 for open source software), and by an integrating role such as a system integrator in the case of distributed innovation (Hobday, Davies & Prencipe, 2005) and leadership in open source software (e.g., Lerner & Tirole, 2002; O'Mahony & Ferraro, 2007; Giuri et al., 2008).

In the open source software setting, the extent to which modular product architecture and leadership can provide coordination is limited, however. Limits arise from the self-organized nature of the community. Self-organized social bodies lack a principal with the authority to direct subordinates. This leads to difficulties in developing a full-fledged architecture in the beginning and the need to instead rely on architectures that are emergent (Narduzzo & Rossi, 2005). The extent to which the need for coordination can be reduced by a modular product architecture is, therefore, restricted. The possibility of providing coordination by leadership is also weakened: in an open source software setting, leadership cannot be enforced by means of an employment contract, and thus, cannot provide coordination by direct supervision (Mintzberg, 1979). Rather, any authoritative structure needs to be sustained by a process of continuous legitimation (O'Mahony & Ferraro, 2007). Such a process is also emergent (Mateos-Garcia & Steinmueller, 2008; O'Mahony & Ferraro, 2007) and thus demanding. Its outcome can be instable and, in some circumstances, unreliable.

There is another mechanism, however, that seems to play a role in coordinating open source software development but has been explored little so far: in open source software development, software code itself has also been considered a coordination tool (Lanzara & Morner, 2005; Rullani & Haefliger, 2013, Dalle & David, 2005). In particular, providing some initial "running code" at the beginning of a project (Raymond, 1998; Lerner & Tirole, 2002; Haefliger, von Krogh & Spaeth, 2008; Sojer & Henkel, 2010) represents a strong regularity in open source software development and has an important impact on the success of development projects (Raymond, 1998; Lerner & Tirole, 2002). Given (a) the limits to modularity and leadership mentioned above, as well as the important role that the release of initial code has for the success of open source software projects, and (b) the recognition that artifacts have a role in providing coordination of open source software development (Chua & Yeow, 2010) and that software code is an artifact (de Souza et al., 2005; Barrett & Oborn, 2010), it seems promising to direct attention to the role that initial code release plays for coordinating joint search in open source software projects.

In this paper we theorize on the role of initial code in providing coordination of joint search processes. The research question we ask is 'Does initial code release help coordination in joint search, and does coordination help projects to remain active?' We develop hypotheses about the role of initial code release for providing coordination, and for whether development projects remain active. We test these hypotheses on a dataset of 5703 open source software projects registered on SourceForge during a two-year period. We find that initial code release is indeed associated with improved coordination, and a higher chance that software development projects will actually release further code subsequently.

This result extends theory on joint search by identifying how artifacts (in our case, software code) made available at the beginning of the joint search process can enable tacit coordination, i.e, 'coordination of activities largely by relying on common ground – knowledge that is shared and known to be shared —formed by means other than ongoing communication' (Srikanth & Puranam, 2011: 850). Through the observability they afford, they provide opportunities for example for direct information sharing, scaffolding, acknowledging and aligning work, and creating a common perspective (Okhuysen & Bechky, 2009; Edmondson et al., 2001) by which they can contribute to building up common ground with regard to the 'preferred direction' (Winter et al., 2007) of the search that facilitates aligning actions of the various searchers. We thus contribute to theory of joint search and distributed innovation by identifying a type of tacit coordination mechanism that facilitates alignment of actions by establishing precedence and, in so doing, fosters coordination in joint search. The findings also suggest that such a mechanism is able to substitute hierarchy and organization structure as mechanisms that provide coordination of multiple actors in search processes. Finally, we specify that this mechanism is particularly valuable in situations of joint search where relying on feedback alone does not provide guidance of search because the different individuals involved might get different feedback and/or interpret it differently, leading to misaligned actions.

2. Background and prior research

2.1. Distributed innovation

The involvement of external sources of innovation (e.g. von Hippel, 1988, Clark & Fujimoto, 1991; Nishiguchi, 1994) has emerged as an important part of the innovation management literature (Chesbrough, 2003; Laursen & Salter, 2006; Dahlander & Gann, 2010). The major motivation for involving external sources of innovation in product development is that doing so can increase the performance of new product development projects (Womack et al., 1990; Clark & Fujimoto, 1991; Wheelwright & Clark, 1992).

Involving external sources of innovation is not trivial, however (Laursen & Salter, 2006; Zirpoli & Becker, 2011). Prior literature has identified the main causes of problems in achieving high project performance in distributed innovation. The most important cause, especially when products are complex, is technical interdependencies (Sosa et al., 2004). It is not trivial to coordinate a system when the innovation process is decomposed into smaller sub-tasks and those smaller development tasks are allocated to a number of different actors. Without effective coordination, technical interdependencies may result in inconsistencies one component or subsystem and others (Zirpoli & Becker, 2011). As a consequence, lack of effective coordination leads to low project performance.

Prior research has identified four issues firms have to address to achieve high project performance: (1) how to divide the development task, (2) how to allocate the sub-tasks, (3) how to coordinate the actors that develop components and subsystems, and (4) once the components and subsystems have been developed, how to integrate the components and subsystems into a whole that has high product performance (Baldwin & Clark, 2000; Takeishi, 2001, 2002; Brusoni et al., 2001). Given the gap concerning the coordination of joint search, in this article we focus on the third issue. In the remainder of this section, we first consider prior research on coordination of joint search more generally, and then prior research on coordination of distributed innovation more specifically.

2.2. Coordination in joint search

As pointed out by Bhardwaj et al. (2006) and Knudsen & Levinthal (2007), extant knowledge on joint search is limited. In the literature concerned with search processes, many models of search assume an individual searcher and have therefore eclipsed the challenge of coordinating multiple searchers. Among those that have considered the organizational aspects of joint search (March 1991, Lin & Carley 1997, Seshadri & Shapira 2003, Rivkin & Siggelkow 2003, Siggelkow & Rivkin 2005, cf. Knudsen & Levinthal, 2007: 39), we can identify two ways of providing coordination of joint search.

Rivkin and Siggelkow (2003) pay attention to the organizational structure within which search takes place. They consider vertical hierarchy, incentives (firm-wide vs. department-wide), decision decomposition, the underlying pattern of interactions among decisions, and limits on the ability of managers to process information. Employing a simulation model, the authors identify the impact of sets of these design elements on the characteristics of the search process undertaken by several actors in that organization, for instance on the breadth of search as a function of whether the CEO rubberstamps or actively canvasses proposals by subordinates (Rivkin & Siggelkow, 2003; cf. Siggelkow & Rivkin, 2005). The authors also identify how different sets of decisions on vertical hierarchy, incentives, decision decomposition etc. affect the appropriate balance between search (identifying good sets of choices) and the ability to stabilize around those sets once they are discovered.

Bhardwaj et al. (2006) explicitly focus on corporate search, 'a continual process of multiple, simultaneous efforts that engages many individuals' (Bhardwaj et al., 2006: 249). As summarized by Bhardwaj et al. (2006: 249), prior literature has identified influences on corporate search that include past entrepreneurial activities, past choices, strategic and structural contexts, sociopolitical and cognitive processes, existing knowledge and capabilities, and temporal economies of scope. As the authors argue, this prior knowledge is, however, not sufficient to understand why firms have taken a specific search path. In an archival study of corporate search at DuPont, Bhardwaj et al. (2006) identified the mechanism that provided coordination of the multiple actors involved in the sample of corporate search processes they analyzed. The mechanism that aligned the efforts of the individual searchers was 'moving, anchored search' (Bhardwaj et al., 2006). As the authors explain, 'despite drawing a boundary for where search will be conducted, a domain still contains innumerable possibilities yet to be discovered or created. Decision makers thus choose a search anchor within the domain to guide further search for growth possibilities. … Taking the domain as given, subsequent search is tethered to the chosen anchor and involves creating and discovering growth possibilities using the anchor as guide' (Bhardwaj et al., 2006: 251).

2.3. The coordination of distributed innovation processes

The more specific literature on innovation management – in particular on developing highly complex systems – identifies two ways of tackling the organizational challenges involved in distributed innovation. First, a focal firm acts as system integrator (Hobday, Davies & Prencipe, 2005). This approach consists in orchestrating the other partners involved in the product development process through a top-down hierarchical approach. The key challenge involved is to take into account all interdependences between actors in aligning their actions. The second possibility is to rely on modular product architecture, i.e., an architecture where interdependences are bundled within modules while modules are independent of each other and have standardized interfaces (Baldwin & Clark, 2000). Such a modular product architecture supposedly allows using a modular organization structure (both within the firm and in the value chain) (Sanchez & Mahoney, 1996). It thereby diminishes the need to actively provide coordination, i.e., align the actions of the actors who develop the individual modules: external sources of innovation can accomplish their development tasks independently and do not require explicit coordination, as the standardized interfaces and independence between modules assure that modules will fit together even without coordination when integrated into the overall product (Baldwin & Clark, 2000). In both approaches, the coordination problem is moved to the level of the architecture and, ultimately, to the architect who designs it.

The same two solutions have also been identified in the more specific context of open source software development. In her comprehensive review of governance of open source software projects, Markus (2007) identifies coordination as one of the central challenges for such development projects: "In the operational coordination literature, OSS governance is understood as a solution to [… the problem of] loss of operational control, and the solution is techniques for managing the process of OSS development work" (p. 156). Markus (2007: 159) also identifies four 'governance dimensions' that are employed to solve coordination problems in OSS: community rules, software development process rules, conflict rules and rules about rules, and information and tools rules. Other research has emphasized the role of software version control (SVC) in providing coordination (Chua & Yeow, 2010), as well as explicit development processes (such as the rules Markus (2007) identified), individual or group code ownership, and required inspections (Mockus et al., 2002), standardization (German, 2003; Chua & Yeow, 2010; Crowston et al., 2012), or restricted access to keep the group of core developers small (Chua & Yeow, 2010).

A specific feature of the OSS innovation model is that individuals self-select the tasks they perform (Langlois & Garzarelli, 2008; Crowston et al., 2012), leading to the emergence of self-organized systems (Kogut & Metiu, 2001; Lee & Cole, 2003; David & Rullani, 2008) where authoritative structures (Mateos-Garcia & Steinmueller, 2008) and leaders (Lerner & Tirole, 2002) are continually created, renewed or destroyed. Social processes leading to this emergence, such as criticism of the status quo (Lee & Cole, 2003) or creation of specific patterns of social ties facilitating the making of individuals into leaders (Dahlander & O'Mahony, 2011), have drawn much research attention and, in fact, became the center of many recent studies in the field. By the same token, attention was devoted to the limits of such processes, investigating how conflicts are resolved (Elliott & Scacchi, 2003) and how authoritative structures are challenged and changed (O'Mahony & Ferraro, 2007). The OSS literatures has thus highlighted two coordination mechanisms that are especially important in the OSS setting (Markus, 2007; O'Mahony & Ferraro, 2007; Dahlander & O'Mahony, 2011): coordination by project leaders and by modular product architectures. The next two sections report prior research on these two issues.

2.3.1. Leadership

Involving voluntary participation, OSS collaboration implies a type of control which can be centralized into a leader (as happens for Linux, Raymond, 1998; Lerner & Tirole, 2002) but that also needs to be continually reproduced and legitimated by the participants in the development process. Raymond (1998) argues that legitimation as leader comes naturally as a consequence of project foundation: the founder considers it recognized by other participants that she has the right to take the final decision with respect to the development of the project. However, legitimation is not a static concept, and has to be recreated and renewed each time (O'Mahony & Ferraro, 2007). Muller (2006) shows through a simulation model how leadership emerges as the result of a dynamic legitimation process among peers clustering around 'opinion leaders'. O'Mahony & Ferraro (2007) connect these results to the evolution of the OSS project they study (Debian) and identify the transformation the authoritative structure and the governance mechanisms go through when the project moves from one phase of development to a more complex one. Their description of the dynamic transformation of governance highlights the passages from an autocratic leadership towards a formalized authority structure that acquires legitimation through the construction of democratic regulatory processes. This echoes Lee and Cole's (2003) identification of the community debate as the key mechanism through which the OSS community evolves. Following the same dynamic perspective, Mateos-Garcia and Steinmueller (2008) argue that "as the capabilities the integrator has for keeping up with a project's development pace start diminishing, a structure with layers of trusted individuals emerge as a way of helping her or him cope with the increased complexity and size of the project. These layers will be composed of proficient individuals with experience in the project. The vision they have of the project will also concur with that of the leader in some essential points" (p. 22). Thus, a pyramidal structure gradually emerges, but again based on legitimation mechanisms involving the leader's vision as well as the technical capabilities of the developers coming to populate the intermediate layers.

Elliott and Scacchi (2003) apply a perspective that allows a more fine-grained definition of the legitimation process. Inspired by the studies that highlight the community-related aspects of OSS projects (e.g. Cohendet et al., 2001) the authors describe how developers resolve conflicts over the legitimacy of undertaking certain disputed actions (such as using software tools that are not open source to produce material for an open source project). The reference to common values and shared norms are the main rhetoric instruments used to solve the conflicts arising when some one's actions are disputed. What is interesting to highlight here is that the authors recognize not only the importance of the norms themselves, but also the fact that they are embodied in the online discussion stored in easily accessible web repositories. "This fast access to archived information perpetuates the cultural beliefs that have been articulated and assists in resolution of conflicts" (Elliott & Scacchi, 2003; p. 9). In this sense, stored online discussions have a crucial role in coordination. They are the instrument for propagating to the periphery the emergent social practice (and the organizational structure and rules defined therein) developed by core members of the project (Rullani & Haefliger, 2013).

2.3.2. Modular structure of code

A recent stream of literature tries to merge the social side dominating the previous points of view with the "materiality" of the production process itself (Orlikowski, 1992, 2000; D'Adderio, 2003; Cacciatori, 2008). Software developers, in fact, do not interact only exchanging opinions, but also act on artifacts – the lines of source code composing the software – that they exchange and jointly develop (Lanzara & Morner, 2005).

The structure of the code, and in particular its modularity, has been identified as a crucial issue, because artifacts mediated the interaction between individuals and thereby, can contribute to coordination. David and Ghosh (2008) have shown that the structure of the technical interdependencies between the different modules composing Linux have a certain degree of correspondence to the pattern of social ties the authors of those modules have built through past collaboration. A similar perspective emerges from the analysis undertaken by Narduzzo and Rossi (2005) in their effort to define modularity in the OSS context. In OSS the architecture of the software is likely to be constantly changed over time, and so is the degree of modularization it embodies. Narduzzo and Rossi study how a community of developers copes with the emergence of interdependencies, and notice that every time a new dependency connects two modules challenging the current architecture, the main principle of modularity – information hiding – is reversed: developers exchange module-specific information with the aim of discussing a common understanding of the new logic underpinning the product. Again, the artifact's structure and the social side are coupled in the process of product development. MacCormack et al. (2006) focus on a similar process but with an opposite perspective: they also study the transformation of a weakly-modular OSS product into a highly modularized software, but in the context of a transformation implemented top-down by a firm (Netscape) opening the source code of its software as a strategy to attract external developers. They observe that the project was initially stagnating because its weakly-modular structure increases the cost of external developers' contribution, and that the subsequent increase in modularizing determined instead its success. The strict link between participation and modularity has been further developed by Baldwin & Clark (2006). Modularity is seen not only as the determinant of a lower cost of contributing. Rather, a high level of modularity also assures a higher option value for external developers in terms of the possible future configurations of the product, and thus increases their willingness to participate. Modularity is thus considered to have the effect of decreasing free-riding. In their review of coordination mechanisms used in OSS development, Crowston et al. (2012: 18) argue that 'modularity is the most explicit mechanism used in FLOSS [free and libre open source software] development'.

While these two mechanisms, leadership and modular structure of code, have received much attention in prior research, they are met with limits that are intrinsic to the open source software setting. In both approaches, the coordination problem is moved to the level of the architecture and the architect who designs it. In self-organized social bodies it is however difficult for architects to master the process in the same way they could do in a firm. The main problem is that it is very difficult to develop a full-fledged architecture at the beginning of a self-organized process of development: self-organization by nature implies a bottom up approach to coordination (von Krogh et al, 2012; David & Rullani, 2008; Lanzara & Morner, 2005), involving the risk that developers split into different and potentially incompatible software (Dalle & Juillen, 2003; Narduzzo & Rossi, 2005). The potential of providing coordination by leadership is limited by the same factors. In a self-organized community, leadership needs to be constantly legitimized (O'Mahony & Ferraro, 2007), in an emergent process (Mateos-Garcia & Steinmueller, 2008, O'Mahony & Ferraro, 2007) that is demanding and can be unreliable. There is, thus, reason to believe that these two mechanisms do not bear the whole burden of providing coordination of joint search in open source software development. We therefore focus our attention on a third mechanism that has been reported as important in open source software development, but whose contribution to providing coordination of joint search has not yet been investigated.

3. Hypothesis development: The role of initial code release in open source software development

In his early work on open source software, Raymond (1998) noted that the release of code at the beginning of the projects had an important role for open source software development. He argued that the nascent developer community 'needs to have something runnable and testable to play with' (Raymond, 1998). Other researchers, too, pointed out that releasing initial code at the beginning of projects is very common (Raymond, 1998; Lerner & Tirole, 2002; Haefliger, von Krogh & Spaeth, 2008).

The provision of initial code is considered to have an important impact on the success of development projects, through attracting programmers to projects, legitimating projects, and code reuse.

(1) Initial provision of code has been considered a (mostly unintentional) strategy to attract other developers. Intrinsic motivation (Lakhani & Wolf, 2005) such as fun in coding and in solving challenges is one of the main incentives for developers' participation (Ghosh et al., 2002; David et al., 2003) and for deciding to join a particular project (David & Shapiro, 2008). The fact that programmers can directly test the code, see what works and what does not, find interesting problems and "scratch a personal itch" of theirs (Raymond, 1998) has been indicated as a crucial mechanism in attracting developers to projects.

(2) In this connection, Lerner and Tirole (2002) noticed that initial "runnable and testable" releases of the code attracted new developers that see interesting challenges. The authors also notice that initial code has a legitimation function: it shows to other potential developers that the code has some merit and that developing it further will not be a waste of time. This concept is further developed by Haefliger et al. (2008), who discuss the fact that a project can attract more developers if its initial code release effectively carries the "credible promise" of a stream of interesting challenges for future developers. The capability of code to attract interest and new developers has also been empirically documented by von Krogh et al. (2003), who notice that in the case of Freenet, no code was provided at the beginning but rather, only 15 weeks later. It was only after this release that the project witnessed a steep increase in attention, number of contributors and of discussions.

(3) Finally, code released at the beginning of a software development project is also considered to have an impact on the success of projects because of code reuse (Haefliger et al., 2008: Sojer & Henkel, 2010). Initial code is often existing code adapted to new purposes (Haefliger et al., 2008). Re-using code saves time and enables programmers to focus on the most interesting and still unresolved issues.

Prior research in OSS has, thus, shown that release of initial code has an important impact on the success of open source software projects, and has identified the three mechanisms, described above, by which initial code release has such an impact. None of the three mechanisms directly explain how coordination of the joint search process takes place in the software development projects, however. Nevertheless, as mentioned above, leadership and modular product architecture have limits with regard to providing coordination and initial code release seems to have some influence on coordination of the joint search of open source software programmers. It is not clear how, though.

To shed light on this question, it is helpful to clearly identify the challenge the developers face in joint search. Consider a software program composed by *n* components. Combining these components in a Design Structure Matrix (DSM), it is easy to see that the potential number of elements composing the off-diagonal upper-right triangle of the DSM is $N = \frac{n(n-1)}{2}$, and thus that the number of all possible relations between them is $\sum_{k=1}^{N} \frac{N!}{k!(N-k)!}$. Thanks to the Binomial Theorem we know that $\sum_{k=0}^{N} \frac{N!}{k!(N-k)!} = 2^N$, which in terms of *n*, leads to $2^{\frac{n(n-1)}{2}}$. This is clearly an explosive function of *n*, leading to an enormous amount of possible future architectures even for quite small *n*. This property remains even for DSM that are very sparse and have few off-diagonal elements. There is, therefore, an explosive number of possible software architectures for fulfilling the functional specifications. Facing too many possibilities is one of the factors that drive the risk of misaligned actions. The other factor is that even if there are few alternatives, actors have problems aligning their actions and converging to one alternative. How could initial code release possibly lead to coordination of joint search for software solutions?

Initial code release is characterized – indeed, defined – by two features: it is released at the beginning of a development project, and it is an artifact (de Souza et al., 2005; Barrett & Oborn, 2010). (Both aspects are developed below.) Moreover, initial code release represents one of the two possible types of information one can release when an OSS project is started. The first type of information concerns the function specification describing what the product is supposed to do. In terms of search, such a description provides the end point to be reached, but in abstract form – what the product is supposed to do, but not how it is to look in order to provide such functions. This approach does not contain any explicit coordination mechanism: only if searchers have the same idea about what product features best generate these functions will they act in a coordinated fashion. The release of an initial code is the second possible type of information that can be provided when a project is started (of course, both can be provided). When the code is released it provides a common starting point to all (potential) searchers. Thereby, it addresses 'the problem of identifying the starting points of the search process' or the 'seeding' of search (Levinthal & Warglien, 1999: 349). Note that while the functional specifications are also released at the beginning of a development project, they do not provide the starting point or the 'seed' of search – they describe the end point of the search, even if it is described at the beginning of the project. In joint search where multiple searchers are involved, providing such a 'seeding' of the joint search process is particularly important because the search process does not just carry the risk of not identifying the best possible solution; as explained above, there is also the possibility that the different searchers will not converge to one solution.

How does initial code release facilitate coordination of joint search? The above suggests two possible levers: (a) diminishing the number of alternatives and thereby increasing the probability for aligned actions, and (b) making it easier for two or more actors to converge on the same alternative.

(a) Initial code release can diminish the number of possible alternatives if the initial release of the code contains *m* components and relates them in a specific manner. Even if *m* is much larger than the number of components of the software program *n* (*m*>>*n*), simple inspection of the software released highlights a small fraction of the *n* components and specific relations that make some of the components more interdependent than others. This information provided by the initial code eases any further development of codes that embed the same decomposition logic, and creates higher costs for developments that instead realize the same tasks with a different architecture, de facto reconfiguring the relationship between the m components. Thus, future developments that change the decomposition logic embedded in the first code released are more costly than development on the basis of the decomposition logic the initial code implies. Vice versa, those improvements and enlargements that respect the way in which the interdependencies are managed according to the decomposition embedded in the initial software release, will be perceived as natural developments of the initial code.

(b) Code released at the beginning of the project can also make it easier for developers to converge to the same alternative and thus align their actions (Narduzzo & Rossi, 2005; Lanzara & Morner, 2005). As Lanzara and Morner (2005) explain, in OSS the code is "exposed", everyone can read it and evaluate it, run it and contribute to it. The initial code thus can be considered an artifact (de Souza et al., 2005; Barrett & Oborn, 2010) that helps multiple searchers to align their actions, similarly to the role of artifacts in providing coordination as pointed out by recent research in other settings (Cacciatori, 2012). In Cacciatori's (2012) typology of artifacts, software code falls in the category of 'speaking' artifacts, i.e., formal representations of knowledge in verbal, mathematical, or visual form. Within this category, software code is of the 'product representation' type (as are technical drawings or virtual prototypes). As Cacciatori (2012: 1562) writes, artifacts that are 'product representations' are often examined in their function as boundary object. From this perspective, the initial release of some running code to work on, provides an artifact at the beginning of the development process that is seen by individuals who (potentially) contribute to software development[1]. This artifact enables other developers to improve code, rather than create it from scratch (Sojer & Henkel, 2010; Haefliger et al., 2008). The difference to creating from scratch is substantial: creating from scratch also means imagining the future structure of the program, and having an idea of the whole system's functions, a situation hardly imaginable in a self-organized setting such as the one under scrutiny. In contrast, providing initial code at the beginning of the process provides a common starting point with regard to the content of the solution; thereby, it provides more than just the end point to be reached, described in

[1] Recent research in OSS has argued that artifacts have a role in providing coordination of open source software development. Chua & Yeow (2010) argue that different types of material artifacts shape and facilitate coordination. For instance, artifacts such as calendar systems (van den Hooff, 2004) or intranets (Kellogg et al., 2006) have been seen to contribute to coordination in open source software development (Chua & Yeow, 2010: 840-1). Considering the role of boundary objects in open source software development, Barrett & Oborn (2010) point out that boundary objects may facilitate collaboration in open source software production sometimes, but can also contribute to conflict.

terms of the functions the solution is supposed to provide. Seen from a coordination perspective, providing a common starting point with regard to the content of the solution can enable 'tacit coordination, i.e, coordination of activities largely by relying on common ground – knowledge that is shared and known to be shared – formed by means other than ongoing communication' (Srikanth & Puranam, 2011: 850). As Srikanth & Puranam (2011: 855) point out, 'technologies that enable observation of the work progress and context across sites enable building common ground through enhancing observability across locations of context, actions, and outcomes rather than through direct communication'. We argue that initial code release fulfils these criteria, and that the observability it offers provides opportunities for instance for direct information sharing, scaffolding, acknowledging and aligning work, and creating a common perspective as pointed out by Okhuysen & Bechky (2009: 475; Bechky, 2003a, 2003b; Edmondson et al., 2001). In the case of initial code release in open source software development, one of the mechanisms that observability of the code can provide is a 'preferred direction' (Winter, et al., 2007) with regard to the content of the solution; thus, initial code release can contribute to building up common ground with regard to the 'preferred direction' of the search that facilitates aligning actions of the various searchers.

What is common to both these potential roles of initial code release for providing coordination of joint search for open source software solutions is that initial code release establishes a precedent. Through mechanisms such as the ones identified above, such a precedent provides guidance in terms of the content of the solution that helps multiple searchers align their actions. On this basis we thus hypothesize that:

H1: The presence of initial code increases the probability of coordination between OSS project members.

Coordination is tightly linked to whether projects remain active. Prior research shows that only a very limited percentage of projects advance, and coordination plays a key role in improving the chance that the project remains active over time and makes actual progress, rather than fizzling out and falling into inactivity (Krishnamurthy, 2002). In the open source context, the self-organizing nature of the work creates considerable risk of misaligned actions (Narduzzo & Rossi, 2005). Misaligned actions can lead to the impression that the project 'goes nowhere', making participants lose interest so it dries up for lack of participation. Hence, one fundamental challenge in joint search is to avoid dispersion of the search efforts of the different individuals involved (Dalle & Juillen, 2003). By increasing the probability of coordination between interdependent actors, initial code release can have consequences on whether development projects remain active over time, i.e., that the project is actually producing some code later on. We therefore hypothesize:

H2: The higher the coordination between OSS project members due to initial code release, the higher the probability that the project produces a subsequent code release.

We do not claim any role of the initial code on the quality of the software developed, such as the degree of innovation. The effect of the initial code is here evaluated only with reference to its capability to produce subsequent code release(s), irrespectively of the quality of the software released.

In the next section, we turn to empirically testing our hypotheses on a sample of projects hosted on SourceForge during 2005 and 2006.

4. Empirical investigation

4.1. Evidence from SourceForge

4.1.1. Data

Our analysis of the OSS context draws on data relative to all projects populating the SourceForge platform from November 1999 to October 2008. Out of this population we selected all 5810 projects registered on the platform from January 15th, 2005, to April, 15th 2005. We tried to select the most

recent projects we could, provided that we had enough data to analyze subsequent developments of the projects. Moreover, as the data consist of monthly snapshots of the situation observed on the platform, and as the way data were stored and managed by SourceForge changed over time, we needed to carefully select the period of analysis in order to preserve data consistency between the different snapshots. This was particularly the case for project categories, as we will see in the next section. The final sample (5703 projects) has been obtained considering only the projects still registered as "Active" in September 2006 and with at least one member in July 2005 and March 2006.
We divided the period in three parts: a first period, called t_0, spans the first months of the projects' life up to July 8th, 2005. A second period (t_1) starts from this date and reaches March 20th, 2006. A third time window (t_2) is opened from that day to September 20th, 2006. We will use these time windows to make sure that in every equation we estimate, the independent variables are lagged by one period with respect to the dependent variable, thus reducing endogeneity. All the data come from within SourceForge.net.

4.1.2. Measures

Within this set of data we need to capture the connection between the provision of an initial piece of code in a project, coordination in the process of joint search, and the subsequent production of a newer version of the code.

We capture the provision of code at the beginning of a software development project by Initial code t_0. It is a dummy variable equal 1 if in t_0 the project team releases an initial version of the software to be developed. In order to set a minimum threshold for the definition of "initial code", we restricted our analysis to the provision of "running code", i.e. to the presence of files released by the project members as official releases, in agreement with Lerner and Tirole's (2002) claim that a minimum amount of running code would be necessary to attract developers and let them play with it (Raymond, 1998). The majority of projects on SourceForge (56%) did not provide any code in t_0.

We measure coordination as follows. Coordination refers to aligned action (Heath & Staudenmayer, 2000). The actions we are concerned with here are those related to the development of an OSS project. These actions can be described along a number of dimensions. In SourceForge.net developers themselves identify what dimensions are meaningful: they self-categorize projects in different categories, according to a list of categories offered by the platform. The categories are intended audience of the project (end users/desktop, developers, …), the programming languages it employs (C++, Java, …), the operating systems it runs on (Windows, Linux, …), the topic the project tackles (communications, security, games/entertainment, …), the environment it populates (X11 applications, web environment, …), and the language used by the developers to interact with one another. Each of these macro categories is then organized into lower level categories. For example, the category "topic" is divided into different subcategories among which "communications", which in turns contains - among others - the subcategory "chat", that leads to two possible final subcategories: "AOL Instant Messenger", and "ICQ" (see Table A1 in the appendix for another example[2]).

We measure coordination by the number of changes in the classification categories over a given period of time. The higher the number of changes that occurred in the classification categories over a given period of time, the less aligned the actions of the different developers involved and the lower their coordination. We thus build our measure considering the 8 months spanned by t_1 by comparing the list of categories each project posts at the beginning of that period to the same list at the end of the period, counting a change in the list every time a new category is added or an old category is dropped.

The fact that these categories are self-reported may be a problem. However, they are used by SourceForge users to browse the enormous amount of projects present on the platform. The rational behavior for current project members is thus to be as careful as possible in identifying the project's

[2] A last category is represented by the development status of the project. As the project progresses, the team updates this category to inform the public that the project has passed, for example, from its beta version to its mature version. We do not include this category in the analysis, as we apply a measure of projects' development status that is based on behavioral data rather than self-report data.

categories. Misleading categories will guide to the project's page SourceForge users that are not interested in it, and fail to attract those that are. Another potential problem comes from the fact that SourceForge changed over time the list of possible categories to choose from. This problem does not affect our data, as we carefully selected the time window to avoid those changes.

Our third measure concerns the probability that the project remains active (alive) over time. We measure this considering whether the project has produced and released any file over the 6 months composing period t_2. In OSS many projects are just dormant and produce no activity at all (Krishnamurthy, 2002). In this environment releasing some code at later stages in the project is a reasonable proxy discriminating between active (alive) and inactive (dead) projects.

Eventually, we need to include a number of controls to take into account possible confounding factors and alternative explanations. We report them in the Table 1 together with the main variables described above (see Table A2 and A3 in the appendix for the summary statistics and the correlations).

In sum, the data described above are used to investigate if the provision of initial code at the moment of a project's foundation (t_0) leads to a reduction of the number of category changes during the development of the project (from the beginning to the end of t_1), and if this effect in turn increases the probability of observing a new release of the code in t_2.

Table 1 – Variables used in the regression analysis

Variable	Description*
INITIAL_CODE_{t0}	dummy variable equal to 1 if there was at least one file posted by the project in the first months of its activity on SF.net, i.e. between its foundation and July 8th, 2005.
$\text{CATEGORIES\_CHANGES}_{t1\_t2}$	number of categories the project has acquired or lost between July 2005 and March 2006
$\text{CATEGORIES\_CHANGES}_{t1\_t2\_p}$	number of categories the project has acquired or lost between July 2005 and March 2006 as predicted by the first equation of the model
$\text{CODE\_RELEASED}_{t1}$	dummy equal 1 if the project has posted at least one file between July 8th, 2005 and March 20th, 2006
$\text{CODE\_RELEASED}_{t2}$	dummy equal 1 if the project has posted at least one file in the 6 months from April 2006 to September 2006 included
$\text{MEMBERS\_TENURE}_{t1}$	registered date in SF.net of those who were project members at July 2005 (average)
$\text{MEMBERS\_TENURE}_{t2}$	registered date in SF.net of those who were project members at March 2006 (average)
NUM_MEMBERS_{t2}	number of project members at March 2006 (average)
NUM_MEMBERS_{t1}	number of project members at July 2005 (average)
$\text{REGISTRATION\_DATE}_{t0}$	registration date of the project on the platform
USE_CVS_TOOL_{t0}	dummy equal 1 if the project uses the concurrent versioning system (CVS), a tool to manage distributed software development (May 2005)
USE_FORUMt_0	dummy equal 1 if the project uses forums (May 2005)
$\text{DUMMY\_CATEGORIES}_{t0}$	dummies for language, programming language, license, operating system, development status, topic, retrieved July 2005, i.e. end of period t_0

*Notice: time-related variables are measured in UNIX time, i.e. in number of seconds from midnight of January 1, 1970, a standard measure in computer science.

The estimation we run is aimed at showing the relationship between the three main variables defined above. In order to do this we take inspiration from Instrumental Variable technique (Greene, 2000). Such models are stage models (usually 2SLS, sometimes 3SLS) where the results of the first regression (here expressed in vector notation, as in Greene, 2000) of the kind:

$$\mathbf{X}=\mathbf{Z}\cdot\boldsymbol{\beta}+\boldsymbol{\varepsilon}$$

feed into a second regression:

$\mathbf{y}=\hat{X}\cdot\mathbf{b_{IV}}+\boldsymbol{\mu}$

where $\hat{X}$ is the predicted value of **X** resulting from the first stage estimation. This technique allows an unbiased estimation of the coefficient b_{IV} of the endogenous variables **X** in the second regression. This is the case because, conceptually, the first stage allows the "split" of the otherwise endogenous variable **X** into two components: one component, the predicted values $\hat{X}$, carries with it the part of **X** solely determined by the exogenous variables **Z** used as regressors in the first stage. It is thus itself exogenous. The other component is the residual of the first stage **ε**, which represents the part of **X** as determined by other variables excluded from the first stage – including those that would make **X** endogenous in the second stage.

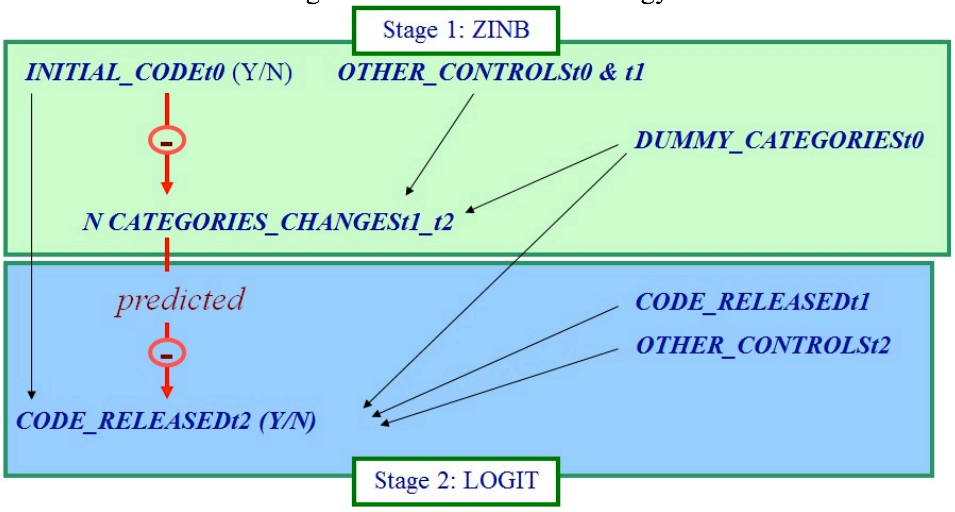
Given the object of the present paper, we take inspiration from this technique and run a first estimation, using INITIAL_CODE$_{t0}$ as independent variable and CATEGORIES_CHANGES$_{t1\_t2}$ as dependent variable. Then we predict the values of CATEGORIES_CHANGES$_{t1\_t2}$ (calling this new variable CATEGORIES_CHANGES$_{t1\_t2\_p}$) and use them as the main regressor in a second equation, where the dependent variable is code release in t_2 (CODE_RELEASED$_{t2}$). In this way we capture the effect of number of changes in categories on subsequent code release(s) when those changes are exclusively due to the presence of initial code. The purpose is to exclude any impact of CATEGORIES_CHANGES$_{t1\_t2}$ and CODE_RELEASED$_{t2}$ not due to INITIAL_CODE$_{t0}$ (and the controls of the first stage).

The first equation we need to estimate relates INITIAL_CODE$_{t0}$ and CATEGORIES_CHANGES$_{t1\_t2}$. Measuring the independent variables at time t_0 and the number of changes in the categorization as the delta between the beginning of t_1 and its end (and also controlling for the specific categories projects listed at the end of t_0) should diminish the possible endogeneity problem. As CATEGORIES_CHANGES$_{t1\_t2}$ is a count variable we could use a Poisson specification. However, the presence of overdispersion pushes us to prefer a Negative Binomial. By the same token, the presence of many zeros implies the use of an additional equation preceding the estimation stage and predicting the probability of being a project which structurally has zero changes in the categories it lists. This is also theoretically sound, as in OSS many projects are inactive even if they are still online (i.e., we expect them to *always* have no activity and thus no changes in their categories), while many others exhibit zero activity in a certain time window simply because in OSS the cycle of activity is very much cyclical, alternating very active phases (where changes of categories are possible) with phases with no activity at all (David & Rullani, 2008). The results of this first estimation (the "zero-inflation" process) are then used in the main estimation equation for the first stage. Thus, the final model for the first stage is a Zero-Inflated Negative Binomial, ZINB (in the table the results relative to a Zero-Inflated Poisson, or ZIP, are also reported as a robustness check). Notice that the sample of projects is also reduced to 5709 due to the fact that we excluded about 100 projects that were deleted from SourceForge before March 2006.

Once the estimation of this first stage has been carried out, the predicted values of CATEGORIES_CHANGES$_{t1\_t2}$ are used as independent variable in the second equation, which includes CODE_RELEASED$_{t2}$ as dependent variable. CODE_RELEASED$_{t2}$ being a dummy variable, it seems appropriate to use a Logistic Regression Model.

As mentioned, some controls have been included in the two equations to make sure no confounding factors are at work. In particular, CODE_RELEASED$_{t1}$ and INITIAL_CODE$_{t0}$ are also included in the second stage. CODE_RELEASED$_{t1}$ controls for the most recent performance of the project (in period t1) while introducing INITIAL_CODE$_{t0}$ also in the second equation controls for the direct effect of INITIAL_CODE$_{t0}$ on CODE_RELEASED$_{t2}$. This last passage reinforces the claim we made before: our coefficient captures the impact of CATEGORIES_CHANGES$_{t1\_t2}$ on CODE_RELEASED$_{t2}$ when CATEGORIES_CHANGES$_{t1\_t2}$ is determined exclusively by INITIAL_CODE$_{t0}$ (and other controls). Figure 1 provides an overview of the estimation procedure, while Tables 2a and 2b report the results of the estimates.

Figure 1. The estimation strategy



Notice that, as a robustness check, we also run a usual Instrumental Variable technique (namely, 3SLS). Our results are not only confirmed but magnified by a higher level of significance.

Table 2a – Results of the regression analysis – stage 1

Stage 1	CATEGORIES_CHANGES$_{t1\_t2}$	
	ZINB	ZIP
	b/se	b/se
INITIAL_CODE$_{t0}$	-0.312***	-0.251***
	[0.094]	[0.083]
REGISTRATION_DATE$_{t0}$	0.000	0.000
	[0.000]	[0.000]
NUM_MEMBERS$_{t1}$	0.042*	0.044***
	[0.024]	[0.016]
MEMBERS_TENURE$_{t1}$	0.000	0.000
	[0.000]	[0.000]
USE_FORUMt$_0$	-0.053	-0.047
	[0.170]	[0.146]
USE_CVS_TOOL$_{t0}$	-0.466**	-0.443***
	[0.186]	[0.167]
DUMMY_CATEGORIES$_{t1}$	YES	YES
Constant	-16.584	-19.879
	[22.675]	[20.744]
Inflation Model: logit		
INITIAL_CODE$_{t0}$	-0.236**	-0.180
	[0.115]	[0.113]
REGISTRATION_DATE$_{t0}$	0.000	0.000
	[0.000]	[0.000]
NUM_MEMBERS$_{t1}$	-0.077***	
	[0.026]	
MEMBERS_TENURE$_{t1}$	0.000	0.000
	[0.000]	[0.000]
USE_FORUM$_{t0}$	0.232	0.286
	[0.209]	[0.208]
USE_CVS_TOOL$_{t0}$	0.280	0.282
	[0.240]	[0.240]
DUMMY_CATEGORIES$_{t1}$	YES	YES
Constant	34.407	30.367
	[26.244]	[24.773]
lnalpha	-.786***	
	[0.167]	
alpha	0.456	
	[0.076]	
N (nonzero)	5703 (461)	5703 (461)
ll	-2533.426	-2644.177
chi2 (df)	145.95 (24)	
Pr > chi2	0.000	
Vuong	9.02	
Pr>z	0.000	

Table 2b – Results of the regression analysis – stage 2

Stage 2	CODE_RELEASED$_{t2}$	
	LOGIT for ZINB	LOGIT for ZIP
	b/se	b/se
CATEGORIES_CHANGES$_{t1\_t2\_P}$	-0.775**	-0.700*
	[0.325]	[0.413]
NUM_MEMBERS$_{t2}$	0.156***	0.123***
	[0.027]	[0.022]
MEMBERS_TENURE$_{t2}$	-0.000	-0.000
	[0.000]	[0.000]
CODE_RELEASED$_{t1}$	2.495***	2.502***
	[0.113]	[0.113]
INITIAL_CODE$_{t0}$	0.831***	0.841***
	[0.131]	[0.131]
DUMMIE_CATEGORIES$_{t0}$	YES	YES
Constant	-2.240*	-2.312*
	[1.312]	[1.312]
N	5703	5703
Ll	1288.1201	-1290.0610
Pseudo R-squared	0.291	0.290
LR chi2 (df)	1059.50 (23)	1055.62 (23)
Prob >	0.000	0.000

4.1.3 Results

Before moving to the regressions, consider a first descriptive result. The correlation between initial code provision and number of category changes is -0.0148. Even if non significant, its negative sign is in line with our argument, pointing towards the fact that projects with initial code exhibit a lower number of changes in the categories over time. This result is confirmed when comparing the frequencies of projects across different category changes distinguishing between projects with and without initial code provision. Both Kendall's tau-b and Mann-Whitney tests show that the two groups of projects differ in their distribution across category changes, and in particular that projects having no initial code exhibit a larger percentage of projects with many changes (p-values are 0.013; and = 0.0067, respectively).
In line with these results, in the first stage of the regression, the coefficient of INITIAL_CODE$_{t0}$ is negative and highly significant. The provision of an initial piece of code thus leads to a decrease in the number of category changes during the course of the project.

In the second stage, the coefficient of the predicated values of CATEGORIES_CHANGES$_{t1\_t2}$ is also highly significant, and also negative. This means that the smaller the number of changes in a project's categories (as predicted by the first equation), the more likely a project will effectively produce and release some files in the latest period considered. Moreover, as we have used the predicted values of CATEGORIES_CHANGES$_{t1\_t2}$ in the second stage, and as we have controlled for INITIAL_CODE$_{t0}$, for CODE_RELEASED$_{t1}$, we can also state that our regression captures exclusively how CATEGORIES_CHANGES$_{t1\_t2}$ - as determined by INITIAL_CODE$_{t0}$ - affects CODE_RELEASED$_{t2}$. To be perfectly sure of this we also run the regressions including the residuals of the first stage in the second stage. The results closely map those obtained here.

These results allow us to conclude that the presence of initial code at t_0 reduces the number of category changes in the course of the project (between t_1 and t_2), and that this reduction is positively correlated with the release of code in the course of the project (t_2).[3]

5. Discussion and conclusion

How are joint search processes coordinated? To shed light on this question, we have analyzed the role of initial code release in coordinating open source software development. Our findings have shown that releasing code – an artifact – at the beginning of a joint search process facilitated the coordination of programmers that participated in open source software development. Coordination, in turn, led to subsequent code releases and thus, to the development projects remaining active.

Our findings provide the basis for elaborating theory on the coordination of joint search processes.

The first contribution to understanding the coordination of joint search is that artifacts can have an important role in coordinating joint search processes. The role of artifacts and boundary objects for facilitating coordination is well-established (Star & Griesemer, 1989; Carlile, 2002; Bechky, 2003a, 2003b; Okhuysen & Bechky, 2009; Cacciatori, 2008, 2012). As summarized by Okhuysen and Bechky (2009: 475) objects facilitate coordination by direct information sharing, scaffolding (as a reminder of which tasks still need to be done, and who needs to do them, in order to complete the work), acknowledging and aligning work (boundary objects allow people from different groups to acknowledge their progress on the task) and creating a common perspective. Boundary objects are often used for particularly difficult coordination challenges, as a solution actors fall back on when other means of coordination have not worked (Bechky, 2003a, 2003b; Okhuysen & Bechky, 2009). Boundary objects are mediating artefacts that have interpretive flexibility and can be an important means of achieving collaboration, promoting the sharing of knowledge between diverse groups (Star & Griesemer, 1989; Sapsed & Salter, 2004; Barrett & Oborn, 2010: 1200).

Artifacts and boundary objects are seen to play a role for coordination in the distributed innovation literature (Carlile, 2002; Swan et al., 2007), where they are considered to be able to facilitate collaboration across knowledge boundaries (Barrett & Oborn, 2010). According to prior literature, the main mechanisms by which boundary objects facilitate coordination are to promote shared representation, transform design knowledge, mobilize for design action, legitimize design knowledge (Bergman et al., 2007), provide a basis for negotiation and knowledge exchange between differentiated communities of practice (Sapsed & Salter, 2004), and translate messages from different epistemic communities and connect such communities (Carlile, 2002). Overall, much of the research on boundary objects emphasizes their role in knowledge sharing and transforming knowledge (e.g., Carlile, 2002, Yakura, 2002, Bechky, 2003a, 2003b, Cacciatori, 2008, 2012).

[3] Note that this econometric analysis could be expanded. We cannot apply directly the Sobel-Goodman test for mediation because we have no OLS and because of the different time frames of the 2 stages that imply different set of controls. To give an idea of the possible result of a Sobel-Goodman test (which is however only indicative), we reproduced that technique comparing the previous equations with an equation testing the direct effect of $\mathrm{INITIAL\_CODE}_{t0}$ on $\mathrm{CODE\_RELEASED}_{t2}$ (including the controls from stage 1 and from stage 2 first alternatively and then contemporaneously, in order to take into account the phenomena relative to the whole time span). When we use controls from stage 2, the direct effect of $\mathrm{INITIAL\_CODE}_{t0}$ is .8548467 with 95% Conf. Interval (.599205, 1.110488) and z=6.55. A similar figure is obtained when controls from both stage 1 and 2 are considered. Including only controls from stage 1, instead, enhances the effect of $\mathrm{INITIAL\_CODE}_{t0}$ up to 1.568536 (Std. Err.=.1187749) and 95% Conf. Interval (1.335742, 1.801331) with z=13.21. Thus, including $\mathrm{CATEGORIES\_CHANGES}_{t1\_t2\_p}$ reduces the coefficient and the z values of $\mathrm{INITIAL\_CODE}_{t0}$, but only when early stage phenomena are controlled for. When also later phenomena enter the picture, the remaining reduction - even if still present - is not enough to be significant.
Therefore, when later phenomena enter the picture, the inclusion of $\mathrm{CATEGORIES\_CHANGES}_{t1\_t2\_p}$ reduces the coefficient and the z values of $\mathrm{INITIAL\_CODE}_{t0}$, but not enough to be significant. When early stage phenomena are controlled for, the reduction of the coefficient and the z values of $\mathrm{INITIAL\_CODE}_{t0}$ is clear and significant, confirming our analysis.

In the context of joint search that we analyzed, code as a boundary object had an impact on the coordination of the joint search process but by playing a role that went beyond the roles identified in prior literature on the role of objects for search and for coordination. What appears particularly important for understanding how the release of code at the beginning of a project facilitates coordination of joint search for software solutions is the combination of the point of time at which the artifact is made available and the impact of the artifact itself on coordination. Providing code at the beginning of a project provides a number of opportunities for facilitating coordination of joint search.

(a) Making an artifact available at the beginning of a joint search process provides the different parties to the joint search effort with a common starting point. Prior research on joint search has emphasized the importance of a common starting point: in their study of DuPont, Bhardwaj et al. (2006) identified that DuPont was successful in its search for successful new products because it used 'moving, anchored search', that is, 'search is tethered to the chosen anchor … using the anchor as a guide' (Bhardwaj et al., 2006: 251). Having a starting point for search was important for reaching a successful conclusion of the joint search.

(b) Models of (individual-level) search have also highlighted the importance of providing a 'preferred direction' of search (Winter et al., 2007). As the authors explain, 'preferred direction' provides guidance of the search process in a way that is very different from guidance by feedback on payoffs of alternatives that are sampled, i.e., from deciding the search path in function of the payoffs of the alternatives tried (Winter et al., 2007). We extend their insight by pointing out that guiding search by preferred direction rather than feedback on payoffs will be particularly important when multiple actors are searching together. To the extent that agents are heterogeneous, payoffs might be different across actors, and actors might even draw different conclusions from the same payoffs. Guiding search by payoffs therefore seems prone to lead to misaligned actions in the case of joint search. Thus, relying only on payoffs seems much more difficult for joint search. Search based only on feedback on payoffs (in Gavetti & Levinthal's (2000) terms, backward-looking search) is therefore much less powerful than in individual-level search. The implication is that in joint search, the alternative to backward-looking search identified by Gavetti & Levinthal (2000) is more attractive than in individual search: forward-looking search, i.e., search that is not guided by feedback from alternatives that actors have sampled (but otherwise, for instance by actors' cognitive maps of action-outcome linkages) (Gavetti & Levinthal, 2000). Thus, artifacts can have an important role in coordinating joint search processes. They facilitate such coordination not only by facilitating knowledge sharing and transforming knowledge as identified in prior literature, however. Rather, when released at the beginning of joint search processes, they provide a common starting point and a preferred direction. In this case, they have an impact on the coordination of joint search through enabling forward-looking search.

The findings extend prior research on joint search by identifying that artifacts can play a particularly important role for coordinating joint search: they represent an alternative guidance to search processes than guidance from feedback. This is particularly important because in joint search guidance from feedback is not likely to lead to aligned action at least if searchers are heterogeneous as they might receive different feedback and draw different conclusions from it. This insight suggests that theory on joint search needs to reconsider the central role of performance feedback in search, or at least qualify it. Feedback is absolutely central for theories of search (March & Simon, 1958; Cyert & March, 1993; Levinthal, 1997; Gavetti et al., 2007; 2012). In joint search, however, the power of feedback is attenuated: while it still plays a role for identifying the high-performing alternatives, not only is it *not* the solution to the challenge of coordinating the multiple searchers involved but rather, might also make coordination less likely (as heterogeneous agents might get different feedback and/or interpret the same feedback differently, leading to different conclusions). We thus contribute to theory of joint search by identifying a central trade-off in joint search, i.e., between potentially positive consequences of performance feedback with regard to identifying the best-performing alternatives, and potentially negative consequences of performance feedback concerning coordination of multiple searchers. While several trade-offs in search have been identified, such as the trade-off between exploration and exploitation (March, 1991) or between local and distant search (Levinthal, 1997), our findings suggest that in joint search, the trade-off identified above also needs to be considered. We furthermore point to a question for the research agenda on joint search: To what extent should agents use feedback on performance of alternatives they sample in situations of joint search, given that doing so might have drawbacks with regard to coordinating with other searchers?

On the basis of the findings, we also extend theory on joint search by suggesting possibilities for providing coordination in joint search, given the trade-off identified above. Prior literature has identified two major ways for providing coordination in joint search: organization structure (Rivkin & Siggelkow, 2003; Rivkin & Siggelkow, 2005) and providing an anchor (Bhardwaj et al., 2006), a 'seeding' (Levinthal & Warglien, 1999: 349), or a preferred direction (Winter et al., 2007) for search. Our findings from the open source software setting shed light on the second mechanism from an extreme case where organization structure cannot be used as a solution.

By identifying that artifacts released at the beginning of joint search processes can have an impact on the coordination of joint search through enabling forward-looking search, we elaborate existing theory on joint search: building on Bhardwaj et al.'s (2006) 'moving, anchored search', we identify a different instantiation of a similar principle but importantly, also specify why this principle is particularly valuable in the case of joint search. This way of facilitating joint search is different from the way identified by Rivkin & Siggelkow (2003), who consider the impact of different configurations of organizational design variables on joint search. In contrast to the mechanisms for guiding joint search identified by Rivkin & Siggelkow (2003), the mechanism of preferred direction of search (Winter et al., 2007) through artifacts does not require an organization structure that provides the organizational levers for guiding search that Rivkin & Siggelkow (2003) have modelled. This feature does not just map on to precisely the situation of open source software, but also applies to other forms of joint search, i.e. the search for new product designs in distributed innovation or for solving problems across firms, such as with suppliers or in strategic alliances. Further empirical validation of these specific mechanisms and of the findings in the latter contexts seems a fruitful way ahead for research. The findings contribute to joint search theory by emphasizing the importance of providing guidance of the search process wherever organization structure comes to limits with regard to providing coordination. (Note that this is not just confined to settings such as open source software, distributed innovation or networks of firms but can also be the case because of trade-offs in organization design that set limits to designing the organization so it is 'optimized' for coordination purposes).

Furthermore, the findings also shed light on a common principle underlying this mechanism. The underlying principle that is common to the mechanisms employed in joint search that are identified in prior literature such as preferred direction (Winter et al., 2007), moving anchored search (Bhardwaj et al., 2006) or 'seeding' of search (Levinthal & Warglien, 1999: 349) is that of precedence. It is precedence that provides guidance for the multiple searchers involved in the joint search process. There are several mechanisms by which precedence provides guidance and coordination that we know from the literature. Mechanisms such as path dependence (David, 1985), individual habits (Camic, 1986), and organizational routines (Nelson & Winter, 1982) capture the tendency to act in previously adopted ways. By providing a common starting point, joint search is provided with a direction through the establishment of precedence.

We also extend prior theory of search by adding to alternatives to relying on feedback in search that were documented in prior literature. Gavetti and Levinthal (2000), for instance, identify two modes of search, backward-looking (based on feedback) and forward-looking (based on actors' cognitive maps of action-outcome linkages). We identify a different alternative to backward-looking search, an alternative that provides guidance in joint search processes that is forward-looking in Gavetti and Levinthal's (2000) sense, but different from the forward-looking search guided by cognitive frameworks that Gavetti & Levinthal (2000) identify. Namely, it provides guidance without having to rely only on cognitive frameworks. Rather, we point to artifacts made available at the beginning of joint search processes that can have a role as tacit coordination mechanisms (Srikanth & Puranam, 2011) that provide opportunities for example for direct information sharing, scaffolding, acknowledging and aligning work, and creating a common perspective (Okhuysen & Bechky, 2009: 475). Our contribution is to identify a type of tacit coordination mechanism that facilitates alignment of actions by establishing precedence and, in so doing, fosters coordination in joint search.

Against this backdrop, we contribute to understanding of the role of artifacts in providing coordination of joint search. To see the contribution of artifacts clearly, note that in joint search, two problems need to be solved at the same time: finding alternatives with high payoffs and aligning the actions of multiple agents. Backward-looking search comes to limits in such a situation because to the extent that agents are heterogeneous, they might get different feedback and/or interpret feedback differently, leading to a misaligned actions (a coordination problem) if they only follow feedback. The alternative to backward-looking search is forward-looking search (Gavetti & Levinthal, 2000). There is a problem with forward-looking search in situations of joint search, however: cognitive frameworks about action-outcome links only lead to aligned actions if actors share these cognitive frameworks. In many situations of joint search such as OSS communities or distributed innovation with suppliers that is not necessarily the case. Artifacts and boundary objects (such as software code) can have a role as tacit coordination mechanisms (Srikanth & Puranam, 2011) that can facilitate coordination through providing the opportunity for direct information sharing, scaffolding, acknowledging and aligning work, and creating a common perspective as pointed out by Okhuysen & Bechky (2009), thus building up common ground with regard to the 'preferred direction' of search (Winter et al., 2007). By providing these opportunities, artifacts facilitate forward-looking search that is needed to avoid the problem of misaligned actions in situations of joint search.

In summary, we elaborate prior theory on joint search by identifying that where feedback does not easily provide guidance of search (the coordination challenge), alternatives to backward-looking search by reacting to performance feedback become more important. We also extend joint search theory by identifying a type of tacit coordination mechanism that facilitates alignment of actions by establishing precedence and, in so doing, fosters coordination in joint search. The findings also suggest that such a mechanism is able to substitute hierarchy and organization structure as mechanisms that provide coordination of multiple actors in search processes. Thereby, the findings suggest that theory of joint search might be incomplete without considering tacit coordination mechanisms that establish precedence, adding a point on the research agenda that holds potential for future research.

References

Baldwin, C. Y., K.B. Clark. 2000. Design Rules: Volume 1. The Power of Modularity. Cambridge, MA: MIT Press.

Baldwin, C.Y., K.B. Clark. 2006. The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? Management Science, 52(7): 1116-1127.

Barrett, M., E. Oborn. 2010. Boundary object use in cross-cultural software development teams. Human Relations, 63(8): 1199-1221

Bechky, B.A. 2003a. Object Lessons: Workplace Artifacts as Representations of Occupational Jurisdiction. American Journal of Sociology, 109(3): 720–52

Bechky, B.A. 2003b. Sharing Meaning Across Occupational Communities: The Transformation of Understanding on a Production Floor. Organization Science, 14(3): 312–330

Bergman, M., K. Lyytinen, G. Mark. 2007. Boundary Objects in Design: An Ecological View of Design Artifacts. Journal of the Association for Information Systems, 8(11): 546-568

Bhardwaj, G., J.C. Camillus, D.A. Hounshell. 2006. Continual Corporate Entrepreneurial Search for Long-Term Growth. Management Science, 52(2): 248-261

Brusoni, S., Prencipe, A., K. Pavitt. 2001. Knowledge specialization, organization coupling, and the boundaries of the firm: Why do firms know more than they make? Administrative Science Quarterly, 46(4): 597-625

Cacciatori E. 2008. Memory objects in project environments: Storing, retrieving and adapting learning in project-based firms, Research Policy, 37(9): 1591-1601.

Cacciatori, E. 2012. Resolving Conflict in Problem-Solving: Systems of
Artefacts in the Development of New Routines. Journal of Management Studies 49(8): 1559-1585

Camic, C. 1986. The Matter of Habit. American Journal of Sociology, 91(54): 1039-1087

Carlile, P.R. 2002. A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development. Organization Science, 13(4): 442-455.

Chesbrough, H. 2003. Open Innovation. Free Press, New York

Chua, C.E.H, A.Y.K. Yeow. 2010. Artifacts, Actors, and Interactions in the Cross-Project Coordination Practices of Open-Source Communities. Journal of the Association for Information Systems, 11(11/12): 838-867

Clark, K. B. and Fujimoto, T. 1991. Product Development Performance. Boston, MA: Harvard Business School Press.

Cohendet P., Creplet F., O. Dupouët. 2001. Communities of practice and epistemic communities: a renewed approach of organizational learning within the firm, presented at the "Workshop on Economics and Heterogeneous Interacting Agents", Marseille, June, 2001.

Crowston, K., Q. Li, K. Wei, U.Y. Eseryel, J. Howison. 2007. Self-organization of teams for free/libre open source software development. Information and Software Technology 49(6): 564–575

Crowston, K., K. Wei, J. Howison, A. Wiggins. 2012. Free/Libre Open-Source Software Development: What We Know and What We Do Not Know. ACM Computing Surveys, 44(2): Article 7

Cyert, R.M., J.G. March. 1963. A Behavioral Theory of the Firm. Blackwell, Oxford.

D'Adderio, L. 2003. Configuring software, reconfiguring memories: the influence of integrated systems on the reproduction of knowledge and routines, Industrial and Corporate Change, 12(2): 321-350.

Dahlander, L., and S. O'Mahony (2011). Progressing to the center: Coordinating project work. Organization Science, 22(4): 961–979

Dahlander, L., D.M. Gann. 2010. How open is innovation? Research Policy. 39(6): 699–709

Dalle, J.-M., P. A. David. 2005. The allocation of software development resources in 'open source' production mode. J. Feller, B. Fitzgerald, S. A. Hissam, K. R. Lakhani eds. Perspectives on Free and Open Source Software. MIT Press, Cambridge, MA., 297–328

Dalle, J.-M., and Jullien, N. 2003. 'Libre' software: Turning fads into institutions?, Research Policy, 32(1):1-11.

David, P.A. 1985. Clio and the Economics of QWERTY. American Economic Review, 75(2): 332-337

David P.A. and Ghosh R.A. 2008. Relating social structure to technical structure: A study of the Linux kernel, presented at the DIME - DRUID Fundamental on Open and Proprietary Innovation Regimes:

"Opportunities and limitations of the open source models of innovation and the role of intellectual property rights", Copenhagen Business School, Copenhagen, Denmark, June 17, 2008

David P.A., Waterman A., S. Arora. 2003. The free/libre/open source software survey for 2003, preliminary draft, September 2003, quoted with authors' permission, at http://www.stanford.edu/group/floss-us/report/FLOSS-US-Report.pdf.

David, P.A, F. Rullani. 2008. Dynamics of Innovation in an "Open Source" Collaboration Environment: Lurking, Laboring and Launching FLOSS Projects on SourceForge, Industrial and Corporate Change, 17(4), p. 647-710.

David, P.A., J.S. Shapiro. 2008. Community-based production of open-source software: What do we know about the developers who participate? Information Economics and Policy, 20(4): 364-398

de Souza, C., J. Froehlich, P. Dourish. 2005. Seeking the Source: Software Source Code as a Social and Technical Artifact. GROUP '05 Proceedings of the 2005 international ACM SIGGROUP conference on supporting group work, 197-206

Edmondson, A.C., R.M. Bohmer, G.P. Pisano. 2001. Disrupted Routines: Team Learning and New Technology Implementation in Hospitals. Administrative Science Quarterly, 46(4): 685-716

Eisenhardt, K, M.E. Graebner. 2007. Theory Building from Cases: Opportunities and Challenges. Academy of Management Journal, 50(1): 25–32

Elliott M., W. Scacchi. 2003. Free Software Developers as an Occupational Community: Resolving Conflicts and Fostering Collaboration, Proc. ACM Intern. Conf. Supporting Group Work (Group'03), Sanibel Island, FL, November 2003: p. 21-30

Fleming, L. 2001. Recombinant Uncertainty in Technological Search. Management Science, 47(1): 117-132

Fleming, L., O. Sorenson. 2004. Science As A Map In Technological Search. Strategic Management Journal, 25(8-9): 909-928

Gavetti, G., D.A. Levinthal. 2000. Looking Forward and Looking Backward: Cognitive and Experiential Search. Administrative Science Quarterly, 45(1): 113-137

Gavetti, G., H.R. Greve, D.A. Levinthal, W. Ocasio. 2012. The Behavioral Theory of the Firm: Assessment and Prospects. Academy of Management Annals 2012, 6(1): 1-40

Gavetti, G., D.A. Levinthal, W. Ocasio. 2007. Neo-Carnegie: The Carnegie School's Past, Present, and Reconstructing for the Future. Organization Science, 18(3): 523–536

German, D. M. 2003. "The GNOME Project: a Case Study of Open Source, Global Software Development," Software Process: Improvement and Practice (8) 4, pp. 201-215.

Ghosh R.A., Krieger B., Glott R., Robles G., 2002. Free/Libre and Open Source Software. Part IV: Survey of Developers, International Institute of Infonomics, Berlecom Research GmbH, at http://www.infonomics.nl/FLOSS/report/Final4.pdf

Giuri P., Rullani F., S. Torrisi. 2008. Explaining Leadership in Open Source Software Projects, Information Economics and Policy, 20(4): 305-315.

Giuri, P., M. Ploner, F. Rullani, S. Torrisi. 2010. Skills, Division of labor and performance in collective inventions: Evidence from open source software, International Journal of Industrial Organization, 28(1) 54–68.

Greene, W.H. 2000. Econometric Analysis, Prentice Hall, New Jersey, 4th ed.

Gulati, R., P.R. Lawrence, P. Puranam. 2005. Adaptation in Vertical Relationships: Beyond Incentive Conflict. Strategic Management Journal, 26(5): 415-440.

Haefliger, S., G. von Krogh, S. Spaeth. 2008. Code reuse in open source software. Management Sci., 54(1): 180–193.

Heath, C., N. Staudenmayer. 2000. Coordination Neglect: How Lay Theories of Organizing Complicate Coordination in Organizations. Research in Organizational Behaviour, 22: 155–193

Hobday, M., Davies, A., A. Prencipe, A. 2005. Systems integration: a core capability of the modern corporation. Industrial and Corporate Change, 14(6): 1109-1143

Kellogg, K. C., W. J. Orlikowski, J. Yates. 2006. Life in the Trading Zone: Structuring Coordination Across Boundaries in Postbureaucratic Organizations. Organization Science, 17(1): 22-44.

Knudsen, T., D.A. Levinthal. 2007. Two Faces of Search: Alternative Generation and Alternative Evaluation. Organization Science, 18(1): 39–54

Kogut, B., A. Metiu. 2001. "Open-Source Software Development and Distributed Innovation," Oxford Review of Economic Policy 17 (2): 248-64.

Krishnamurthy, S., 2002. Cave or community? An empirical examination of 100 mature open source projects. First Monday 7 (6).

Lakhani, K.R., R. G. Wolf. 2005. Why hackers do what they do: understanding motivations and effort in free/open source software projects. J. Feller, B. Fitzgerald, S. A. Hissam, K. R. Lakhani eds. Perspectives on Free and Open Source Software. MIT Press, Cambridge, MA., 3–21.

Langlois, R.N., G. Garzarelli. 2008. "Of Hackers and Hairdressers: Modularity and the Organizational Economics of Open-source Collaboration", Industry & Innovation, 15(2), 125-143

Lanzara G.F., M. Morner 2005. Artifacts rule! How organizing happens in opens source software projects, in: Czarniawska B. and Hernes T., Actor-Network Theory and Organizing, Copenhagen, Copenhagen Business School Press, p. 67 - 90.

Laursen, K., A. 2006. Open for innovation: The role of openness in explaining innovation performance among U.K. manufacturing firms. Strategic Management Journal, 27(2): 131-150

Lee G.K., R.E. Cole. 2003. From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development, Organization Science, 14(6): 633-649.

Lerner J., Tirole J. 2002. Some simple economics of Open Source, The Journal of Industrial Economics, L(2): 197-234.

Levinthal, Daniel A. 1997. Adaptation on Rugged Landscapes. Management Science, 43(7): 934-950

Levinthal, D.A., M. Warglien. 1999. Landscape Design: Designing for Local Action in Complex Worlds. Organization Science, 10(3): 342-357

Lin, Z., K.M. Carley. 1997 Organizational response: The cost performance tradeoff. Management Science, 43(2): 217-234

MacCormack, A., Rusnak, J., Baldwin, C. Y. 2006. Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Management Science, 52(7)

March, J.G. 1991. Exploration and exploitation in organizational learning. Organization Science, 2(1): 71-87

March, J.G., H. Simon. 1958. Organizations. Blackwell, Oxford

Markus M.L. 2007. The governance of free/open source software projects: monolithic, multidimensional, or configurational? J Manage Governance, 11(2):151–163

Mateos-Garcia, J., W. E. Steinmueller. 2008. The institutions of open source software: Examining the Debian community. *Inform. Econom. Policy*, 20, 333–344.

Mintzberg, H. 1979. The Structuring of Organizations. Prentice-Hall, Englewood Cliffs/NJ

Mockus, A., R.T. Fielding, J.D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. ACM Transactions on Software Engineering and Methodology, 11(3): 309–346.

Muller P. 2006. Reputation, trust and the dynamics of leadership in communities of practice, Journal of Management and Governance 10(4): 381–400

Narduzzo A., Rossi A. 2005. The Role of Modularity in Free/Open Source Software Development, in S. Koch (ed by), Free/Open Software Development, Idea Group.

Nelson, R.R., S.G. Winter. 1982. An Evolutionary Theory of Economic Change. Belknap Press of Harvard University Press, Cambridge/MA

Nishiguchi, T. 1994. Strategic Industrial Sourcing. New York: Oxford University Press.

Okhuysen, G.A., B.A. Bechky. 2009. Coordination in Organizations: An Integrative Perspective. The Academy of Management Annals, 3(1): 463–502

O'Mahony S., F. Ferraro. 2007. The emergence of governance in an open source community, Academy of Management Journal, 50(5): 1079–1106

Orlikowski, W.J., 1992. The duality of technology: Rethinking the concept of technology in organizations. Organization Science 3(3): 398-427.

Orlikowski, W.J., 2000. Using Technology and Constituting Structures; A Practice Lens for Studying Technology in Organizations. Organization Science 11(4): 404-428.

Powell, W.W, Koput, K.W., L. Smith-Doerr 1996. Interorganizational collaboration and the locus of innovation: networks of learning in biotechnology, Administrative Science Quarterly, 41(1), 116-145.

Raymond, Eric S. (1998): The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. Sebastopol, CA: O'Reilly Associates.

Rivkin, J.W., N. Siggelkow. 2003. Balancing Search and Stability: Interdependencies Among Elements of Organizational Design. Management Science, 49(3): 290–311

Rullani, F, Haefliger, S. 2013. The periphery on stage. Properties, functions, and dynamics of the periphery in the Free/Open Source Software community, Research Policy, 42(4), 941-953.

Sanchez, R., J.T. Mahoney, J.T. 1996. ‘Modularity, Flexibility, and Knowledge Management in Product and Organization Design’, Strategic Management Journal, 17(winter special issue): 63-76.

Sapsed, J., A. Salter. 2004. Postcards from the Edge: Local Communities, Global Programs and Boundary Objects. Organization Studies, 25(9): 1515–1534

Seshadri, S., Z. Shapira. 2003. The flow of ideas and timing of evaluation as determinants of knowledge creation. Industrial and Corporate Change, 12(5): 1099-1124

Siggelkow, N. 2007. Persuasion with Case Studies. Academy of Management Journal, 50(1): 20–24.

Siggelkow, N., J.W. Rivkin. 2006. When Exploration Backfires: Unintended Consequences of Multilevel Organizational Search. Academy of Management Journal, 49(4): 779-795

Siggelkow, N., J.W. Rivkin. 2005. Speed and Search: Designing Organizations for Turbulence and Complexity, 16(2): 101-122

Sojer, M., Henkel, J. (2010) Code Reuse in Open Source Software Development: Quantitative Evidence, Drivers, and Impediments. Journal of the Association for Information Systems 11(12), 868-901

Sosa, M. E., S. D. Eppinger, C. M. Rowles. 2004. The Misalignment of Product Architecture and Organizational Structure in Complex Product Development. Man. Sc., 50(12): 1674–1689

Srikanth K, Puranam P. 2011. Integrating distributed work: comparing task design, communication, and tacit coordination mechanisms. Strategic Management Journal 32(8): 849–875

Star, S.L., J.R. Griesemer 1989. Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. Social Studies of Science 19(3): 387-420

Swan, J., M. Bresnen, S. Newell, M. Robertson. 2007. The object of knowledge: The role of objects in biomedical innovation. Human Relations, 60(12): 1809-1837

Takeishi, A. 2001. ‘Bridging inter- and intra-firm boundaries: management of supplier involvement in automobile product development’, Strategic Management Journal, 22(special issue): 403-433.

Takeishi, A. 2002. Knowledge Partitioning in the Inter-Firm Division of Labor: The Case of Automotive Product Development. Organization Science, 13(3): 321-338.

Thompson, J.D. 1967. Organizations in Action – Social Science Bases of Administrative Theory. McGraw-Hill, New York.

van den Hooff, B. 2004. Electronic Coordination and Collective Action: Use and Effects of Electronic Calendaring and Scheduling. Information and Management, 42(1): 103-114.

von Hippel, E. 1988. The Sources of Innovation. Oxford University Press, Oxford.

von Hippel, E., G. von Krogh, G. 2003. Open source software development and the private-collective innovation model: Issues for organization science, Organization Science 14(2): 208-223

von Krogh, G., Haefliger S, Spaeth, S., Wallin M.W. 2012. Carrots and Rainbows: Motivation and Social Practice in Open Source Software evelopment. MIS Quarterly.

Wheelwright, S., Clark, K. 1992. Revolutionizing Product Development. New York: Free Press.

Winter, S.G., G. Cattani, A. Dorsch. 2007. The Value of Moderate Obsession:
Insights from a New Model of Organizational Search. Organization Science, 18(3): 403–419

Womack, J.P., D.T. Jones, D. Ross. 1990. The Machine that Changed the World, Rawson Ass.: New York.

Yakura, E.K. 2002. Charting time: Timelines as temporal boundary objects. Academy of Management Journal, 45(5): 956-970.

Zirpoli, F., M.C. Becker. 2011. The limits of design and engineering outsourcing: performance integration and the unfulfilled promises of modularity. R&D Management 41(1): 21-43

Appendix

Table A1 – An example of the category tree for the main category "Environment"

- Environment
    - B20
    - Cocoa (MacOS X)
    - Console (Text Based)
    - Handhelds/PDA's
    - No Input/Output (Daemon)
    - Other Environment
    - Web Environment
    - Win32 (MS Windows)
    - Console (Text Based)
        - Curses
        - Newt
    - X11 Applications
        - Gnome
        - KDE

Table A2 – Correlation table of the main variables used in the regression (N=5703)

		1	2	3	4	5	6	7	8	9	10
1	CATEGORIES_CHANGES$_{t1\_t2}$	1									
2	INITIAL_CODE$_{t0}$	-0.0148	1								
3	CODE_RELEASED$_{t1}$	0.2288*	0.3398*	1							
4	CODE_RELEASED$_{t2}$	0.1494*	0.2382*	0.4595*	1						
5	REGISTRATION_DATE$_{t0}$	0.0215	-0.0113	0.0301*	0.0114	1					
6	NUM_MEMBERS$_{t1}$	0.0482*	0.0149	0.0981*	0.1033*	-0.0394*	1				
7	MEMBERS_TENURE$_{t1}$	-0.003	-0.0270*	-0.0299*	-0.0297*	0.0348*	0.0704*	1			
8	USE_FORUMt$_0$	-0.0105	-0.1095*	-0.1015*	-0.0693*	0.0205	-0.0742*	0.0831*	1		
9	USE_CVS_TOOL$_{t0}$	-0.0510*	-0.1099*	-0.0700*	-0.0466*	0.0074	0.0145	0.0199	0.3174*	1	
10	NUM_MEMBERS$_{t2}$	0.0798*	0.0265*	0.1389*	0.1507*	-0.0280*	0.8977*	0.0603*	-0.0961*	0.0047	1
11	MEMBERS_TENURE$_{t2}$	0.0086	-0.0238	-0.0184	-0.0198	0.0386*	0.0789*	0.9815*	0.0799*	0.0191	0.0815*

Table A3 – Summary statistics of the main variables used in the regression (N=5703)

	N	Mean	Std.	Median	Min	Max
1 CATEGORIES_CHANGES$_{t1\_t2}$	5703	0.35683	1.618404	0	0	21
2 INITIAL_CODE$_{t0}$	5703	0.442574	0.496735	0	0	1
3 CODE_RELEASED$_{t1}$	5703	0.190251	0.392533	0	0	1
4 CODE_RELEASED$_{t2}$	5703	0.097142	0.296177	0	0	1
5 REGISTRATION_DATE$_{t0}$	5703	1109589749	2227664	1109550605	1105748133	1113608824
6 NUM_MEMBERS$_{t1}$	5703	1.619499	1.5751	1	1	39
7 MEMBERS_TENURE$_{t1}$	5703	1074318797	43518509	1074318797	942363968	1116026112
8 USE_FORUMt$_0$	5703	0.940032	0.237449	1	0	1
9 USE_CVS_TOOL$_{t0}$	5703	0.962125	0.19091	1	0	1

10 NUM_MEMBERS$_{t2}$	5703	1.725583	1.890252	1	1	41
11 MEMBERS_TENURE$_{t2}$	5703	1074947564	43337612	1094994304	942363968	1142006656