OSS Quality as Argument: The Impact of Member Roles, Motivations, and Business Models

Benno Luthiger, ETH Zurich, Switzerland

Carola Jungwirth, University of Passau, Germany

Abstract

The attitude of companies towards open source software has significantly changed. Few years ago, the quality of open source software has often been challenged, based on the rule of thumb "If something has no price, it also has no value!" Nowadays, however, the most stated reason why companies use open source software is its apparently high quality. We present the idea that a system of incentives of both private programmers with their different motives to participate and companies paying their programmers for contributing to OSS, are responsible for the software quality—even if all programmers do not pursue a common purpose. The chapter delivers a conceptual framework from an economic perspective showing that every stakeholder can provide valuable input to the success of an open source project. Crowding out between contributors with different motivations does not necessarily exist even if companies with monetary intentions participate. Therefore, we assume OSS as an attractive forum for different interests that can seminally intertwine, while quality software is generated nearly as a by-product.

Introduction

Open source developers produce software—frequently of high quality—that is freely available for everyone. For computer users that are accustomed to purchasing outright the software they use, this might sound puzzling: Is it possible that something *free* is at the same time *good*? Therefore, such computer users, assuming a trade-off, might doubt the quality of open source software and hence refrain from using such software. Computer users that apply software in a business context, for example, may abstain from using open source software even if such software is available at no cost if they are not confident about the quality of such software. Using software in a business context means a heavy investment, even if the license fee of the software used is null because considerable TCO (total costs of ownership) exists. Integrating certain software in the business process leads to a lock-in situation in a manifold way (e.g., investments in human capital, system reorganisation, etc.). Thus, crucial for the future success of open source software is not the fact that there is no associated licensing fee, but the question of the quality of such software.

What can we learn about open source software if we look at the developer's motivations? Understanding the motivations of open source developers allows us to assess the importance software quality has for these software developers and to comprehend the conditions needed for that software quality to be realised. This insight might increase confidence in the quality and sustainability of open source software and, as a result, lower the barrier to use it.

Although examples of high quality open source software are well known, open source still has the connotative impression of being the play area of hobbyists. This impression is not necessarily far from the truth, insofar as most of the open source projects existing on open source platforms (e.g., SourceForge) are indeed the outcome of hobbyists, as various studies show (see Krishnamurthy, 2002; Weiss, 2005, etc.). However, the numbers in our FASD study[1] and in other studies (e.g., Lakhani & Wolf, 2003) indicate that professional developers who are compensated for their work do a significant share of open source software development. Indeed, the commitment for open source projects primarily occurs in their spare time; on the other hand, the share open source projects developed within working time amounts to a considerable 42%. Professional open source projects might be underrepresented in previous studies, like our FASD study, because firms can afford their own project infrastructure and, therefore, are not dependent on the open source platforms we have addressed in our study.

This recognition shows that the motivations on the programmer level on the one side and the motivations on the firm level on the other side should be distinguished. The former concerns developers who commit themselves to contributing to open source projects. The latter concerns developers being motivated simply by the fact that they are advised to create open source software and they are paid to do so. In this case the interesting question is why their employers pay them to work on open source software. This should be a logical consequence of the business models for firms that build on sponsoring open source projects. However, their motivation should be discussed later. First, we analyse motivations on the programmers' level.

Analysing Motivations on the Programmers' Level

One way to resolve the puzzle concerning the open source phenomenon is to focus on the programmer as "prosumer" (see Toffler, 1980). A prosumer is a user who adapts and refines the software according to his or her needs. Von Hippel and von Krogh (2002) showed that such a point of view provides a substantial insight into the understanding of an open source development process.

And it also allows for maintaining the notion of rational actors. Prosumers do not act less rational then software firms investing in software. Both types of actors, the prosumer and the software firm, release the software under an open source license, if the benefits exceed the costs of such an action. Nevertheless, the prosumer's context differs fundamentally from a software firm's context. Software firms are in stiff competition against each other. In such a situation, uncompensated code releases imply high opportunity costs: By revealing the source code, software firms give away their business secret and, therefore, abandon a competitive advantage.

User-developers face a different situation. The interaction of prosumers is determined by only low rivalry conditions and, therefore, by low opportunity costs. Additionally, the direct costs of giving away the source code are rather small considering the Internet. However, small expenses on the cost side cannot explain prosumers' behavior. Even with small costs, it would be superior for rational actors to participate as free riders in the open source area and profit from others' work rather than to engage actively in the creation of such software. Therefore, there must be a selective incentive for contributors—a benefit that only persons who engage can reap. However, in low cost situations only a small selective advantage is needed and the free revealing of the source code is favourable for the user-developer. To understand the open source phenomenon on the programmers' level, we have to identify and quantify the possible selective advantages of open source developers:

A lot of work is done in this field and qualitative studies have identified a variety of motivations that can serve as selective incentives for open source developers.

Use

The simplest reason to engage in an open source project is that developers need a certain application. This is a straightforward implementation of the prosumer model: An actor has a problem that he or she could solve via suitable software. Therefore, they create that software or adapt existing open source software according to need. In the study of Lakhani and Wolf (2003), this motive was named most often. In our study we asked the respondents why they started their open source engagement. Running a cluster analysis, we could identify three distinct types of open source developers and one of them is clearly motivated by pragmatic reasons. The share of pragmatic developers in our sample amounted to about one-fourth.

Reputation and Signalling

In his essay, "Homesteading the Noosphere," part of the well-known trilogy "The cathedral and the bazaar" (2000), Raymond describes how norms and taboos affect the gain of reputation within an open source community. Raymond demonstrates that code forking and, above all, the removal of the contributors' names from the applications' credit files are strongly proscribed. The awareness of the open source community towards these norms makes sense, considering the gain of reputation. Without these norms, it would be hard to track which contribution comes from which person and this would hamper reputation building. Raymond concludes that these norms allow for a reputation game that is essential for the open source movement. It rewards the productive, creative, innovative contributors and thus founds the success of the open source movement.

Lerner and Tirole (2002) interpret the reputation game as signalling incentive. According to their view, the disclosure of the source code in combination with the specific norms governing the open source community as described by Raymond is quite attractive for developers for the following reason. It is very easy tracing the codes of developers back to them and assessing the quantity and quality of their contributions. A developer's status within the project depends on their performance and, therefore, reputation reflects skills, talent, engagement, and all other important characteristics from an employer's point of view. This coherence allows programmers to convert their reputations into cash, for example, by finding a better job offering or by better access to venture capital. One might ask why a firm does not assess the quality of the programmers' skills and talent by itself. However, this is difficult for a person who is not familiar with a special field. If a person's reputation is a valid indicator of his or her talent, this reputation can act as a signal in the sense described above: From the open source project and the person's status within this project, a potential employer can make valid conclusions of the person's talent. Signalling has the best effect in an area with great technical challenges, where the relevant community (e.g., the peer group) is technically experienced, able to distinguish between good and outstanding performance, and capable of esteeming performance and ability (Weber, 2000; Franck & Jungwirth, 2003). In the case of open source, these conditions are exceedingly accomplished. Hann et al. (2004) was able to empirically prove that the status achieved in open source projects under the Apache umbrella had significantly positive effects on the programmer's job income.

Community Identification

Persons perceive themselves not only or not always as independently acting individuals, but they also feel and define themselves as members of a specific group. Therefore, they behave according to the norms and standards of this group. Identification with a group and its goals can explain an individual's actions (Kollock & Smith, 1996). Hertel et al. (2003) examined empirically whether this phenomenon does play a role even in the open source area. In their study among Linux developers, they asked the programmers about various aspects of their activity and correlated this information with data about the developers' engagement. Indeed, using statistical methods, they could verify that a significant amount of the developers' engagement can be explained by their identification with the developer team.

This result has been affirmed by the study of Lakhani and Wolf (2003). In their hacker survey they examined how group identification affects the developer's time engagement. They observed a significant positive effect. Even in the FASD study, we identified a type of contributor motivated by the social context. About one-third of the respondents in the FASD sample belonged to this type (Franck, Jungwirth, & Luthiger, 2005).

Learning

Each social movement offers its participants the possibility to learn and to acquire special capabilities. This aspect is even more pronounced for the engagement in an open source project. Open source projects, furnished with the appeal of programming at the edge of technological innovation, promise to offer extraordinary learning opportunities. In addition, the peer review system specific for the open source area provides timely feedback (e.g., identification of software bugs or suggestions for code improvements) that increases the contributor's learning effect. The desire to improve one's skills as a software developer appears in various studies (see Ghosh et al., 2002; Lakhani & Wolf, 2003; Hars & Ou, 2001, e.g.).

Altruism

Programmers sometimes engage in an open source project with motivations that can be described as altruistic. They contribute, for example, because they use open source software and, thus, feel the obligation to reciprocate. In other cases, they contribute with the intention to aid other people, for example, in developing countries because freely available software helps to bridge the digital divide. In this case, the utility of the open source programmer is increasing with the other's benefit (pure altruism) whereas in the first case the programmer might feel a "warm glow," indicating impure altruism, by doing the right thing (Haruvy et al., 2003). In any case, such contributions can be interpreted as donations. The logic of this interpretation is that making the contribution or not does not have any traceable consequences for programmers themselves. On the other side, the open source project profits from such contributions.

According to the study of Lakhani and Wolf (2003), about one-third of the respondents indicate such motives as relevant for their engagement. Interestingly, Lakhani and Wolf could show via cluster analysis that the ideologically-based argument that software shall be free is close to the reciprocity motive. That would mean that such ideological motives could have an altruistic connotation too.

Fun

Most of the persons developing software perceive this activity as exceedingly fulfilling: “Programming [then] is fun because it gratifies creative longings built deep within us and delights sensibilities we have in common with all men” (Brooks, 1995, p. 8). Torvalds (Ghosh, 1998, p. 9) said as follows: “[M]ost of the good programmers do programming not because they expect to get paid or get adulation, but because it is fun to program. […] The first consideration for anybody should really be whether you’d like to do it even if you got nothing at all back.” Thus, developing software can have an immediate benefit for the programmers that can be named *homo ludens payoff* in accordance with Huizinga (2001).

Open source developers program in their spare time because they consume “fun” with this activity and, therefore, open source software is a by-product of this activity. Indeed, several empirical studies corroborated the importance of fun as motivation to engage in open source projects. As an example, the study by Lakhani and Wolf (2003) showed that 73% of the open source developers experience flow while programming. Although this explanation sounds reasonable, it’s not enough to justify the existence of open source software. The fact that programming can be a fun activity independent of compensation explains only the existence of software developers in general. However, we have to take into consideration that one can earn money by developing software. If we are dealing with rational software developers, the possibility to earn money is without doubt an additional utility. Consequently, we do not expect anyone, at least no rational actor, to program in his or her spare time anymore, because having fun *and* earning money is *mutatis mutandis* better then only having fun. Therefore, if we want to explain the existence of open source software by the fun motive, we have to postulate that having fun while programming is somewhat substitutive to earning money with software development. The open source development has to offer better opportunities to enjoy programming then working in the commercial software area.

In the FASD study, we focused exclusively on the fun motive. The aim of the study was to quantify the importance of fun as motive to engage in open source projects. In addition, we also tried to verify the hypothesis that programming provides more fun in an open source context then under commercial conditions (Luthiger, 2006).

To quantify the importance of fun, we looked at the variation in the open source developers’ engagement and inquired how much of this variation can be explained by the variation of fun the developers enjoy while programming. Thus, the task to quantify the importance of fun becomes an exercise in regression analysis. To master this task, we developed a simple model combining the open source developer’s engagement with the fun he or she enjoys and the amount of spare time he or she has. We used a production function whose input factors, in our case the fun and spare time, have diminishing marginal effects on the output factor, the programmer’s engagement. This can be achieved with quadratic terms having negative signs: $E = c + a_1 * F - a_2 * F^2 + b_1 * T - b_2 * T^2$

where

E: voluntary, unpaid engagement

F: fun T: spare

time a1, a2, b1,

b2 > 0

To operationalise the fun developers generally have while programming, we used the flow construct introduced by Csikszentmihalyi. For the developers’ engagement we used two measures: First we determined the amount of hours the developers spend in their spare time for open source. Second, we asked for their willingness for future engagement in open source projects.

The first regression analysis yielded the result that flow contributes significantly only linearly to the amount of time the developer spends for open source, whereas concerning the availability of spare time, both terms contribute significantly. This means that the joy of programming does not wear off: Each additional unit of fun is transferred linearly into additional commitment. Another result is that the amount of time spent is controlled about ten times more by the available spare time than by the joy of programming. Obviously, the limiting factor concerning the amount of time spent is not the fun experienced while programming but the available time of the programmers. With this model, we're able to explain 33% of the variance in the amount of time the open source developers contribute.

If we look at the determinants considering the willingness to engage for open source in the future, we get another interesting result. Again, flow contributes to the model significantly only with the linear term. This time, however, the available spare time does not contributes to any of the terms significantly. We conclude that when open source developers evaluate their willingness for future engagement, they take into consideration only how much they enjoy programming and neglect completely whether they will have the time needed for future engagement. With this model, we're able to explain 27% of the programmer's open source engagement.

The assumption that programming in an open source project provides more fun than doing this activity under commercial conditions can be tested by comparing the answers of our survey addressing open source developers with those of software developers working in Swiss software firms. We found out that indeed the open source developers enjoyed significantly more fun while programming than commercial software developers. To allow for a systematic bias coming from the comparison of two different samples, we looked for a possibility to compare the experience of flow within the sample of open source developers. Based on the answers about how much of their time they are paid for programming, we have been able two identify two sub samples. We named the first "professionals" because this sample consisted of open source developers that are paid for more than 90%t of their working time in open source projects, whereas the second sample, the "hackers," are paid for less than 10% of their time spent for open source projects. Therefore, our "hackers" stand for the classical open source developers, the "hobbyists," spending their spare time to develop open source software. The comparison of these two samples yielded again that the "hackers" experienced significantly more fun than the "professionals." This result confirms our conjecture that fun is an important driver for the creation of open source software.

Analysing Motivations on the Firm's Level

Because of their own infrastructure, open source activities of firms are underrepresented in the FASD study as well as in other studies. Nevertheless, evidence exists that paid software developers create a significant part of open source software. What incentives do employers have to pay software developers who create a product that is given away for free?

There are two main reasons for an employer to do so: The first reason is that the firm needs a certain software solution for its own use. By opening the source code application or joining an open source project, the firm can lower costs and spread risks. The second reason is that the firm has a business model that builds on open source software.

Use Value

Software developed in-house that has use value for the company and that does not represent any core competency of the company should not cause any losses if the source code is opened. In fact, Raymond (1999) identified two cases where a company can win by doing so.

If a firm operates a Web platform, for example, for selling low margin services or products, it usually needs a Web server, some kind of content management system, and a database. Nowadays, no firm would even think to develop such applications or to pay for the development of such applications. Instead, the firm takes one of those excellent applications released under an open source license and adjusts it to its specific needs. But then, to a certain extent the firm becomes dependent on the continuing existence of this software. Because of its investments in the software, the firm is strongly interested in its survival. In such a situation, it could be reasonable to improve the application's attractiveness and, hence, its user base, by code donations that expand the software's stability or functionality, for example.

Besides this possibility to lower costs, there is also the prospect of risk spreading by open sourcing code. Raymond (1999) exemplified this option by a story from a firm that developed in-house a special print spooling application. After putting the application into operation, the firm decided to release this print spooler under an open source license. The firm's ulterior motive was to stimulate an improvement process for the software. By releasing the code, noticeable problems become evident, other applications are found, and missing features are developed. In essence, the community should be interested in the software and further develop it. Without this move, the firm would have run the risk of letting the application become unmaintained, that is, to let it gradually fall out of sync with technological progress. By releasing the application, the firm could spread the application's maintenance over various independent contributors, thus minimizing the risk that the application goes out of date. If different independent stakeholders are interested in the survival of a software application, the probability grows that the software is kept up to the status quo of the technological progress, whereas none of the different stakeholders can privately appropriate the code.

Business Models

Open source software that is freely available poses a serious threat to the sales value of software. Nevertheless, good reasons exist why it can be worthwhile for firms paying developers to create free software. So-called "business models" describe different situations where firms profit from investing in open source projects. The following discussion bases heavily on the considerations made by Raymond (1999); Hecker (1999); Leiteritz (2004); O'Mahony et al. (2005); Weber (2004); Kalla (2005); Chesbrough (2007):

- **Use:** A company can use open source software if this software provides functionality the company needs for their products or services. Thus, this company does not have to invest money to develop the required software by itself. Instead, the company can rely on the proven quality of the existing software and limit their investments to adapt the software to fit their specific needs (Chesbrough, 2007). IBM's use of the Linux operating system for their servers and Google's use of the Python software language for their services are examples this use case.
- **Open source application provider:** Such application providers create software that they distribute under the terms of an open source license. An "Open source application provider" is a generic term, in which various variations of this business model exist. Most of them succeed based on the existence of a complementary product or service (e.g., "Loss Leader," "Sell it, Free it," "Widget Frosting," "Service enabler"). The basic pattern to generate profits is that by giving away the software for free, the company enlarges the application's user base thus increasing the market for the complementary product.
- **Loss leader:** In the "Loss Leader" model, an application is given away as open source software to improve the company's position in the software market. According to Hecker, the open source product could increase the sales of the complementary software product "by helping build the overall vendor brand and reputation, by making the traditional products more functional and useful (in essence adding value to them), by increasing the overall base of developers and users familiar with and loyal to the vendor's total product line" (Hecker, 1999). Netscape's open source strategy with the Netscape/Mozilla Web browser is an example of this business model.
- **Sell it, free it:** In this variation, the application is sold (i.e., distributed with a commercial license like any commercial product) when it is ready for release. In a later part of the application's life cycle, for example, if the software company has developed a new version of the application, the application's source code (i.e., the older version) is opened. In such a model, the customers buying the software are paying a premium for the value of using the application earlier rather than later. This makes sense

when the application introduces a functionality that is novel in the software market. After opening the code, the freed version can act as a “Loss Leader” for the application’s new version. The later versions of the application can be built on the code of the earlier open source versions. To make this possible, the open source license chosen has to be liberal, that is, it has to allow that derived work can be distributed under a commercial license.

- **Dual licensing:** This is a business model similar to “Sell it, Free it” in so far as the application is available both under a commercial and an open source license. In this model, however, the application is simultaneously distributed under both license schemes. The two versions of the software address different target groups. The free version is intended for users that get familiar with the software by installing and using it and thus preparing the market for it. The open source license chosen for the application’s free version has to be restrictive (e.g., General Public License GPL). Thus, software companies that want to integrate the software into their own applications need the software version with the commercial license. The code base of the two versions is the same but the version with the commercial license delivers additional support or product guarantees (in addition to the right to integrate the software). The well-known MySQL database is a good example of this business model.
- **Widget frosting:** In this model, the complementary product is hardware. For example, a printer manufacturer might release the drivers for their printer under an open source license, thus gaining a larger developer pool and better driver software. In the end, this improves the printers’ acceptance and, therefore, a better market position for the printer manufacturer. In a way, Linux represents the open source software to sell Linux computers, that is, computers preconfigured with Linux, specially designed for an optimal support of this operating system. In fact, some companies do exactly this, and consequently, sponsor the development of the Linux software.
- **Service enabler:** In this business model, the complementary product is neither software nor hardware but a service that generates the company’s revenue stream. The company sells a service online and needs software so that users can access the service. If the community enhances the client software and makes it more user friendly or ports it to new platforms, the market of this service will be expanded.
- **Standard creation:** If a company wants to create a technical standard it can safely use as a foundation to build its proprietary applications, open source can play a crucial role. A company sponsoring a standard faces a serious problem: In order for the initial work to evolve into a standard, it has to be taken up by other companies. How can other companies be convinced to

adopt this standard and to contribute to it? The company that sponsored the code that builds the new standard can level the playing field for potential competitors by open sourcing this code. This works because within an open source environment, the competitors do not have to fear that the initiator can exploit hidden features creating software that is superior to those created by the competitors using the same standard. Moreover, by making the code open source, the initiator signals that contributors can participate in the negotiation about the future evolution of the standard, thus providing incentives for other companies to join it.

This strategy seems only possible for big companies having a long-term policy and the perseverance to pursue it. On the one hand, standard building needs several years of high investment without any return. On the other hand, to appropriate the gains of an established standard to an extent that exceeds the primary investments, the company needs a full portfolio of products and services that can be related to the new standard. An example of this model from practise is IBM's sponsoring of the Eclipse open source project.

The business models above described situations where the companies pay software developers to create open source software that constitutes the basic part of their business model. In other business models the company does not create software but profits as a free rider from open source software and the open source movement. However, by selling their services, they popularise open source software in many ways. Therefore, companies implementing such business models are rather symbiotically related to open source and, thus, well accepted in the open source movement:

- **Support sellers:** The principle of this business model is to sell support for users of open source software. There are two versions known for this business model: Distributors combine different open source applications to assorted and tested distributions (i.e., media and hard copy documentation) that can be sold. The famous company Red Hat pursues this business model very successfully. In another variant of this business model, companies sell technical support for users of open source software. This covers teaching, counselling, system integration, and system tuning, and so forth. The "Support sellers" business model is subject to risk for two reasons. First, the entry barriers of competitors in that market are very low; therefore, stiff competition drives the prices down. Second, the more open source software becomes user friendly, stable, and well documented, the less users of open source software have the need to buy support for such software.
- **Mediators:** The strategy of an open source mediator is to operate a hardware and software (e.g., collaborative tools) platform where open

source projects can be hosted. Gains can be obtained by selling advertising space (banners). The "Mediator" model is characterised by a "winner take all" effect. The more successful a platform is in terms of the amount of participants, the more attractive it is for new participants. Having selected a certain platform, the developer's willingness to change to another platform is very small, especially if the other mediator hosts lesser projects and is frequented by lesser users. Furthermore, a challenging mediator has few chances to attack the leading platform because no battle on prices is possible since the services are free of charge anyway. Therefore, the market entry barriers for other mediators are rather high. Additionally, the costs of operation of a full-fledged open source platform hosting thousands of projects and frequented by thousands of users seven days a week are high, whereas the opportunities to create a revenue stream are limited. It is questionable whether the cash receipts from selling banners on the Web pages exceed the operation costs. The best-known and greatest provider of such a mediator service is SourceForge.

- **Accessorizing:** Companies pursuing this business model sell accessories associated with and supportive of open source software. T-shirts printed with the name and logo of a famous open source project or a professionally edited and produced documentation of open source software are examples of these products. O'Reilly pursues this business model and is well known for their various books on open source software.

In previous years the research community studying the open source phenomenon made remarkable advances. The research yielded interesting results in topics as the open source developer's motivations, the management of open source projects and the coordination of contributors, the importance of government funding, the consequence of software patents, and others. In addition, there are more and more studies available that explore the relationship between open source software and the commercial business area. Especially the question how companies can foster and leverage innovation happening beyond the company's boundary for their business model has attracted interesting research (see Gabriel and Goldman (2002), Dahlander (2004), Bonacorsi and Rossi (2005), Chesbrough (2007), Garriga et al. (2011) or Mahajan and Clarysse (2013) for example).

Analysing the Interplay of Different Motivations

Collective action problems describe situations in which everyone in a given group is confronted with a certain set of choices. If every member of the group chooses rationally in the economic sense, the outcome will be worse than if all members are willing to choose another, individually suboptimal alternative. Open source projects face collective action problems on different levels. The software produced is a public good, and therefore, the project has to deal with free riders in an *n-person prisoner's dilemma* situation: if the project succeeds and is able to deliver the software, everybody benefits. However, everybody can improve his or her situation by not collaborating. Thus, if everybody acts rationally, the project will not succeed and, hence, the software will not be produced. As we mentioned above, this first order social dilemma can be overcome by selective incentives. However, the problem to coordinate the different contributors remains. This is a second order social dilemma because coordinating the contributors and reinforcing the social norms guiding the collaborative work is a public good, too.

According to Elster (1989), social norms play an important role in overcoming the social dilemmas and promoting collective action. Concerning the possible attitudes toward social norms, Elster identified five different positions: (1) *rational, egoistic* persons are guided by the dominant strategy of non-cooperation; (2) "*Everyday Kantians,*" by following Kant's "Categorical Imperative," are guided by a norm that Weber named "ethics of conviction" (Gesinnungsethik), thus cooperating by all means; (3) *utilitarists* cooperate conditionally, if their contribution increases the average utility; (4) *elite-cooperators* contribute in an early phase of the project when there are few participants, whereas *mass-cooperators* contribute only after many other persons decided to cooperate. Both types share the common attribute that they have a private benefit not only from the results of the cooperation, but from the act of cooperation too; and (5) persons motivated by *fairness norms* contribute as soon as the general level of cooperation exceeds a certain threshold.

With such a set of motivational types, it is possible to explain the dynamic of collective action. Everyday Kantians act as catalyst for cooperation. Persons motivated by fairness norms can amplify this process. The number of utilitarists and persons motivated by fairness norms move inversely: The more persons contribute, the less effect there is from a utilitarist's contribution. Hence, their number is decreasing. At the same moment, an increasing number of persons motivated by fairness norms are induced to cooperate. The same happens for elite- and mass-contributors. Such a dynamic model establishes that persons who are differently motivated should be found in different phases of the open source project.

We can roughly distinguish three different project phases: project initialisation, development to stability where community building also happens, and maintenance phase with stable releases. Concerning the project roles, the following differentiation is useful: the *project leader* (also known as "benevolent dictator") is usually the project founder and has the ultimate responsibility. Among others, the project founder chooses the type of open source license for the

project. Thus, the project leader normally has the right to relicense the open source project. The *committers* are contributing to the project's code base on a regular basis, the *lead users* are actively using the application, sometimes contributing bug fixes, adaptations, feature wishes, and the like, and *silent users* that use only stable releases, silently quitting as soon as the application does not meet their requirements any longer. Even though silent users do not contribute any code or information to the project, they might be important because of positive network externalities they create.
Using the motivational types we explained in the first section, we can depict the dynamics in an open source project as follows. Project initialisation is done by the project founder (we do not consider open source projects fully sponsored by companies here). The project leader may be of "everyday Kantian" type and moved by altruism. Stallman's founding of the GNU project may be interpreted in this light[2]. Project founders might also be moved by fun or by the need of the functionality. An example of this type is Torvalds and his Linux project. In this case, it's rather egoism then altruism guiding such project leaders.

In the project's next phase, the application's core functionality is created. In this phase of the project its core community has to be built while different types of committers join the project. They may be utilitarists as well as elite-contributors. Utilitarists contribute because they are interested in the result and their engagement helps the project. As elite-contributors they need a selective advantage from cooperation. This might be the fun they enjoy while developing for the project or the learning effect from contributing. At this stage, the project leader's attitude has to change gradually, at least if he or she has been moved by the fun motive originally. In order for community building to occur, he or she has to offer a credible project vision and challenging tasks for the developers joining the project. The more the project establishes, the more people join moved by fairness norms. Such persons are important to build community culture and identity. They also do the more tedious work essential to reach project stability, for example, project documentation, usability tests, quality and release management, and so on.

The "break-even" point referred to by project stability is the stage where the project has gathered enough momentum to attract reputation-motivated contributors. At this point, the prospects are good and the project will provide valid signals to outsiders in the near future. Concerning the social norms, such contributors are rather masscontributors. At the same time, however, they need to be rather competitive, for that they can capture a position within the project that allows them to stand out from the other persons involved. It is well known from famous open source projects (Linux of FreeBSD, e.g.) that the entry barriers for new contributors are very high: Project leaders only accept codes that are unobjectionable and of outstanding quality. Thus, such projects indeed provide credible signals for the outsiders and, therefore, are attractive for programmers who play the reputation game.

The ultimate proof that an open source project is both stable and successful is its inclusion into a distribution. A distributor selects an open source application only if it adds value to his distribution on the one hand and if it is easy to install on the other. The first condition implies that the distributor has enough clues that silent users demand this application. The latter condition means that the project concerns not only about coding and architecture, but about documentation and packaging, too. However, being neatly packaged and included in a distribution for the delight of silent users bears the danger of stagnation for the project. Because of that reason, the project needs lead users so that it can evolve even in its stable form. Whereas silent users only work with the stable releases of an application, lead users download and install release candidates. Thus, they are acting as beta testers and provide helpful feedback to the project if they find bugs or deficiencies.

Lead users are elite-cooperators. They have fun using the newest version of a slick tool long before others; at the same time they learn and build up valuable knowledge about the application, its evolution, and hidden goodies and limitations. An interesting point is that whereas egoistic, rational persons do not participate in usual collective action providing a public good, even such people may contribute to open source projects, as long as they can consume enough fun, for example.

Which institutional arrangements are needed to foster such a dynamic of an open source project?

As described above, the first phase of an open source project can be explained either by altruistic or egoistic programmers. The first donate their work and time for a good aim. Even if the motives behind their donations are unidentified and possibly assessed as irrational, Franck and Jungwirth (2003) argued that donators choose a beneficiary who is "worth" the donation, for example, by credibly committing himself or herself to the nondistribution constraint. Thus, while pursuing their targets, donators act rationally. In the context of an open source project, an altruistic founder who contributes the initial code base to the public wants that the code donated will be free and not privately appropriated by commercial software companies. Thus, he or she wishes that nobody can turn donations into private profits. The institutional safeguard for this is the Copyleft clause of a restrictive open source license. The Copyleft not only ensures that the code donated will be free, but that additional work building on this code has to be free, too. Therefore, a restrictive licensing scheme efficiently prevents any attempt to commercialise on any such code.

A fun-seeking project founder, on the other hand, does not bother much about licensing. Indeed, having developed the initial code base, he or she has already consumed the *homo ludens payoff* and has, therefore, little reason to release the code at all. Thus, such project owners need additional incentives to do that. This might be that releasing code is cheap because of the Internet and because of platforms offering their hosting services for free. Another reason might be the pragmatic motive that the project needs a community when it reaches maturity. Considerations about the project leader's reputation might be an additional motivation to release the code. In all cases, the existence of both mediums, an active community that can be addressed and the Internet for communication, play a crucial role.

Elite-cooperators and utilitarists joining the project now may be moved by the fun motive. They have good chances to get satisfied, because in its early stage, the project might provide the most challenging tasks and there are only few developers competing for such tasks. To be attractive for these developers, the project's entry barriers must be low. A developer who scans through project descriptions looking for a nice challenge does not want to wait days until he gets his or her contributor's access. On the other hand, the environment has to be "forgiving." If one developer delivers a code piece that breaks the application's functioning in another module, this should not break down the whole project. Instead, it must be easy to roll back the code state and go on again from there. A forgiving and responsive environment can stimulate a vibrant developer community and lead to considerable results within a short time (see Broadwell, 2005).

In the later phase of the development stage, contributors enter the project who might be motivated rather by fairness norms than by fun. Just as for altruistic project leaders, for such contributors the license type is of importance. They donate their time and creativity to produce code, documentation, and so on and they wish that these donations should not be appropriated privately. Thus, a license having a copyleft clause provides the right incentives for such contributors.

Reputation-motivated contributors entering the project in its stable stage might have an ambivalent attitude towards the license of the open source project. On the one hand, the chosen license has to guarantee the visibility of their contributions. On the other hand, a too restrictive license might impede the application's diffusion, thus reducing both the project's and the contributors' reputations. In the end, they might prefer a liberal license as long as the project owner can credibly assure that he or she never re-licenses the project to a closed scheme. At last, it is rather the existence of lead users than a license issue that determines whether reputation-motivated contributors join or not. The lead users' feedback drives the project to a considerable amount, whereas a project without lead users will stagnate and decline within a short time. Lead users on the other side are attracted by new features. Therefore, reputation-motivated contributors and lead users have a reciprocal relationship: Reputation-motivated contributors implement the new features of an application, which the lead users demand, whereas the latter provide the feedback and stimulate activity. Therefore, lead users need a low-cost access to the source code or the application's installers as well as a credible signal that the open source project in its actual form will persist. Project failure or closing its source code will devaluate the lead-users' knowledge, so that uncertainty about the project's future will detract lead users from the project. Thus, a project whose code is owned by only one person is rather unattractive, whereas a project with broad code ownership or a code being held by a foundation is more attractive for lead users.

To conclude, as long as an open source project succeeds in accomplishing heterogeneous needs, it can attract the differently motivated contributors building a vibrant community required to make the project successful. To make this possible, both the project's license and the infrastructure are issues. For altruistic contributors, the Copyleft clause is a prerequisite to attract them in the initial and the project's building phases. For fun-seeking contributors, the joy of programming depends on the infrastructure. Programming is usually more enjoyable, thereby enhancing the fun factor, if the developer can focus on coding without being distracted by organisational issues. To accomplish this positive environment where the developer can concentrate on what he or she loves to do most, the infrastructure has to be both highly available and responsive.

The remaining interesting question is, What happens to this arrangement when companies pursuing open source business models enter the scene?

We can distinguish between one scenario where the company donates the idea and vision of the application as well as its initial code base and a second scenario where the company joins an already existing project and establishes a business around the application.

In the first scenario, the company will not succeed in building up a community around the application until the project reaches stability. The project is not attractive for altruistic contributors because, even if the chosen license is restrictive, the company cannot credibly promise to let the source code open. On the other hand, the company cannot accept any code contribution from the outside without commanding the right to re-license the code. However, the company-sponsored project is of less attractiveness for the fun-motivated contributors, too. This is because such a project is driven by the in-house software developers and, thus, cannot offer challenging tasks to outside contributors. Nevertheless, having reached stability the company could try to build up a community by attracting both reputation-seekers and lead users. As we explained above, reputation-seekers might prefer projects released under a liberal license. However, implementing a dual licensing scheme might be a viable alternative. The commercial license makes the application fit for commercial use whereas the open source license with a copyleft clause guarantees the continuous openness of the source code, thus making the contributions visible. The same consideration holds for lead users.

Another strategy for companies planning to build up an open source community in the project's stable stage could be to hand over the code ownership to a foundation. Even this ensures the source code's continuous openness, thus making the project attractive for rent-seekers and lead users.

A company will join an already existing open source project most probably in its stable stage when the project has already attracted a community. For playing a role within the project, the company has to be accepted by the community and also be very careful not to scare away the community members. Such a strategy can be successful only if the company is very open and transparent about its intentions concerning the project participation. The company has to communicate clearly how it wants to earn money by promoting the project and at the same time why the openness of the code is vital for the company's business model. In addition, the company has to be careful that the project stays attractive for fun seeking contributors. This can be achieved if the company does not "in-house" the application's further development but lets the community develop and implement the application's enhancements.

The Pursuit of Software Quality

Software quality can be considered as consisting of code quality (e.g., testability, simplicity, readability, self-descriptiveness) and software usability (e.g., ease of learning, efficiency of use, error frequency and severity, subjective satisfaction). According to actual surveys, open source software has caught up to proprietary software concerning code quality (Coverty Scan Report, 2013). Meanwhile, companies rank the quality as most important factor to adopt open source software (Skok, 2013). When it comes to usability, however, open source software still suffer problems (Nicholas and Twidale (2002), My Personal Thoughts (2012)).How can the open source software development model achieve code quality and how could it improve on usability?

Pragmatic contributors want to see the improvements they added in an operational state and, thus, don't bother much about code quality. Fun seekers may enjoy "elegant," manageable code that is both simple and readable. However, the quality of their contributions depends largely on their ability and experience. If they are unskilled, they may experience fun producing "ugly," unwieldy code as well. Therefore, it's mainly the reputation-motivated contributors that drive open source projects' code quality because their reputation builds heavily on the quality of their contributions. In addition, their intensive interaction with lead users that act as beta testers enables them especially well for the pursuit of code quality.

Why is this same reasoning not true for usability? Lead users are computer experts no less then the contributors to the open source project. They don’t need elaborate user-interfaces to fully exploit the functionality offered by the software. Therefore, they can’t feed back usability issues to the project. Those who are best suited to evaluate the application’s usability are the silent users. However, as they are silent, they don’t give any feedback. Is there a way to capture the silent user’s experience? Here’s the place where companies entering the open source area while pursuing a business model can add value to the open source project. At least if they have a history in the software business, they are in connection with the application’s potential and silent users and, therefore, can act as a proxy for them and provide valuable feedback concerning usability issues.

These reflections show that every stakeholder can provide valuable input to the success of an open source project. Crowding out between contributors with different motivations does not necessarily exist even if companies with monetary intentions participate. Therefore, we assume that open source development is not a temporary but rather a stable phenomenon because its particular production context allows every participant to put in and to put out as much (or as little) as he or she wants. Therefore, it is an attractive forum for different interests that can seminally intertwine, while quality software is generated nearly as a by-product.

References

Arjona Reina, L. & Robles, G. & González-Barahona, J.M. (2013). *A Preliminary Analysis of Localization in Free Software: How Translations Are Performed*. In E. Petrinja& G. Succi & N. Ioini & A. Sillitti (Eds.), *Open Source Software: Quality Verification* (pp. 153-167). Heidelberg: Springer.

Bonacorsi, A., & Rossi, C. (2005). Intrinsic motivations and profit-oriented firms in open source software. Do firms practice what they preach? In M. Scotto & G. Succi (Eds.), *Proceedings of the 1st International Conference on Open Source Systems*, Genova (pp. 241-245).

Broadwell, G. (2005). *-Ofun*. Retrieved October 10, 2005, from http://www.oreillynet.com/pub/wlg/7996.

Chesbrough, Henry W. (2007). *Why Companies Should Have Open Business Models*, Retrieved July 28, 2013, from http://sloanreview.mit.edu/article/why-companies-should-have-open-business-models/.

Csikszentmihalyi, M. (1975). *Beyond boredom and anxiety*. San Francisco: JosseyBass.

Csikszentmihalyi, M., & Csikszentmihalyi, I. S. (1988). *Optimal experience: Psychological studies of flow in consciousness*. Cambridge, UK: Cambridge University Press.

Coverity (2013). *Coverity Scan: 2012 Open Source Report*, Retrieved July 28, 2013, from http://softwareintegrity.coverity.com/register-for-the-coverity-2012-scan-report.html.

Dahlander, L. (2004). *Appropriating returns from open innovation processes: A multiple case study of small firms in open source software*. Retrieved April 10, 2006, from http://opensource.mit.edu/papers/dahlander.pdf

Elster, J. (1989). *The cement of society: A study of social order*. Cambridge, UK: Cambridge University Press.

Franck, E., & Jungwirth, C. (2003). Reconciling rent-seekers and donators. *Journal of Management and Governance*, *7*, 401-421.

Franck, E., Jungwirth, C., & Luthiger, B. (2005). *Motivation und engagement beim OSS-programmieren—Eine empirische analyse*. Retrieved July 18, 2005, from http://www.isu.unizh.ch/fuehrung/Dokumente/WorkingPaper/36full.pdf

Gabriel, R. P., & Goldman, R. (2002). *Open source: Beyond the fairytales*. Retrieved

October 4, 2003, from http://opensource.mit.edu/papers/gabrielgoldman.pdf

Garriga, H. & Spaeth, S. & von Krogh, G. (2011). *Open Source Software Development: Communities Impact on Public Good*." In *Social Computing, Behavioral-Cultural Modeling & Prediction, Lecture Notes in Computer Science* (Vol. 6589, pp 69-77). Heidelberg: Springer.

Ghosh, R. A. (2003). *Copyleft and dual licensing for publicly funded software development*. Retrieved July 6, 2004, from http://www.infonomics.nl/FLOSS/ papers/dual.htm

Goldman, Ron & Richard P. Gabriel (2005). *Innovation Happens Elsewhere. Open Source as Business Strategy*. San Francisco: Morgan Kaufmann.

Hann, I-H., Roberts, J., Slaughter, S., & Fielding, R. (2004). *An empirical analysis of economic returns to open source participation*. Retrieved July 12, 2006, from http://www.andrew.cmu.edu/user/jroberts/Paper1.pdf

Hannemann, A. & Klamma, R. (2013). *Community Dynamics in Open Source Software Projects: Aging and Social Reshaping.* In E. Petrinja& G. Succi & N. Ioini & A. Sillitti (Eds.), *Open Source Software: Quality Verification* (pp. 80-96). Heidelberg: Springer.

Hars, A., & Ou, W. (2001). Working for free?—Motivations of participating in open source projects. In *34th Annual Hawaii International Conference on System Sciences* (Vol. 7, pp. 1-7).

Haruvy, E., Prasad, A., & Sethi, S. P. (2003). Harvesting altruism in open-source software development. *Journal of Optimization Theory and Applications, 118*(2), 381-416.

Hecker, F. (1999). Setting up shop: The business of open-source software. *IEEE Software*, *16*(1), 45-51.

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers. *Research Policy, 32*(7), 1159-1177.

Kalla, R. (2006). *Eclipse as an ecosystem*. Retrieved March 3, 2006, from http:// www.eclipsezone.com/eclipse/forums/t64080.rhtml

Krishnamurthy, S. (2002). Cave or community? An empirical examination of 100 mature open source projects. *FirstMonday, 6*. Retrieved June 6, 2002, from http://www.firstmonday.org/issues/issue7_6/krishnamurthy/index.html

Lakhani, K. R., & Wolf, R. G. (2003). *Why hackers do what they do: Understanding motivation effort in free/open source software projects*. Retrieved October 6,
2003, from http://opensource.mit.edu/papers/lakhaniwolf.pdf

Leiteritz, R. (2004). Open source-geschäftsmodelle. In R. A. Gehring & B. Lutterbeck (Eds.), *Open source jahrbuch 2004* (pp. 139-170). Berlin: Lehmanns Media.

Lerner, J., & Tirole, J. (2002). Some simple economics of open source. *Journal of Industrial Economics*, *52*(6), 197-234.

Luthiger, B. (2006). *Spass und software-entwicklung. Zur motivation von opensource-programmierern*. Retrieved July 13, 2006, from http://www.dissertationen.unizh.ch/2006/luthigerstoll/diss.pdf

Mahajan, A. & Clarysse, B. (2013) *Technological Innovation and Resource Bricolage in Firms: The Role of Open Source Software.* In E. Petrinja& G. Succi & N. Ioini & A. Sillitti (Eds.), *Open Source Software: Quality Verification* (pp. 1-17). Heidelberg: Springer.

My Personal Thoughts (2012). *Why free software has poor usability, and how to improve it*. Retrieved August 1, 2013, from http://www.mpt.net.nz/2012/06/why-free-software-has-poor-usability/

Nichols, D. M., & Twidale, M. B. (2002). *Usability and open source software*. Retrieved October 4, 2003, from http://www.cs.waikato.ac.nz/~daven/docs/oss-fm.pdf

O'Mahony, S., Cela Diaz, F., & Mamas, E. (2005). *IBM and Eclipse*. Harvard: Harvard Business School.

Osterloh, M., Rota, S., & Kuster, B. (2002). *Open source software production:*

Climbing on the shoulders of giants. Retrieved July 21, 2003, from http://www. iou.unizh.ch/orga/downloads/publikationen/osterlohrotakuster.pdf

Raymond, E. S. (1999). *The magic cauldron*. Retrieved July 21, 2003, from http:// www.catb.org/~esr/writings/magic-cauldron/

Raymond, E. S. (2000). *Homesteading the noosphere*. Retrieved July 21, 2003, from http://www.catb.org/~esr/writings/homesteading/

Riehle, Dirk (2012). *The Single-Vendor Commercial Open Source Business Model.* In J. Becker & M.J. Shaw (Eds.), *Information Systems and e-Business Management* (Vol. 10, Issue 1, pp. 5-17). Heidelberg: Springer.

Stamelos, I., Angelis, L., Oikonomou, A., & Bleris, G. L. (2002). Code quality analysis in open source software development. *Information Systems Journal*, *12*(1), 43-60.

Skok, Michael J. (2013). *2013 Future of Open Source. 7th Annual Survey results*, Retrieved July 28, 2013, from http://www.slideshare.net/mjskok/2013-future-of-open-source-7th-annual-survey-results.

Stürmer, M. & Spaeth, S. & von Krogh, G. (2009). *Extending private-collective Innovation: a Case Study*. In *R&D Management* (Vol. 39, Issue 2, pp. 170-191). Wiley.

Toffler, A. (1980). *The third wave*. New York: Bantam Books.

Torvalds, L., & Diamond, D. (2001). *Just for FUN: The story of an accidental revolutionary*. New York: Harper Collins Publishers.

von Hippel, E., & von Krogh, G. (2002). *Exploring the open source software phenomenon: Issues for organization science*. Retrieved July 21, 2003, from http://opensource.mit.edu/papers/removehippelkrogh.pdf

Weber, S. (2000). *The political economy of open source software*. Retrieved March 6, 2002, from http://brie.berkeley.edu/~briewww/pubs/wp/wp140.pdf

Weber, S. (2004). *The success of open source*. Cambridge, MA: Harvard University Press.

Weiss, D. (2005). Measuring success of open source projects using Web search engines. In M. Scotto & G. Succi (Eds.), *Proceedings of the 1st International Conference on Open Source Systems*, Genova (pp. 139-170).

Endnotes

1) In a study about "Fun and Software Development" (FASD), we explored the importance of fun that the open source developers enjoy on their open source engagements. The survey addressed both open source and commercial developers and was filled out by 1330 programmers from the open source area and by 114 developers working in six Swiss software companies. The surveys have been open during about two months in early summer and autumn 2004 respectively.

2) "I'm looking for people for whom knowing they are helping humanity is as important as money" (Richard Stallman on www.gnu.org/gnu/initial-announcement.html).