Designing, Producing and Using Artifacts in the Structuration of Firm Knowledge: Evidence from Proprietary and Open Processes of Software Development

Alessandro Rossi

ROCK (Research on Organization, Coordination and Knowledge)

Department of Management and Information Science

Università di Trento

Via Inama 5 38100 Trento, Italy

alessandro.rossi@unitn.it

Marco Zamarian

ROCK (Research on Organization, Coordination and Knowledge)

Department of Management and Information Science

Università di Trento

Via Inama 5, 38100 Trento, Italy

marco.zamarian@unitn.it

Quaderno DISA n. 116/2006

Mimeo version, December 28, 2006. The authors wish to thank the participants to the XXII EGOS Colloquium (Sub-theme 14: Technology, Organization and Society: Recursive Perspectives) – Bergen – 6-8 July 2006 – for helpful comments on a previous draft. Financial support from MIUR (projects FIRB03 and PRIN05) is gratefully acknowledged. The usual disclaimer applies.

Abstract

In the paper we study the recursive nature of artifacts in the production and the socialization of organizational knowledge. In this respect, artifacts are interpreted both as the product (output) of organizational knowledge processes and, at the same time, as tools easing the development of other artifacts. We compare different practices of knowledge creation and diffusion in complex software production processes with the aim of understanding the effects of interplay between (1) coordination and control practices, (2) mediating artifacts and development tools, and (3) interactions between different actors in the development process. We aim at identifying the peculiar traits emerging in contrasting development paradigms, namely the closed, fully proprietary one widespread in the gaming console industry, and the open model of free/open source software development.

Keywords: video/computer game industry – artifacts – Free/Open Source Software – video game consoles.

In this paper we study the recursive nature of artifacts in the production and the socialization of organizational knowledge. In this respect, artifacts are interpreted both as the product (output) of organizational knowledge processes and, at the same time, as tools to ease the development of further knowledge.

Specifically, in this paper we compare different practices of knowledge creation and diffusion in complex software production processes with the aim of understanding the effects of the often neglected interplay between 1) coordination and control practices, 2) mediating artifacts and development tools, and 3) interactions between different actors in the development process. In particular, we aim at identifying the peculiar traits emerging in contrasting development paradigms, namely the closed, proprietary model, which is widespread in the gaming console industry, and the Free/Open Source software development. Comparisons between proprietary and F/OSS software development processes are, oftentimes, characterized in terms of the relationship between property rights, revenue distribution and power within a network of actors.

We believe that by extending this comparison to include 1) knowledge exchanges between the relevant actors, 2) the different strategies they employ to overcome asymmetries in information and knowledge, and 3) the relevant tools they adopt to implement these strategies, we can further the lively debate animating organizational literature on the dual nature of artifacts, stressing the importance of the interplay between their technical features and the normative meanings they can assume. Moreover, our contribution can help shed some light on recent developments in the video gaming industry, which is moving from a proprietary structure in the software development tools to the provision of open technologies.

The paper is organized as follows. In section 1 we briefly outline the main contributions on the recursive nature of organizational production of artifacts, focusing on the structuring role of artifacts with respect to the organizational structure. In section 2 we propose a reinterpretation of the role of artifacts as mediating instruments for indirectly regulating inter-organizational relationships characterized by differing degrees of one part's control over the production process. In section 3 we give a brief account of our methodological choices. In section 4, we outline the first case, taken from an industry characterized by a strict proprietary control, namely the development of a videogame

software for consoles. We will show how the control in the production of intermediate artifacts helps one of the firms involved to regulate the work of the downstream development process by means of imposing the use of certain "intermediate" tools. This control structure partly substitutes and partly integrates the more traditional control process, typically based on contracts and observation of practices. In section 5 we present the case of a software project developed from a F/OSS community, characterized by a fuzzier definition of borders between the roles of individual actors in the development process, and the rights and control they have over the code. Finally, in section 6 we compare and discuss the two cases, suggesting possible paths along the way of a hybridization process that seems to be emerging in the practices of some of the major players in the video-game industry.

1. The organizational role of artifacts: contrasting deterministic and structuring approaches

The first step in our analysis consisted in a review of the main approaches attempting to explain the role that artifacts can play in fostering and developing organizational capabilities. There are two different reasons that compelled us to devote our attention exclusively to Information Technology artifacts. On the one hand, obviously, they represent the prevailing type of artifacts that are employed in the software development industry, which is the focus of our paper. On the other hand, Information Technology artifacts are ubiquitous, both in the social and in the organizational environments.

After many years of research in this field, it is becoming clear that using deterministic approaches, originally developed to study the impact of non information-related technologies on organizations, is not very fruitful. Several authors have clearly shown that empirical findings are very ambiguous and contradictory (Robey 1977, Attewell and Rule 1984, Fulk and DeSanctis 1995, Robey and Boudreau 1999). Deep and numerous contradictions occur not only across studies, but also within studies, and most of them cannot be solved with just a refinement of the research methodologies, but they call for a deep rethinking of the theoretical approaches and the analytical models. Several researchers have proposed to use new and more complex frameworks, new concepts and new research methods (Barley, 1986, 1990; Orlikowski & Baroudi, 1991;

Orlikowski, 1992; DeSanctis & Poole 1994; Sahay 1997; Griffith, 1999). While all these proposals offer new and very interesting insights, we argue that a crucial point needs to be made: artifacts should not be considered as "external shocks" impacting on organizations in complex ways, but as structuring devices and, at the same time, as outcomes of the organizational processes themselves.

For these reasons, we chose to regard artifacts as negotiated, and embedded sets of rules for goal oriented action (Norman, 1991, 1993). Their organizational role is shaped by the same goal oriented activities for which they are conceived, implemented, and used. Our conceptual approach of artifact is, thus, similar to the concept of 'instrument' proposed by Rabardel (1995), where acceptance by the user plays a key role in defining actual behavior (Davis et al., 1989): only the actor's intention in the use of a tool can give it the quality of artifact. As a portion of the rules that inform organized action, the artifact is relevant at the organizational level as a means of regulation for distributed (Engestrøm, 1991) and situated (Suchman, 1987) activities.

We believe that our approach can be especially fruitful for the analysis of Information Technology artifacts, and in particular for the analysis of software-based, or 'logical' artifacts. The obvious way to describe software artifacts is, in fact, to define them as sets of rules that, on the one hand, are executed by a machine, and, on the other hand, have a regulatory impact on work activities.

2. Why are artifacts relevant from an organizational standpoint?

We can define artifacts as a visible translation of the knowledge developed within and outside organizations. They are a sort of crystallization of effective solutions to recurrent problems: developing and using artifacts that embed these solutions allows the sharing of these 'filtered' forms of experience.

In turn, artifacts are devices that can have a decisive role in the processes of structuration of the organizational action. Mainly, this role is due to their cognitive value.

Norman (1991, 1993) observes that the items he calls 'cognitive artifacts' have two purposes.

First, they support the use of scarce cognitive resources in various ways, and they can guide the representation of relevant knowledge in order to better define and shape the problem to be solved. As an example, we can think of the checklists we use before going to a grocery store, or the ones we use to simplify the packing process before a long trip: these lists are simple artifacts conceived and built with the purpose of focusing our attention in a complex or stressful situation. In a problem-solving perspective, the artifact embeds information that is useful to decompose a complex problem into an ordered set of sub-goals: solving all the sub-goals allows, in turn, to solve the complex problem that was set at the beginning.

Second, artifacts have an important social meaning as they represent receptacles of common experience which is shared within the group that conceives, builds and uses them (Hutchins, 1991, 1995). Artifacts are, at the same time, part of a shared organizational language and a tangible expression of the organizational values.

Organizations, then, just like individuals, build artifacts with the goal of structuring actions. In fact the artifact itself is often meant as a device to influence and shape, in different ways, individual and group decisions and actions.

Moreover, the introduction of artifacts that were designed to help solving a particular problem can, in fact, produce a new kind of problem for the user. He or she needs to be able to 'read' and verify the domain of application for the artifact. As an example we can hypothesize the introduction of a standard operating procedure (SOP). An SOP changes in a profound way the nature of the problems an actor is supposed to solve. The actor needs to be able to analyze the situation, in order to understand if the criteria to apply the procedure are met. Once the actor has taken the decision to activate the procedure, the SOP can structure the action in a very precise way, and the actor will not have to take a whole set of decisions that used to be a part of his/her representation of the situation before the adoption of the SOP.

To summarize, if artifacts are conceived as sets of negotiated, sedimented and embedded rules, their organizational relevance can take many different shapes, that are not always present at the same time, and that are not necessarily mutually exclusive.

First, artifacts can be a means to improve rationality, whether it is individual, collective or organizational: as sets of rules, they can modify the representation of problems, they

can further the transmission of knowledge and information, and they can make explicit a set of shared symbolic values.

Second, artifacts can be, as well, a direct or indirect means of regulation and structuration of the organizational action (providing rules and resources), with results that can vary considerably, in form and content.

IT artifacts and organizational decisions

Our interpretive framework is based on the concept of bounded rationality (Simon, 1947; March and Simon, 1958). The concepts of choice and decision process that we are using imply all those informational, social and cognitive limitations and biases that are included in Simon's approach.

We consider organizational structuring as the collective outcome of a complex and dynamic set of interactions between different, bounded rational decision processes, enacted by individuals and/or collective actors.

IT artifacts have an important role for organizational structuring. This importance can be better understood if we consider three analytical sets of decision processes related to the presence of artifacts in organizations: design, adoption and use decisions. In the remaining sections of the paper we employ an analytical framework we presented elsewhere (Masino and Zamarian, 2003) to describe the role of Information Technology artifacts for the regulation of work activities related to the development of complex video-game software products, stressing the recursive nature of the process generating the artifact itself, and the interplay between relational and technical features of the technologies involved.

3. Method and data collection

Coherently with the exploratory nature of this stage of the research, we decided to proceed in an inductive way starting from a set of interviews and the collection of secondary sources with the key actors in the product design and production processes of two technology leaders operating in the video-game development industry. Since we were trying to improve our understanding of the phenomenon we deemed necessary to use a method that enabled us to capture the dynamics of the processes we were observing and to provide a detailed description of the phenomena themselves.

Within these constraints we decided to introduce some degree of inter firm variability in some key aspects, namely the proprietary structure of rights on software products, in order to refine our understanding of the investigated phenomena (Eisenhardt, 1989, Yin 1984), as we will see in some detail in the sections describing the projects in our sample.

We chose projects as our unit of analysis. There are two main reasons for this choice, given our goal of exploring the meanings and the importance of the relationship between property rights and "power" considerations, on one hand, and competence related problems on the other. In the first place, the focus on projects allows the joint analysis of the stage of design and development of an innovation with the stage of the actual implementation. Secondly, given the strongly asymmetric features of players operating in the very proprietary and closed world of console game development, versus the F/OSS environment, it would have been difficult to properly define the concept of "firm" or even "development team" in the second case. Of course, we felt the need to guarantee some degree of comparability between projects. As these software projects, by design, need to work on very heterogeneous platforms, we decided that complexity measures directly linked to logical features of the software, such as lines of code, were uninformative. On a different level, trying to measure the dimension of the projects in terms of sheer number of people involved also seems wrong, because we can safely presume different levels of involvement between employees in a software firm and volunteer developers in a F/OSS project.

We conducted our first interviews with the project leaders of the two most recent research project in which the two firms were involved in order to be able to fully exploit the memories of the involved actors, along with and archival data. Starting from the concepts and ideas that emerged during the pilot interviews we proceeded to expand the set of people interviewed, in order to include most of the people who actually took part in the design and development of the product.

4. Proprietary game development in the console industry

The typical development process for a console video game can be characterized as in Figure 1.

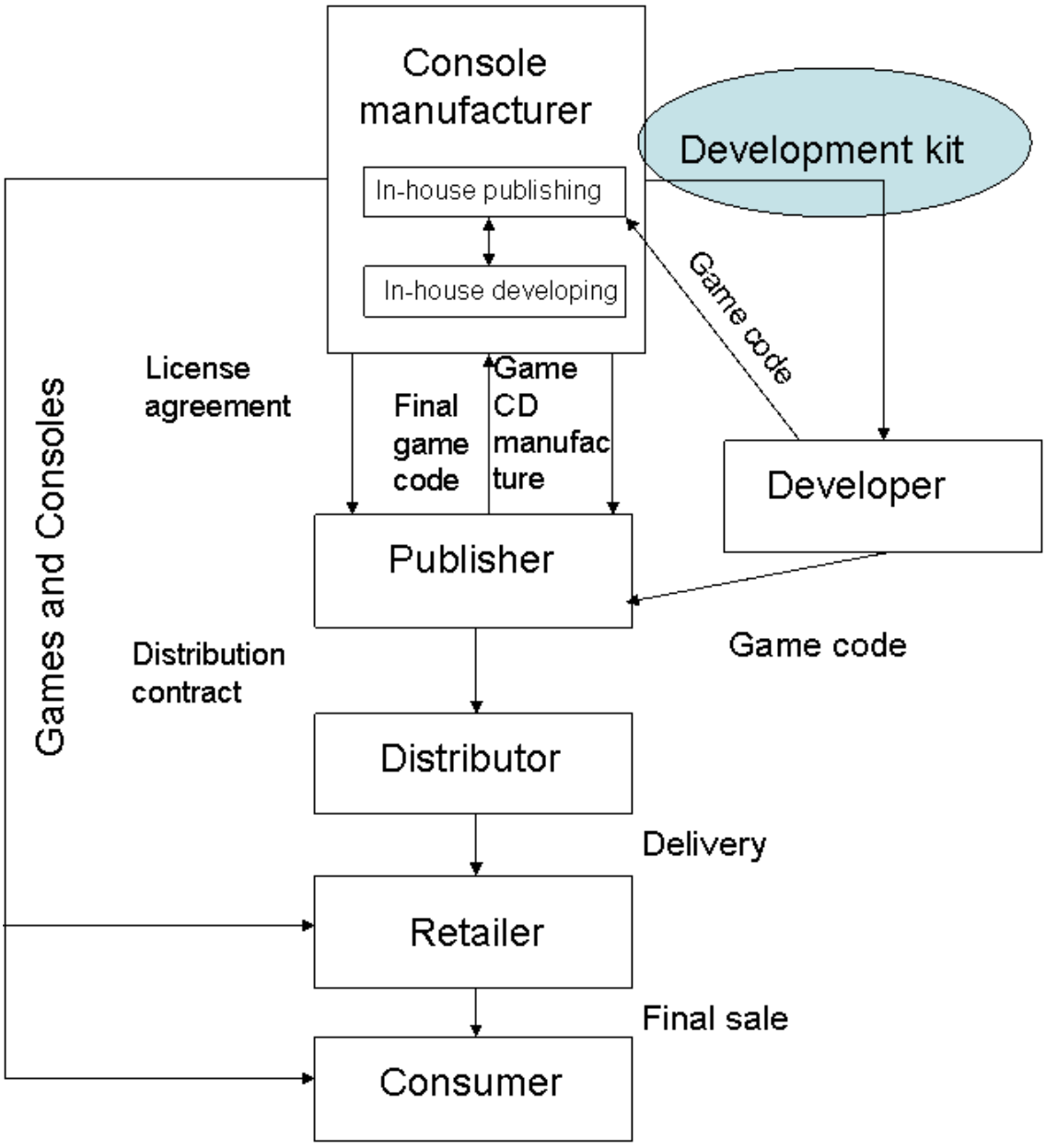

FIGURE 1: Actor's relationships in the video game industry for consoles.
Source: Adapted from Johns, 2006.

Essentially the console producer has two main sources of games for its product. On the one hand it can use internal resources both to design the game (i.e. development teams) and to publish it. On the other, external developers, approved by the console manufacturer by means of a preliminary screening process, can submit project ideas either to internal (to the console producer) or independent publishers. Publishers, in turn, will decide on the marketability of the proposal and, as a consequence, either to

finance or to kill the project. If given approval, the developer produces the game code and releases it to the publisher. At this point the publisher sends the code to the console manufacturer which does two things. First it gives final approval of the software project as a whole after one more round of drive-testing of the code, as in the console market the game needs to be released with zero-defect quality as a goal. The practice, common in the PC games market, of a rushed release that is patched with remote upgrades by the end-user, has not been feasible up to the current generation of consoles. Then, if the release is accepted, the console manufacturer takes care of the production of the physical copies (usually on CD or DVD support) of the game. In turn, the manufacturer releases the copies of the game back to the publisher (again, this can be either internal or independent) that manages the downstream relationships with distributors and retailers.

The complex interplay between these actors is usually described in terms of two kinds of exchanges. On the one hand we have direct monitoring on the output of each phase. The extent o this form of control is linked to the negotiating power of partners, and it finds its natural expression in contracts and legal agreements. The second exchange consists in the financial flows between the involved parties. The interplay between these two kinds of relationships is, thus, coherent with the traditional view of transaction costs economics, and the same explanation applies to the tendency of vertical integration driven either by console manufacturers and large publishers.

With this perspective in mind it appears obvious that developers are described as marginal actors in the network of interdependencies that ultimately generate the end product (Johns, 2006), as they are completely dependent on both the console manufacturers and the (usually much larger) publishers. However, this picture does not capture one of the essential ingredients of developing a successful piece of gaming software, that is, the ability of the network to tap strongly asymmetric sources of ideas, creativity, and technical coding expertise. If we look at this side of the issue, we discover a few layers of complexity that can hardly be considered secondary to explain the whole process.

In fact, we believe that the often neglected part of the network encompassing the social role of development tools and middleware as both control tools and boundary objects

(Yin, 2005) between idiosyncratic domains of knowledge and ideas is key to understand the phenomenon.

In order to do this, in the next section we will take a closer look at the development process in the ultra proprietary domain that characterizes the console market.

Developing games in the proprietary arena: the Milestone case

Milestone is a small multi-platform game developer based in Milan. Born in 1996 as a developer for PC games, it now employs around 40 people, including programmers, artists and producers in just one, fairly large, development team and minimal managerial staff. Milestone moved into the console arena in 2001 with titles for Microsoft Xbox, Sony's Playstation2, and Nintendo Gamecube. When compared to the standard process we described in the previous section, Milestone has two main characteristics that are relevant for our discussion. On the hand, it is a genre specific, niche developer specializing in racing games, a specialization that dates back to the very foundation of the firm. On the other hand, it entered the console market during the second wave of entries for the seventh generation, 128-bit consoles.

These two features have had clear implications in the way Milestone started developing its projects for the console world, in terms of knowledge domain of the development team, project management (i.e. make or buy decisions with respect to the final code production), development tools used, and, of course relationships with other relevant actors in the value chain. In the following subsections we will take a close look at these aspects of the game development operations, starting with a general description of the code-generating process.

Game development at Milestone

Because of its history of early success in developing a racing game which owned its success primarily on how it modeled the physics of motorcycles and on its graphics, Milestone decided to keep developing games based on these same two core components, changing, from time to time only the peripheral aspects of the code. This is the essential reason why, at least in Milestone's case, it does not make sense to use the individual title as focus of analysis, but rather the development team itself. Moreover, and this is more of a general comment on how, historically, they decided to produce

software, Milestone started from a situation in which the components of each game were tightly coupled and moved towards a modular design for each game, with two core modules, several peripheral components and an outer layer of software devoting to the porting (or adaptation) of software into the individual hardware platforms.

In this regard we can look at the different generations of gaming code as a set of incremental innovations on the core components. The first component, in the jargon of the developers we interviewed the "physics engine", can be considered the real core of the game. Essentially, it models how moving objects behave in relation with other objects, trying to fulfill two distinct requirements: to mimic, at least to some extent, real world physics, giving the right sensations to the player; to avoid introducing too much complexity in the interplay between user interface and the modeled physics, in order to avoid frustration and to keep the game playable. This piece of software, and in particular the fine balance between realism and playability, according to the developers themselves, is the key to explain the commercial success of the games: in fact, this core component has not evolved much since its first release in 1996, and its development has always been carried out by a group within the development team. The graphics engine is the second major component of the core. This component takes care of the generation of 3-D models representing bikes, cars, tracks and their features. Albeit being as central as the physics engine, and thus being developed completely in-house, the development of the graphics component has one major difference with the former. Namely, the graphics engine goes through a continuous process of innovation, in order to incorporate and exploit the new features of each generation of hardware (this works both for PC and console). For each major step forward of the available commercial hardware, the graphics engine is reworked in order to make it capable of fully exploiting the *least* performing platform available, with extra add-ons and adaptations for better performing machines.

On a second layer with respect to the two core components, each game is completed with at least four other modules. The most critical ones are the rendering engine, providing the algorithms to generate, in real time, the textures for the surfaces of the moving objects in the game, the sounds component, letting "events" in the game generate appropriate effects, the module managing the user generated input (typically coming from joysticks on PCs, and similar peripheries for the consoles), and the module

regulating the behavior of the computer controlled “players”(AI module). All of these second layer components are produced by Milestone by recurring heavily to third party software development tools. These tools provide two kinds of services. First, they effortlessly allow for the porting process from the high level code, typically produced by the creative artists within the development team, into the specific, compiled code that can be run by any hardware combination. Second, they allow the developers to concentrate on the internal mechanics of the software they are producing, with no need to invest energies in researching the specific hardware architectures of each platform. The importance of these intermediate tools in the development process cannot be overstated: according to the developers trying to interact directly with the development libraries provided by the console manufacturer has several shortcomings, especially for smaller developers, such as Milestone. We can broadly categorize these problems into two main classes. On the one hand we have problems related with the different knowledge domains that the parties control. The distance between competences is particularly relevant in the case of graphic artists (usually part of the peripheral development team) and the manufacturers hard-coders that produce the low-level libraries that are provided with the development kit. This gap, typically, can not be filled by most small developers, hence the decisive importance of the middleware producers.

Technically savvy developers, such as Milestone, on the other hand, can overcome the problem and build in-house the intermediate layers of software necessary to resolve the interdependences. This solution has the definite advantage of allowing for a better fine tuning of software components, thus, generally speaking, generating a better performance. However there are at least two decisive drawbacks associated to the internal development strategy. First, and more general, there are costs considerations: incorporating the physical characteristics of a given hardware architecture into the software development phase, is a long process absorbing many skilled resources. This sort of investment makes financial and operative sense only if the low level software can be developed at the very start of a new console’s life, so that it can be amortized by re-using the libraries themselves for the whole length of the console’s life. Moreover, devoting time and skills to develop architecture specific outer layers of the software can bring about a certain degree of loss of focus in the development team.

In the particular case of Milestone, as they entered the console arena in 2001, when both Sony's Playstation2 ™ and Microsoft's Xbox™ had been around for about a year, they decided that the investment to produce the rendering module and the low-level interface with the development software supplied by the manufacturers were not financially sound. They ended up buying the software from the middleware producer Criterion, which was at the time the developer of Renderware™, the *de facto* standard for rendering engines at the time. Faced with the same decision for the next generation of consoles, Milestone went with a "make" solution to face the same problem. The first reason they gave, is, once again, the prospect of being able to recoup the initial investment exploiting the code over the whole life-cycle of the new consoles. However, a second, critical reason, is that Criterion has been acquired by the large publisher Electronic Arts who chose vertical integration to both acquire an interesting knowledge base and, at the same time, to subtract a valuable resource to competitors. The other modules, albeit not as critical, share the same role in the development process, and go through the same screening process, that ends, typically, with a "buy" solution. An outer layer of software, with respect to these "other" modules, allows for a smooth interfacing between the software actually implementing the game and the hardware platform. Traditionally console manufacturers provided these libraries alongside the hardware part of the development kit. However, with the growing complexity of the consoles, third parties have now entered this market with products mediating between developer produced code and low level proprietary language of the available platforms.
Once implemented by means of these specific tools for the specific platform, Milestone hands the completed project over to the manufacturers quality control team that, after vetting the project, gives the final approval.

Artifact-producing artifacts: middleware and developing tools. As we have seen, illustrating the game production process in Milestone, looking at the production of a complex piece of software, such as a console video game, as the product of an individual team of developers is misleading. In fact, if we look at Figure 2, which reproduces the relationships between the different software objects involved in the production of the final release of the software, it is clear that we can distinguish at least four, hierarchically defined layers of tool.

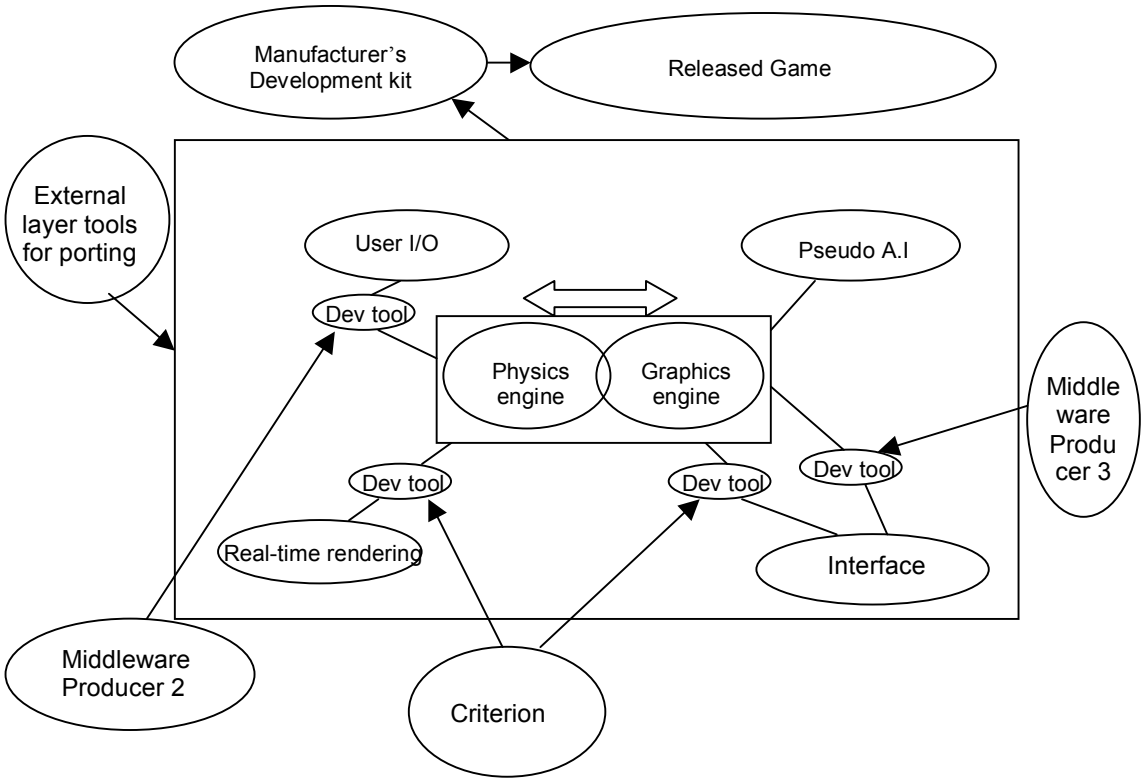

FIGURE 2: Relations between software components in Milestone's development process of a racing game.

In the first layer we have the core components, produced completely in-house and in such a way that they require additional levels of software to become platform specific. On the second layer, we have four peripheral modules, still essential to the mechanics of the game, that connect to the core. These modules have been produces partly by Milestone itself, partly by means of "off-the shelf utilities" available in the middleware market. Then, we have two sources of outer libraries and tools built to help make the code platform specific. Part of these tools is, again, third party, and part come directly from the console manufacturer. Apparently, thus, it seems that the level of control that the console manufacturer can exercise on the developer is strictly defined by either the (rather shallow) quality control processes taking place at the start and at the end of the development phase and, by the mediation of the libraries included in the development kits. However, middleware and development tools providers are not totally independent. In fact, they are selected by the console manufacturer with the same sort of procedure that is in place for selecting developers themselves. Hence the dual layer of control that

the manufacturers can actually subject the developers to. Of course, the end result still depends on the use decisions that the developer makes of these enabling technologies.

The development network of agents

From the network standpoint, Milestone behaves in a way that is pretty similar to the archetype we described in the previous section, with some remarkable differences, mainly due to adoption choices of artifacts, used as boundary spanners between different knowledge domains. From the point of view of vertical relations in the value chain, Milestone has always been considered a top developer in the market, cooperating with powerful publishers, such as Infogrames (the leading European publisher) and Electronic Arts, building these critical relationships on its reputation as a niche developer, and on a past history of commercially successful productions. These relationships work in a standard way, with respect to the general model. Milestone starts developing a game, as soon as a clear market opportunity in the form of a gap in the game catalogues of the major developers in the market. The project is then presented to prospective publishers who, in turn, decide to adopt and share the financial burden of the development. The console manufacturers, in time, included Milestone in the ranks of their "official" developers after a screening of the past history of the company. After this official endorsement, the manufacturer provided the developing kit and special testing machines, dubbed debugging machines, which can replicate the behavior of the actual, commercial console.

Alongside these basic devices, Milestone makes use of a wide array of other development tools provided for by third parties, the middleware producers. These companies specialize in developing tools that can translate the high-level code typically produced by the creative artists into functioning, compiled pieces of code. Describing the code generation we outlined the different levels at which middleware producers enter in the production process.

Heterogeneity of knowledge domains and inter-firm division of labour: a self-reinforcing loop?

As we have seen, room for independent development tools and middleware producers emerges either when a developer has not the technical expertise to develop the software

itself, or when it does not believe it can recoup the investment. According to the developer we interviewed, the complexity of new generation consoles makes both these circumstances more probable. The upside, from the developer's point of view, is the ability to concentrate on core activities, typically creative solutions to visualization problems (both physics ad graphics in the case at hand) or storytelling and setting elements in other cases. The downside, of course, derives from relinquishing control to third parties of the implementation of these high levels ideas into working code. We need to underscore that, at least in principle, the more complex the new machines, and the faster the cycles of development for these platforms, the less a developer, even a competent one, will adopt a "make" strategy for its middleware. We believe that it is not far fetched to hypothesize a progressive loss of competence on the translation mechanisms from the high level, descriptive languages typically used by creative artists into the lower level code required by the different platforms, if developers drop this practice altogether. On the other hand, middleware and development tools producers might become more and more powerful actors in the value chain, if this trend is confirmed.

5. The development of computer games in the F/OSS community

F/OSS connections with the gaming industry dates back at least to the mid-Nineties and are to be traced at least to two distinctive albeit intertwined trends: efforts undertaken by some large players in the industry in leveraging the user–base potential in the development of *game mods* and the emergence of user/developer communities developing independent gaming projects.

Regarding the former element, it has to be noted that it does exist in computer games a firm attitude by users to temper with the code for enhancing and customizing their gaming experience[1]. Later on, companies such as id Software and Epic Games leveraged the potential of user communities by deliberately distributing software with liberal licensing terms (mandating the distribution of mods as open source software) and

[1] For instance, Wolfenstein 3D (developed by id Software) gained popularity within the modder community since the map format had been reverse engineered, making it possible to edit existing game maps, thus customizing the game.

by support their user–base through the development of specific tools to ease the modding process (Scacchi, 2001).

This trend has been reinforced, in recent years, by the possibility, afforded by on-line gaming communities, to get more people involved in a coherent network of individuals sharing the same gamin interests. It is by means of this superficial involvement that most gamers can become "modders", and, over time, outright developers.

While early communities revolved under projects that built upon large proprietary software later released as open source (as in the case of the OS release of DOOM in 1997), and involved the modification and development of F/OS variants of proprietary versions of games, later on, the emergence of independent communities of user/developers developing original games, has been more and more frequent.

Computer game projects seems to be very popular among F/OS communities: as of June 2006 the Freshmeat.net portal has 3025 registered projects (out of 40k), while the Sourceforge.net repository shows 13582 projects (out of 140k). Computer games that are developed in these communities range from role-playing games (the most popular category), to simulation games, MUD (Multi User Dungeons) games, first person shooters, arcades, to board/card/strategy games. Sourceforge's activity rank shows that among the 100 most active projects 17 belong to the category "Games/Entertainment". Almost 60% of the computer game projects that are present in SourceForce have end users as their intended audience, the remaining ones are software projects such as toolkits, modeling, rendering, animation engines, frameworks that target developers rather than final users.

It is difficult to sketch the development cycle for the typical F/OS computer game project due to the highly heterogeneous nature of F/OS communities. In other words, there is a growing line of research that looks at F/OSS development not in mere terms of technical enhancement but, rather, as a complex socio-technical system (Kuwabara, 2000; Lin, 2005a). While early studies have convincingly contrasted F/OSS development with respect to closed source models, in doing so they subsumed the existence of "a unified community with shared values, motives, and development approaches" (Tuomi, 2004). Overall, these studies have failed to understand the heterogeneous and contingent nature of the processes of F/OSS deployment, development and implementation and of the contexts in which those processes occur. In

this line, Lin (2005) suggests to interpret F/OSS under the lens of a sociological perspective based on the social worlds theory (Strauss, 1978; Gerson, 1983). According to this approach, one has to focus on artifacts, actors and artifacts as the elementary unit of analysis for interpreting the complex nature of the socio-technical processes that occur within F/OSS communities.

De Paoli et al. (2006), in the line of Lin (2005) and Scacchi (2004), suggest that there are several artifacts in the F/OSS communities that have the "ability of connecting different social worlds and sustaining socio–technical interactions". In this line, Lin (2005b) claims that liberal licensing schemes acts as mechanisms involving interested participants (hackers from the F/OSS community and software corporations in the OSS industry) in the innovation process. In particular, the type of license (GPL vs. BSD-like) provides different incentives for the participation in the innovation process, according to the specific business model employed by the firm. For instance, GPL-like licences, contrarily to BSD-like licences, limit the incorporation in proprietary (closed source) projects of incremental and cumulative innovations made upon a given piece of code, thus deterring from the participation firms those business model place a strong emphasis on direct revenues schemes while attracting firms from complementary industries or with complementary business models.

In this line De Paoli *et al.* (2006) claim that "Licenses specify the boundaries of the permission granted by the copyright owner to the user". The authors contrast the static and homogeneous view of F/OSS communities as ideal free worlds as assumed by part of the existing literature, and provide evidence that the free/open identity is actually the result of negotiation between participants in everyday interactions and practices. For instance, the decision upon releasing the code according to a particular licensing scheme is interpreted as the result of the negotiation at the participants level upon the direction in which one wants the technology to go, the boundaries of the community to be set, and so forth. In this sense, difference licenses represent artifacts that structure activities and practices through the imposition of different political and technical boundaries, affecting the participation of actors, how innovation is deployed and implemented, and so on.

Despite the difficulties in deriving a clear cut model of the development process, it is still possible to describe, at a very general level, which processes and practices are very

popular in F/OS computer game projects. In doing so, we will try to highlight the major difference with the proprietary model.

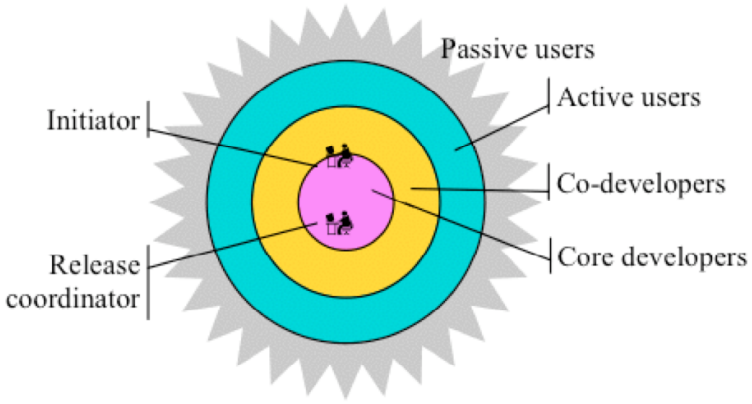

FIGURE 3: A stylized F/OSS development community (Crowston & Howison, forthcoming).

A typical project revolves around a community of "stakeholders" in which end-users' and developers' roles often overlap. The social structure of the project reminds an onion-like structure (as in Figure 3), in which it is possible to distinguish between different roles and levels of involvement in the project. Central for the project are *core developers*, that contribute the largest part of the code for the project and that may take part in fundamental decision-making for the project, such as design of the architecture, division of the project in modules, and so on. External rings represent a more partial involvement in the development activities: co-developers, in this sense, provide smaller parts of code, such as patches and bug fixes, localization activities, or are distinguished by more episodic contribution of larger piece of code. A more external layer is represented by active users that are involved in various feedback activities, such as testing and bug signaling. Most of the time within the project a fundamental role is played by the project leader, that in many cases is the initiator of the project, strongly involved in the development of the early releases of the software and that overtime turns much of his efforts in more general activities of coordination of the whole project (fundamental design decision, involvement of other developers, peer review of the code, and so on).

Typically, the contribution efforts in a single project are very skewed and centralized into a very small subset of participants (Krishnamurthy, 2002). This has been proved to be particularly true for fundamental activities (such as coding), suggesting the existing tension regarding the preservation of the conceptual integrity and the direction of development of the project. The same is, on the contrary, less evident for less critical activities (such as bug tracking and fixing; Mockus et al. 2002), in which more man-power is always welcomed.

The overall picture is rather distant from the idea of a "great babbling bazaar of differing agendas and approaches […] out of which a coherent and stable system could seemingly emerge only by a succession of miracles" (Raymond, 1999); rather, most of the time contributions to the common project is highly monitored and peer-reviewed by various form of hierarchical control, in order to preserve, albeit within an overall highly decentralized development environment, the conceptual integrity of the whole project.

Developing games in F/OSS communities: The Battle for Wesnoth case

The Battle for Wesnoth (from now on, Wesnoth) is a cross-platform, fantasy-themed, turn-based strategy game, released under the GNU GPL license, allowing multi-player gaming over the Internet and with language support for over twenty different languages. The computer game, developed by a voluntary and distributed community of developers, has been labeled as the F/OS counterpart of the year 2005 best seller World of Warcraft, a commercial proprietary Massively Multiplayer Online Role Playing Game (MMORPG) in which users have to pay for an individual license plus a monthly fee in order to access to the on-line multiplayer capabilities. According to Freshmeat.net statistics, Wesnoth ranks 22nd (2nd in the Game subcategory) for user ratings and 240th for popularity (over a population of over 40k registered projects).

David White started to develop Wesnoth on July 2003 and, as it is often the case for many F/OSS projects, conceived the early prototype as a one–man project in order to roughly showcase the potential of the project. In his words:

"Version 0.1 of Wesnoth was developed entirely by me, and it was ugly. It had awful graphics, and no sound or music at all. I think the best way to deal with the problem is to make an early version of the game which showcases the desired gameplay. Then, people with the appropriate skills who like the game will contribute. This worked out

well with Wesnoth, anyhow, as I soon attracted a fine artist, Francisco Munoz, and once the graphics were decent, more people started wanting to help."[2]

White, still the project leader nowadays, was able to rapidly involve other developers and an interested audience and the development process proceeded at a fast pace, thus reaching the first stable release (version 1.0) on October 2005.

The development process is similar to what can be observed in many F/OS communities in the sense that the degree of involvement of developers vary across the various parts and tasks of the project, according to their relative importance. So, if one focus on the most critical activities of development of the project core, it is possible to observe a limited number of core developers involved and a typical pattern of power-law distribution of contribution efforts. If one consider the project according to a broader perspective, there is, on the one side, a larger group of co-developers involved in complementary activities such as bug–reporting or the development of small patches of code and, on the other side, a large involvement of a the user/developers community in the production non core and less critical modules, such as customizations and add-ons. Most of them, such as themes, campaigns, units, scenarios, and other art elements, are distributed under free/open source license and shared across the community using an on-line forum that acts as a meeting point for developers, contributors and mere game users. This feature seems to be very attractive and to explain in part the large popularity of the among computer game players, since it allows a direct involvement of user base in the development of the project. In particular, as put by the project leader "it does blur the line between 'developer created content' and 'user created content' and so […] makes it much easier for any user to contribute to the game"[3].

In essence, the project is structured in a way to allow, both technically and according to its licensing terms, modifications/customizations ranging from mild tailoring to devising a totally new game scenario based on the same game engine (such as in the case of the project known as Spacenoth, that is based on a futuristic/sci-fi theme).

Division of labor and coordination of efforts in achieved using an heterogeneous sets of instruments and strategies, balancing the need for preserving the conceptual integrity of

---

[2] Excerpt from a *Wikinews* interview to David White, June 1, 2006, available at http://en.wikinews.org/wiki/In_the_land_of_the_open_source_elves:_Interview_with_"Battle_for_Wesnoth"_creator_David_White.

[3] See the previous note.

the overall project and, at the same time, the need to exploit man–power and collaborative efforts by interested participants. First of all, the project leader choose to adopt a development style labeled as “evolutionary programming”, starting from a simple prototype and improving the code via incremental modifications, without having in mind a complete pre-planned design of the project. In doing so, a lot of effort has been placed in keep the project as simple as possible (avoiding more complex and up-to-date design options) and replying as much as possible on loose–coupling principles, based on the modularity paradigm, as a way to effectively involve developers in the project, allowing for distributed, concurrent and independent development of specific parts (arts, core engine, network, widgets, and so on). Moreover, larger degrees of freedom in the project, such as access privileges to the CVS, has been granted overtime only to contributors that have provided a credible commitment and contributions aligned to the project aims. Despite these efforts, and given the imperfectly planned design of the project, the quasi-independent modules still allowed for the existence of a non marginal degree of interdependencies between different parts of the project, that was solved superimposing a significant share of “glue code”, largely developed by the project leader, and intended to effectively connecting modules and solving those interdependencies.

Apart from loose coupling and the production of glue code by the project leader, coordination of efforts is pursued also, on one side, via stardardization of activities and coding conventions, and on mutual adjustment enabled by the use of coordination tools that acts as mediating artifacts.

With reference to the former topic, Wesnoth is written in C++ language, employing a minimal set of basic and low–level conventions regarding code writing and relying to the general KISS (Keep it Simple Stupid) principle, mandating simplicity over state-of-the-art and/or complex design. It is worth to note that almost everything needed in the project is coded from scratch without the support of toolkits, frameworks or other middleware technologies. While this choice clearly introduces steep barriers to entry into the project core from interested, albeit not skilled enough users/developers, there is a clear exception that is represented by the existence of very simple toolkits easing non-core contributions from technically inexperienced users in terms of new scenarios, add-ons, campaigns, maps and such, through simple editors.

Mutual adjustment is, typically, pursued using the typical coordination and collaboration tools that are commonly adopted in F/OSS communities, such as the forums, IRC channels, bug-tracking systems, to–do lists, (see for instance van Wendel de Joode et al., 2006, for a review of the principal F/OSS coordination tools).

6. Discussion

As we have seen from the cases, describing the two development processes just in terms of different property rights on the software code produced can be misleading.

A first useful dimension along which to compare the cases can be defined by the actors involved in the two instances we observed. In the F/OSS development process we, typically, have a good level of technical knowledge homogeneity between the actors involved, be they core developers, content providers, or more peripheral users. User/developers are, in fact, a characteristic feature of F/OSS development, as there is no equivalent in the proprietary domain. However, on the other hand, these actors are severely limited from a coordination point of view, being seldom co-located and, usually, not sharing a clearly defined production plan. In the proprietary case, we find the opposite problems. Here, actors involved in the development process do not, typically, suffer from coordination problems. In fact, within company coordination problems are resolved by hierarchy, and inter-firm coordination problems are managed through a double mechanism of contracts and quality control interactions. However, in this case, knowledge quality and heterogeneity constitute the main the problems. On the one hand, users are not considered a meaningful source of feedback, as console games cannot, for all practical purposes, be updated, and, in any case, users lack the necessary technical expertise to directly contribute to the development process. On the other hand, even the actors directly involved possess highly differentiated competences, strictly associated with the phase of the development process they specialize in.

For these reasons, on a second level, we can observe the emergence of two pretty differentiated sets of tools that developers use in the two cases. In the F/OSS world artifacts working as coordination tools (such as IRC channels, CVSs, and the like) are prevalent, whereas, software tools used to directly produce other pieces of code are seldom employed. In the proprietary, console software development market, by contrast,

artifacts bridging high-level, creative software pieces, usually produced by artists or storytellers with a very shallow programming experience and lower-level code that can be compiled on a given platform. The presence of these asymmetries in knowledge domains that characterize the different actors create the room for third parties to introduce tools that end up reinforcing the asymmetries, that is, if a developer, for whatever reason, stops making lower level software, it will, over time, lose that ability, either directly, that is, because of a lack of practice, or indirectly, that is losing tech savvy personnel currently working for them, or failing to attract competent programmers. In this second domain, in contrast with what typically happens in the F/OSS communities, even developers have a very limited say in the way development tools are reshaped and modified by others (typically the console manufacturer, for the "inner" layers of the software kit, and by the middleware producers for the outer layers that interface with the high level language developers typically produce.) In this regard we might add that middleware mediates between the console hardware producer and the end product software developer not merely from a technical standpoint. In fact, the ability to filter, select and scrutinize the middleware producers' work is one of the main tools that the console producer has to indirectly influence the developer's work.

At this same level, by contrast, F/OS computer gaming communities can be interpreted as an archetypal instance of user-driven or community based innovation model, in which final users are viewed as a fundamental driver in innovation generation processes (Mockus et al., 2002; von Hippel and Katz, 2002; Shah, 2005; Haefliger, 2006).

To summarize, then, it is apparent that the interplay between, property rules on the final code, coordination needs, relative heterogeneity of actors' knowledge domains, artifacts they adopt to, in turn produce other artifacts for the final users, is very complex. Moreover, the specific combinations of solutions to this array of problems that characterize each of the development arenas we explored can help us better understand tendencies emerging at the network and power levels between individual actors (Johns, 2006). The evidences we presented in this paper, although still very preliminary and essentially anecdotal, might prove fruitful when trying to explain new trends emerging in the proprietary software development world, that, apparently, are bent towards capturing advantages recognized to the F/OSS development model, especially in terms of speed and depth of the feedback cycle (see, for instance Sony's move towards the

adoption of a GNU development kit for its generation VII new console (PlayStation 3) planned for release on November 2006. In addition it has adopted a variety of open development tools: COLLADA, an XLM-based tool for 3D modelling; PSGL, a modified version of OpenGL ES 1.0 with extensions specifically aimed at the PS3).

References

Attewell, P., & Rule, J. 1984. Computing and organizations: what we know and what we don't know. Communications of the ACM, 27:1184-1192.

Crowston, K. & Howison, J. forthcoming. Hierarchy and centralization in free and open source software team communications. Knowledge, Technology and Policy.

De Paoli, S., Teli, M., D'Andrea, V. 2006. Free and Open Source licenses in communities life: two empirical cases, mimeo.

DeSanctis, G., & Poole, M.S. 1994. Capturing the complexity in advanced technology use: adaptive structuration theory. Organization science, 5(2): 121-147.

DiBona, C., Ockman S., Stone, M. (eds.). 1999. Open Sources: Voices from the Open Source Revolution. Sebastopol, CA, O'Reilly.

Engestrøm, Y. 1991. Developmental work research: reconstructing expertise through expansive learning. In Nurminen, M.I. and Weir, G.R.S. (Eds.) Human jobs and computer interfaces. Elsevier Science Publishers.

Fulk, J., & DeSanctis, G. 1995. Electronic communication and changing organizational forms. Organization science,6(4): 337-349.

Gerson, E. M. 1983. Scientific Work and Social Worlds, Knowledge, Vol. 4.

Haefliger, S. 2006. Permeable development: Technological innovation by users across organizations, mimeo.

Hippel, E. von , and R. Katz. 2002. Shifting Innovation to Users via Toolkits, Management Science, 48(7): 821-833.

Hutchins, E. 1991. *The social organization of distributed cognition*. In Resnick, L.B., Levine, J.M., & Teasley, S.D. (Eds.), Perspectives on socially shared cognition: 283-307. American Psychological Association.

Huysman, M. & Lin, Y. 2005. Learn to solve problems: a virtual ethnographic case study of learning in a GNU/Linux users group. The Electronic Journal for Virtual Organizations and Networks,7.
URL:http://www.ejov.org/apps/pub.asp?Q=1643&T=eJOV%20Issues&B=1 (retrieved May 20, 2006).

Johns, J. 2006. Video games production networks: value capture, power relations and embeddedness. Journal of Economic Geography. Vol 6 pp 151-180.

Krishnamurthy, S. 2002. Cave or community? an empirical examination of 100 mature Open Source projects. First Monday, 7(6).

Kuwabara, K. 2000. "Linux: A Bazaar at the Edge of Chaos", First Monday, volume 5, number 3. URL:http://firstmonday.org/issues/issue5_3/kuwabara/index.html (retrieved 20 August 2002).

Lanzara, G. F., & Morner, M. 2003. The Knowledge Ecology of Open-Source Software Projects. Presented at 19th EGOS Colloquium, Copenhagen.

Lanzara, G. F. & Morner, M. 2005. Artifacts rule! how organizing happens in open software projects. In Czarniawska, B. & Hernes, T. (eds.), *Actor Network Theory and Organizing* (pp. 67 – 90). Copenhagen: Copenhagen Business School Press.

Lanzara, G. F. & Morner, M. 2004. Making and sharing knowledge at electronic crossroads: The evolutionary ecology of open source. Paper presented at the Fifth European Conference on Organizational Knowledge, Learning and Capabilities, Innsbruck, Austria.

Lin, Y. 2004. Hacking Practices and Software Development: A Analysis of ICT Innovation and the Role of Open Source Software. Unpublished doctoral thesis. Department of Sociology, University of York, UK.

Lin, Y. 2005a. The future of sociology of FLOSS. First Monday, Special Issue 2: Open Source. URL: http://www.firstmonday.org/issues/special10_10/lin/index.html (retrieved April 19, 2006).

Lin, Y. 2005b, Hybrid Innovation: How Does the Collaboration Between the FLOSS Community and Corporations Happen?. Knowledge, Technology and Policy.

March, J.G., & Simon, H.A. 1958. *Organizations*, New York: John Wiley.

Masino, G. and Zamarian, M. 2003. Information technology artifacts as structuring devices in organizations: design, appropriation and use issues. Interacting with Computers 15 (2003) 693–707

McKelvey, M. 2001. "Internet Entrepreneurship: Linux and the dynamics of open source software", CRIC Discussion Paper no. 44. Centre for Research on Innovation and Competition, The University of Manchester & UMIST.

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

Norman, D.A. 1991. Cognitive artifacts. In Carroll, J.M. (Ed.), Designing interaction: Psychology at the human-computer interface: 17-38. Cambridge: Cambridge University Press.

Norman, D.A. 1993. Things that make us smart. Reading, MA. Addison Wesley.

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

Orlikowski, W.J., & Baroudi, J.J. 1991. Studying information technology in organizations: Research approaches and assumptions. Information systems research, 2(1): 21-42.

Rabardel, P.1995. Les hommes et les outils. Approche cognitive des instruments contemporaines. Paris: Armand Colin.

Raymond, E.S. 1999. The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. Sebastopol, CA: O'Reilly & Associates.

Robey, D. 1977. Computers and management structure: some empirical findings re-examined. Human relations, 30: 963-976.

Robey, D., & Boudreau, M.C. 1999. Accounting for the contradictory organizational consequences of information technology: theoretical directions and methodological implications. Information systems research, 10(2):165-185.

Scacchi, W. 2001. Understanding the requirements for developing Open Source Software systems. IEE Proceedings Software, 149 (1). 24-39.

Scacchi, W. 2004. Socio-technical interaction networks in free/open source software development processes. In Acuna, S. & Juristo, N., (eds.), Peopleware and the Software Process. World Scientific Press.

Shah, S.K. 2005 Open Beyond Software. In Chris Dibona, Danese Cooper and Mark Stone (eds.). Open Sources 2. O'Reilly Media, Sebastopol, CA.

Simon, H.A. 1947. Administrative behavior. New York: McMillan.

Strauss, A. 1978. A Social World Perspective. Studies in Symbolic Interaction, Vol. 1.

Suchman, L.A. 1987. Plans and situated actions. The problem of human-machine communication. Cambridge : Cambridge University Press.

Tuomi, I. 2004. Evolution of the Linux Credits file: Methodological challenges and reference data for Open Source research, First Monday, volume 9, number 6 (June 2004), URL: http://firstmonday.org/issues/issue9_6/tuomi/index.html (retrieved 10 December 2005).

van Wendel de Joode R., de Bruijn, H. & van Eeten, M., 2006. Software development and coordination tools in open source communities. mimeo.