Norwegian University of Science and Technology
Department of Industrial Economics and Technology Management

Peculiarities of the Commercial Open Source business model: Case study of SugarCRM

Master Thesis

Simon R. B. Berdal

January 2013

NTNU
Department of Industrial Economics and Technology Management

Til Gumman

Preface and acknowledgements

This thesis is submitted in partial fulfillment for the Master of Technology Management (MTM) degree, administered by the Department of Industrial Economics and Technology Management (IØT) at the Norwegian University of Science and Technology (NTNU). The MTM program is based on close collaboration between IØT and two departments at the Norwegian School of Economics (NHH): Strategy and Management, and Business Economics. Also, this degree includes the fulfillment of a visiting fellowship program at Sloan School of Management, Massachusetts Institute of Technology (MIT).

I want to thank my supervisors, Arve Pettersen and Benjamin Mako Hill, for their advice and helpful comments on the text. Likewise, I feel obliged to John Mertic, Community Manager at SugarCRM, for providing valuable information (interviews and data sets) and facilitating an online questionnaire. Last but not least, I must apologize to all victims of my absent-mindedness while being engrossed in this thesis. Sorry for that.

Oslo, January 2013

Simon R. B. Berdal

Abstract

This thesis investigates the so-called “single-vendor” commercial open source business model, as exemplified by SugarCRM. Within the conceptual framework of Actor-Network Theory, the case study is informed by three interrelated research questions. The first relates to the different stakeholders and their incentives for participation. The second concerns the strategies applied by the owning entity to meet these stakeholders’ expectations and needs, whilst at the same time safeguarding its own shareholders’ interests. Following, the last takes on friction points that could undermine the viability of SugarCRM’s business model, and corresponding strategies by the firm to avoid them.

Keywords

Open Source, Commercial Open Source, community, business models, multi-licensing, incentives, collaboration.

Citing

Berdal, S. R. B. (2012) *Peculiarities of the Commercial Open Source business model: Case study of SugarCRM*. Master Thesis, Department of Industrial Economics and Technology Management, NTNU

Contents

1 Introduction

The last decade has witnessed overwhelming adoption- and adaption rates of open source software (OSS[1]). From being a hacker's exclusive domain of endeavor, it has rapidly evolved into a wider ecosystem of aligning interests. Accordingly, OSS has been heralded as a "game changing" force – a constituent of the contemporary socio-technological Zeitgeist. The recent developments, coupled with the catalyzing mediatory role of the Internet, have raised a multitude of questions among scholars and business leaders alike. How, for instance, can market players effectively utilize OSS as a strategic asset? And what are the implications for the competitive landscape?

1.1 Background and motivation

The players of the software industry have approached OSS from a variety of angles. Some outright attacked it, as if it were a cancer to their intellectual property rights. Others wholeheartedly embraced it in an attempt to harness its powers. Software companies are said to draw a range of benefits by adapting an open source business model, such as:

- Reduced costs for development, support and marketing
- Efficient quality control
- Demand-driven innovation and –adaptation
- Short time to market.

These benefits build on two intrinsic features of OSS projects: A pool of voluntary co-developers and testers (community), and a distinctly lean development model[2]. Thus, to get it right, open source software projects need to fulfill at least two essential preconditions:

1. Attracting a sizable and healthy community
2. Employing the community effectively

In a commercial context, meeting these requirements is no trivial task. Pure OSS projects are driven by community members' need and enjoyment of developing and using a given

[1] With OSS, I refer not only to the software itself, but also related development methodologies and licensing schemes. See Chapter 2.3 for definition of OSS.
[2] The "Cathedral" model, coined by Eric S. Raymond (1997). See chapter 2.3.5 and Appendix B.

software[3]. Commercial variants (COSS), on the other hand, must also adhere to the interests of shareholders[4]. This necessitates one further precondition:

3. Balancing the interests of commercial stakeholders and the open source community.

In other words, conventional OSS projects' *raison d'être* is to satisfy the wants of their community, while COSS project's ultimate goal is to increase shareholders' value. The challenge is to reconcile these deviating interests into long-lasting symbiotic relationships.

Against this backdrop, my thesis investigates the commercial application of OSS, with a main focus on the incentives, character and interplay of involved stakeholders. My interest in this subject was initially triggered by the extensive use of a COSS application (SugarCRM, Community Edition) by my former employer. The quality and rapidly growing market share of this product compelled me to take a closer look, considering the prospect that it could inspire entrepreneurial emulation of the underlying business model.

It is worth noting that this problem domain probably expands beyond the boundaries of the software industry. Mimicking the success of OSS, we have lately witnessed the open source model being applied to everything from computer hardware and architecture, to cars, medical devices and design. Even though I have no ambitions to extrapolate my findings to other industries, I suggest they should be considered to be of relevance to all commercial open source adaptations.

[3] The motivations for participation in OSS communities are fully discussed in chapter 2.3.1.
[4] Which also implicates the interests of paying customers.

1.2 Problem definition and structure of the thesis

The objective of this thesis is to explore and analyze the viability of the "commercial open source" business model, as exemplified by my object of inquiry – SugarCRM[5]. Strictly defined, this case study aims to shed light on the following set of research questions:

1. Stakeholder motivation: Who are the involved parties in the software development process, and what are their primary incentives for participation?

2. Strategies: Which strategies do the commercial entity employ to satisfy the stakeholders' expectations and needs, whilst at the same time safeguarding shareholder interests?

3. Friction points: Following from the previous – are there notable friction points between involved stakeholders, which could undermine the viability of SugarCRM's business model? If yes, how does the firm handle them?

In my pursuit for relevant answers, I take on an exploratory grounded theory approach (see Chapter 3). This methodological approach is well reflected in the successively narrowing focus of the thesis' narrative – from general theoretical foundations of open source and its commercial incarnation (Chapter 2), to an analysis of SugarCRM's distinguishing particularities (Chapter 4). For readers unfamiliar with the concepts of open source, I also include a literature review of perhaps the most defining writing on the phenomenon (see Appendix B).

[5] SugarCRM is both the name of the company, and its software solution (prominently labeled as "commercial open source CRM").

2 Theoretical Foundations

In this chapter, I introduce and discuss the concepts that provide the theoretical foundations for the later analysis. It opens with a prologue to set a historical scene (2.1), followed by an introduction of essential Actor-Network Theory (ANT) terminology to prepare for its subsequent use (2.2). This lays the ground for a closer study of the multifaceted phenomenon of open source software (2.3), including a systematic dissection along its socio-technological, organizational and legal dimensions. After a brief look at generic open source business models (2.3.7), we are then in the position to turn our attention to the specific variant that is relevant to the object of inquiry (2.4).

2.1 Prologue: The renaissance of open source software

"In the beginning, there were Real Programmers"

This is how Eric S. Raymond (1999a) boldly kicks off his historical account of software "hackerdom". With "Real Programmers", he refers to members of the "self-conscious technical culture of enthusiast programmers, people who built and played with software for fun" (ibid.). These were scientists or amateur-radio hobbyists from the early days of computing who shared their ideas openly through various communities. Thus, open revealing and distribution of code began as the default *modus operandi* of software development, albeit without any defined naming label or alternative. But with the rise of proprietary software in the late 70s and early 80s (e.g. AT&T's UNIX, Microsoft DOS and Windows) the hacker movement came under increasing pressure. Software manufacturers stopped distributing their source code and restrictive licenses prohibited copying and redistribution. These developments prompted an aggravated Harvard graduate and MIT[6] employee, Richard Stallman, to announce the following in 1983[7]:

[6] MIT Artificial Intelligence Laboratory

[7] This is the extended version of the original announcement, as incorporated into *The GNU Manifesto*. Retrieved October 8, 2012 from http://www.gnu.org/gnu/manifesto.html. Note: GNU was to be a free (as in liberty) replacement of the proprietary Unix operating system. The name stands for "Gnu's not Unix".

I consider that the Golden Rule requires that if I like a program I must share it with other people who like it. Software sellers want to divide the users and conquer them, making each user agree not to share with others. I refuse to break solidarity with other users in this way. I cannot in good conscience sign a nondisclosure agreement or a software license agreement. For years I worked within the Artificial Intelligence Lab to resist such tendencies and other inhospitalities, but eventually they had gone too far: I could not remain in an institution where such things are done for me against my will. So that I can continue to use computers without dishonor, I have decided to put together a sufficient body of free software so that I will be able to get along without any software that is not free. I have resigned from the AI Lab to deny MIT any legal excuse to prevent me from giving GNU away.

The GNU project heralded the start of the free software movement, and in 1985 Stallman established the Free Software Foundation (FSF). Ever since, the FSF has been a fervent promoter of *free software*, which concedes its users the rights (or "freedoms") to "run, copy, distribute, study, change and improve the software"[8]. To that end, the FSF devised novel licensing schemes (see chapter 2.3.6) and continued to push the GNU project onwards.

In 1997, Eric Raymond published his famous inquiry into an emerging open software engineering methodology, based on lessons drawn from the development of Linux[9] and his own *fetchmail* project (see chapter 2.5.3 and Appendix B). He likened the conventional software development process to that of a cathedral (monolithic, slow, top-down) and contrasted it to the model of a bazaar (open, distributed, agile and bottom-up). By pointing out the shortcomings of the Cathedral model, and confronting them with exemplified benefits of the Bazaar model, he conveyed a convincing case for others to adopt the latter[10]. However, the model is inherently linked with open revealing of source code, a concept associated with an obscure hacker culture that antagonized commercial interests. Raymond and others therefore deemed the ideological underpinnings and confrontational attitude of the FSF as detrimental (Perens, 1999). As Raymond put it in an interview with the Salon magazine[11]:

[8] From The Free Software Definition (FSD), http://www.gnu.org/philosophy/free-sw.html
[9] Licensed under the GPLv2 (GNU General Public License, version 2), which was authored by the FSF.
[10] Raymond has been widely credited (e.g. Charles, 1999) as a major inspiration for the alteration of Netscape Navigator into the open sourced breed of Mozilla Firefox.
[11] Leonard, A. (1998-03-30), *Let my software go!* Interview with Eric Raymond. Retrieved October 8, 2012 from http://www.salon.com/1998/03/30/feature947788266/

The name "free software" has to go. The problem is nobody knows what "free" means, and to the extent that they do think they know, it's tied in with a whole bunch of ideology and that crazy guy from Boston, Richard Stallman. […] I love Richard dearly [..], but in the battle we are fighting now, ideology is just a handicap. We need to be making arguments based on economics and development processes and expected return. We do not need to behave like Communards pumping our fists on the barricades. This is a losing strategy. So [..] we needed a new label, and we brainstormed a bunch of them and the one that we finally came up with is "open source."

The open source movement was born, and, in early 1998, Raymond and Bruce Perens established the Open Source Initiative (OSI). In spite of their differences, both camps promote many of the same licensing schemes. But while the FSF still maintains that software should be "free" on ethical grounds, the OSI represents a more pragmatic view that sees open source as a means to an end[12]:

Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in.

Since the turn of the millennium, OSS has gradually evolved into a mainstream phenomenon and altered the basic nature of the software industry (Fitzgerald, 2006). As it grew in popularity, it also attracted increasing amounts of attention and opposition from closed source incumbents. In 2001 Microsoft's CEO, in an attempt to discredit the movement, attacked the company's major OSS rival (Linux), describing it as "cancer that attaches itself in an intellectual property sense to everything it touches"[13]. Meanwhile, new enterprises have thrived within the new OSS paradigm (e.g. Red Hat), and many established behemoths have endorsed, supported and / or utilized it for own benefit (e.g. Novell, IBM and Sun). When IBM's Watson won the 2011 TV game show *Jeopardy*, it was much thanks to Linux on which it ran. Apparently, IBM treats OSS know-how as a strategic asset (Capek et.al, 2005). Already in 1999, it established the IBM Linux Technology Center, which cooperates with the open source community and "serves as a center of technical competency for Linux"[14]. In the same spirit, it started several OSS initiatives by itself and adopted OSS licensing schemes for

[12] From the OSI's mission statement, Retrieved October 8, 2012 from http://opensource.org/
[13] In *Microsoft CEO takes launch break with the Sun-Times*, Chicago Sun-Times, June 1, 2001; archived from the original on December 11, 2001; retrieved November 6, 2011 from http://web.archive.org/web/20011211130654/http://www.suntimes.com/output/tech/cst-fin-micro01.html
[14] Retrieved November 14, 2011 from the LTC web page, http://www-03.ibm.com/linux/ltc/

existing software. In 2005, IBM also made a significant "non-assertion" pledge to let OSS developers use 500 of its patents without fear of infringement[15]. Even Microsoft seems to have softened its stance lately. In 2006 it launched CodePlex, an open source project site[16], and soon got two of its own licenses approved by the OSI[17]. Such Microsoft-led initiatives have lit the blogosphere ablaze with speculation. Some construe them as a tactical smoke screen, others as a sincere strategic realignment – "*If you can't beat them, join them*", so the logic goes. Whatever the underlying rationale, it underscores the focal role that OSS has gained in the software industry: You have to relate to it somehow.

[15] "*IBM Statement of Non-Assertion of Named Patents Against OSS*", retrieved November 14, 2011 from http://www.ibm.com/ibm/licensing/patents/pledgedpatents.pdf

[16] Which mainly hosts OSS projects relating to Microsoft technologies (e.g. .NET framework).

[17] Namely the *Microsoft Public License* (Ms-PL) and *Microsoft Reciprocal License* (Ms-RL).

2.2 Actor-Network Theory terminology

For the analysis and description of socio-technical ensembles, I (as others, e.g. Tuomi, 2001; Tatnall 2003) find it valuable to borrow key assumptions and terminology from Actor-Network Theory (ANT). I have no ambitions to provide a full introduction to ANT here, but rather explain its relevant concepts and how they are applied throughout this thesis (rephrasing Berdal 2004).

Two ANT concepts are of particular relevance to describe and analyze characteristics of the open source development process and governance models: *inscription* (Akrich 1992; Akrich and Latour 1992) and *translation* (Callon 1991; Latour 1987). Inscription refers to the way technical artefacts embody patterns of use: "Technical objects thus simultaneously embody and measure a set of relations between heterogeneous elements" (Akrich 1992: 205, in Hanseth and Monteiro 1998). As Hanseth and Moneiro further explain:

> *The term inscription might sound somewhat deterministic by suggesting that action is inscribed, grafted or hard wired into an artefact. This, however, is a misinterpretation. Balancing the tight rope between, on the one hand, an objectivistic stance where artefacts determine the use and, on the other hand, a subjectivistic stance holding that an artefact is always interpreted and appropriated flexibly, the notion of inscription may be used to describe how concrete anticipations and restrictions of future patterns of use are involved in the development and use of technology.*

Translation is the process of continuous negotiations within a network of inter-related actors (that is, an actor-network), where aligning interests result in manifestations of "...ordering effects such as devices, agents, institutions, or oraganizations" (Law 1992: 366, in Hanseth and Monteiro 1998). The input of the translation (such as anticipations or interests) is thereby converted into another state, for which the resulting inscribed entity (which may be anything from a work practice or software to legal framework) is to serve as its agent. The inscription includes programs of action for the end-users, by defining roles to be played by both users and the inscribed entity. The success of the translation depends on the strength of the inscription, or sum of inscriptions, and the resistance, or anti-programs, towards the inscribed patterns of use. A classic example is that of a hotel manager who attaches weights to keys to persuade guests to leave their keys at the reception desk on leaving (Latour 1991). Thereby, the inscribing actor is making "assumptions about what competencies are required by the users as well as the system (Latour 1991). In ANT terminology, she [the actor]

delegates roles and competencies to the components of the socio-technical network, including users as well as the components of the system. By inscribing programs of action into a piece of technology, the technology becomes an actor [or *actant*[18]], imposing its inscribed program of action on its users" (Hanseth and Monteiro 1998).

In the case of this thesis' problem domain, the concepts of inscription and translation are highly relevant to understand and depict the characteristics that influence the modus operandi of open source innovation and stakeholder interaction. It is, in this context, important to understand the complexity found in most actor-networks, such as open source projects. The number of involved actants is immense, and the different cause-effect relations can be based on everything from rational intentions, aligning power-structures and compromises, to random accidents and human emotions. Thus, the programs of use within a given OSS project may draw their roots from many dispersed actants, of which only some are readily available or even traceable. As Monteiro (1999, p.244) writes:

> *To make sense of such a complex context, it is absolutely essential to simplify, that is, collapse complexity by zooming out, by treating comprehensive actor-networks as simple actants. Hence, we talk about the interests of whole organizations, governmental agencies etc. even though it is clear that this is but a short-hand.*

Accordingly, I find it useful to talk about the *project maintainer**[19] of an OSS project as a single entity, while it in fact may be constituted by a number of people, who follow directives and use technical enhancers to employ their will. The level of "zooming out" is naturally given by the extent of information I have at hand, and the amount of complexity that seems reasonable to handle.

[18] As the actors in a given network can be both human and non–human, *actant* is used as a general term.
[19] Terms assigned with an asterisk (*) are explained in Appendix A.

2.3 Open Source Software

Although the term "open source" originates from the software domain, it has since gained a much more generic meaning and appliance. One may sensibly ask, then, what *is* "open source"? A development methodology? A culture or movement? A legal framework? All these definientia seem plausible, yet none is sufficient in its own right. In this chapter I convey my understanding of this multifaceted term and clarify its use for the remainder of this thesis.

In one sense, "open source" is ostensibly self-explanatory – it refers to source that is open. But, as the reader by no means is required to be a computer scientist, a short clarification is warranted. Proprietary software is typically distributed as binary code. That is, code which the operating system can run, but is incomprehensible to human eyes (hence, often denoted as "closed code"). An analogy is a newly purchased car, along which the buyer receives some rudimentary documentation. If the owner wanted to make extensive adjustments to the vehicle, the provided documentation is unlikely to suffice. For that, one would need the car's blueprints. Correspondingly, to alter software effectively, one needs access to its source code[20]. In the most rudimentary sense of the term "open source" thus refers to readily available access of blueprints[21] of any type of artefact.

A wide body of literature implies at least three dimensions to OSS (e.g. Raymond, 1997; Lerner and Tirole, 2001; Bonaccorsi and Rossi, 2003; Lakhani and von Hippel, 2003; Lakhani and Wolf, 2005; Henkel, 2006):

1. Socio-technological foundation: A shared code repository, along with a set of evolved best practices and compatible norms, serves as the basis for collaboration. The Internet enables effective communication and coordination, while readily available software tools boost the development process. A wide range of motivational factors attract members to OSS communities and encourage participation.

[20] Which is in a human-understandable programming language, and (unless in an interpreted programming language, such as PHP or JavaScript) needs to be compiled in order to be executed.

[21] In most fields where the term "open source" is applied, there exists a set of formal criteria to which one must adhere in order to use an officially approved label. The OSI operates with a clean-cut "Open Source Definition" (OSD) , which comprises ten such criteria. See chapter 2.3.6, under "FOSS licenses".

2. Organization: The distributed and collaborative "Bazaar" development model (Raymond, 1997) promotes rapid evolution of the code, bug detection and fixing. Through self-selection of tasks among community members, complexity is broken down and handled effectively. To guide this process, OSS projects adopt different forms of governance models.

3. Legal framework: Licenses guarantee against copyright ownership fragmentation and discriminatory access to the code base. Depending on the normative underpinnings of the license, various legal terms further determine how the code can be applied and reused.

These interconnected dimensions demonstrate the intricacy of OSS projects' inner workings. Although several attempts have been made to suggest successful models for OSS projects (e.g. Fogel 2005), it seems overly simplistic to prescribe generic templates that work under all conditions. For that, the internal and contextual parameters are too numerous (Mikkonen and Vadén, 2009). Variables such as community size and composition, license category and governance type are linked through a complex equation. An alteration of one variable may trigger changes in others. To elucidate this intricate interdependence, I will in the following subchapters examine involved variables and seek to establish their mutual relevance.

2.3.1 Motivations and cultural Norms

A 2009 study of Debian[22] 5.0, a popular GNU/Linux distribution, estimated that it consisted of close to 324 million lines of source code and would have cost around €6.1 billion to develop from scratch using a classical, proprietary development model (Amor et al, 2009). As a product of free- and open source collaboration, however, the principal costs involved were time and effort (i.e. individual opportunity costs), voluntarily donated by members of various communities. Thus, as Lerner and Tirole already inquired in 2002 (p.198):

> *Why would thousands of top-notch software developers contribute for free to the creation of a public good?*

Numerous academics have sought to find answers to this question (e.g. Harhoff, Henkel, et al., 2003; von Hippel and Von Krogh, 2006). A recent paper (von Krogh et al., 2012) reviewed and summarized findings from relevant literature, and clustered ten "motivational categories" according to an intrinsic / extrinsic taxonomy[23]:

Intrinsic		
1	Ideology	Normative, e.g. FSF's proclaimed user freedoms or similar personal stance
2	Altruism	Selfless concern for the welfare of others without motive of own gain
3	Kinship amity	Based on sense of kinship and belonging to movement / group, e.g. community
4	Enjoyment and fun	As hobby, passion for experimentation or enjoyment of hacking
Internalized Extrinsic		
5	Reputation	"Peer reputation" (insiders) and "outside reputation" (general prestige)
6	Reciprocity	"Gift economy", i.e. giving in anticipation of likely reward in future
7	Learning	Acquire new skills and knowledge on inner workings of particular software
8	Own-use value	Needs for given set of (improved) functionality
Extrinsic		
9	Career	Increase value on labor market by signaling talent and motivation
10	Pay	Monetary reward (in context of employment).

Table 1: Condensation and interpretation of individual motivation for OSS participation, as summarized from literature in Appendix B by von Krogh, Haefliger, Spaeth and Wallin, 2012.

According to the study, these individual motivations "rooted in people's search for immediate outcomes" are important, but seem insufficient to explain other central issues (ibid., pp.656). Thus, it seeks to complement this "self-determination view" with a "social

[22] The Debian project is known for its adherence to both the FSF's ideals on user freedoms and the "bazaar style" development methodology that is advocated by the OSI. The Debian distribution consists of an integrated stack of components originating from various *FOSS** projects.

[23] The term "internalized extrinsic motivation" refers to motivation which the authors by definition ascribe as extrinsic, but can be internalized – i.e. "perceived as self-regulating behavior rather than external impositions" (ibid. p. 653).

practice view" to explain "what moves OSS developers to (1) produce high-quality work when they do, (2) engage in institutional change, or (3) sustain OSS development" (loc. cit.). At this point, the authors take on a more "holistic" and multi-disciplinary approach, building on the social and moral philosophy of Alasdair MacIntyre (thus invoking terms as virtue and "good life"). Although this theoretical choice may seem somewhat eccentric, the authors preempt skepticism through an exegesis and informed, selective application of the theory. As such, their reasoning has merit: "Individuals are [also] motivated because, through participatory exposure to social practices, they learn what it makes sense to do; and, vice versa, by reflecting on the *unity of life* they shape social practices" (original emphasis, ibid. p. 663-4). By unity of life, the authors take to mean "how the individual subjectively holds outcomes and actions to be consistent over time". This is based on the assumption that individuals "want to reach or maintain consistency of action throughout their life—an ambition that values personal development and contextual events and points beyond the attainment of specific and immediate rewards." From this "narrative of life" perspective "OSS development embedded in a social practice becomes meaningful and a way of life with its own pleasure, challenges, and other benefits." Following, the paper provides a new "motivation–practice framework" with a series of theoretical conjectures and propositions to answer the above posed questions (summarized in Appendix C). The propositions seem convincing, as are the suggested implications to OSS projects. Because OSS projects essentially aim to be arenas where heterogeneous motivations of individuals and institutions are played out in concert, it is important that resulting actions are in tune. This suggests that motivations by themselves need to be compatible to a certain extent, also taking into account individuals' value-informed, long-term pursuits. Hence, in ANT-terminology: Motivational factors constitute the underlying driving force in the socio-technological ensembles of OSS projects – either through actors' direct action, their indirect translation of will into other forms (e.g. by influencing a shared governance model) or through resulting inscribed imperatives (e.g. a chosen license and established rules for submission).

As touched upon in the previous chapter, there are different normative camps and factions in the intertwined free- and open source movement. The Free Software Foundation's (FSF) stance is that the software user by moral imperative should have access to the underlying source code in order to exercise underlying "freedoms" to study, change and redistribute it to others. The Open Source Initiative (OSI) also promotes open source code, but of a more instrumental and pragmatic reason – to allow software to effectively evolve through

collaborative, distributed innovation (see chapter 2.3.5 and Appendix B). Such normative variations echo a multitude of attitudes and cultures in the wider free- and open source community. The reader should note, however, that such differences are compatible to a large extent. The popular Ubuntu operating system[24], for instance, successfully amalgamates arguments from both camps in its proclaimed philosophy[25]. Accordingly, it has become increasingly popular to use the more inclusive abbreviation of "FOSS"[26] when speaking of both free- and open-source software. As the FSF puts it[27]:

> *The term "open source" software is used by some people to mean more or less the same category as free software. It is not exactly the same class of software: they accept some licenses that we consider too restrictive, and there are free software licenses they have not accepted. However, the differences in extension of the category are small: nearly all free software is open source, and nearly all open source software is free. We prefer the term "free software" because it refers to freedom—something that the term "open source" does not do.*

Reconciliatory tendencies and overlapping categories notwithstanding, even subtle cultural differences may matter (Stewart and Gosain, 2006). This is because culture to a large extent defines the identity and purpose of software projects and their communities. In this context, an OSS project's mission statement, governance model and legal framework can be interpreted as an expressive manifestation of communal identity and purpose – a differentiating cultural proclamation in formal jargon, as it were[28].

[24] Based on the Debian GNU / Linux distribution

[25] *About Ubuntu – our philosophy*. Retrieved may 14, 2012 from http://www.ubuntu.com/project/about-ubuntu/our-philosophy

[26] Commonly used alternatives are "F/OSS", and "FLOSS" (L for *libre*).

[27] *Categories of free and nonfree software*, Retrieved may 17, 2012 from http://www.gnu.org/philosophy/categories.html

[28] To clarify, the legal framework is here understood to go beyond the choosing of a license, it may involve differentiating additional terms and varying practices of implementation.

2.3.2 The Community

The driving force behind any OSS project is a community of volunteering developers and users with a shared interest in the making, improvement and support for a given software. This chapter examines essential features of communities, and discusses relevant aspects concerning the building and preservation of them.

Unless the product of a *fork**, OSS projects typically start off without any community. During an incubation period, the initiator (*project maintainer**) must build a sufficiently appealing program to spur interest of potential users and developers – what Raymond calls a "plausible promise"[29]:

> *When you start community-building, what you need to be able to present is a plausible promise. Your program doesn't have to work particularly well. It can be crude, buggy, incomplete, and poorly documented. What it must not fail to do is (a) run, and (b) convince potential co-developers that it can be evolved into something really neat in the foreseeable future.*

In order to display vitality of the project and engage early adopters, it is important to "release early and often" and listen to feedback from users (see Appendix B, lesson 7). Likewise, "barriers to entry" should be kept low for both users and voluntary co-developers. This is a "twofold task", which according to Fogel (2005, pp.10-13) adds complexity to the project's initial presentation to the world: Software users and developers have different needs and expectations, so the project maintainer must be careful to address both audiences in parallel. Users may demand a slick homepage, intuitive installers and documentation, while developers require a well working community forum / mailing list and the facilitation of suitable toolkits (see chapter 2.3.4). In a commercial context, this becomes a question of building deliberate marketing strategies.

Developers typically emerge from the user base, but also from elsewhere[30]: "drawn in by the technical challenge, kudos, or opportunity to improve or publicise their programming skills". Given the existence of competing FOSS projects (i.e. offering similar functionality), differentiation and expectation management can serve as effective strategies to attract developers from elsewhere. Fogel suggests the use of a short and concise mission statement

[29] Chapter "*Necessary Preconditions for the Bazaar Style"* in Raymond 1997, retrieved May 18, 2012 from http://catb.org/esr/writings/homesteading/cathedral-bazaar/ar01s10.html

[30] OSS Watch, How to build an open source community, retrieved May 19, 2012 from http://www.oss-watch.ac.uk/resources/howtobuildcommunity.xml

to announce the project's purpose and identity. He exemplifies this by the mission statement used by OpenOffice.org (loc. cit.):

> *To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format.*

Besides technical differentiators, it clearly states that this is a community-driven project (in spite of, or because of, the project being initiated by Sun Microsystems). This, apparently, is to appeal to "open" credentials of potential volunteers. Further, it positions itself as an international project, indicating demand and hospitality for developers from around the world.

Strictly speaking, OSS communities need not be confined to free-standing individuals. External entities with vested interests in OSS software (e.g. partners or third party providers of complementary offerings) are also inclined to contribute code, bug-reports, legal help and financial resources. Depending on circumstance, the project may want to address such entities through a dedicated marketing channel and organizational unit. As this is only relevant for a few well-established OSS projects (e.g. the Linux Kernel / Linux Foundation), it is a rarely incurred scenario. In the particular case of dual licensing, however, corporate participation in the community is the norm: The open source edition[31] is then typically managed by an employed "community manager", and the company's developers partly contribute as members of the community. This provides an additional option of vitalizing the community with "synthetic" activity.

[31] I.e. the edition that is licensed under an OSI-approved license.

2.3.3 Governance

> *OSS governance can be defined as the means of achieving the direction, control, and coordination of wholly or partially autonomous individuals and organizations on behalf of an OSS development project to which they jointly contribute. (Markus 2007, p.152)*

Whichever governance model a OSS project chooses to follow, it has considerable impact on stakeholders' perception of the project as whole, as well as the community's internal power distribution and *modus operandi*. As OSS Watch puts it[32]:

> *There are almost as many variations of open source management strategies as there are open source projects. It is therefore critical that a project clearly communicates its policies and strategies to potential users and developers of the project's outputs. A clear governance model also allows potential contributors to understand how they should engage with the project [..]. In addition, it describes the quality control processes that help to assure potential users of the viability of the project. The development and communication of a clear and concise governance model is one of the most important steps a project can take towards sustainability through open development.*

Not all OSS projects start off with a formal governance model, but evolve from a natural initial state in which the project's initiator serves as a *benevolent dictator* (BD). According to Fogel (ibid., p.60) one should think of a BD as "community-approved arbitrator" or "judge":

> *Generally, benevolent dictators do not actually make all the decisions, or even most of the decisions. It's unlikely that one person could have enough expertise to make consistently good decisions across all areas of the project, and anyway, quality developers won't stay around unless they have some influence on the project's direction. Therefore, benevolent dictators commonly do not dictate much. Instead, they let things work themselves out through discussion and experimentation whenever possible. They participate in those discussions themselves, but as regular developers, often deferring to an area maintainer who has more expertise. Only when it is clear that no consensus can be reached, and that most of the group wants someone to guide the decision so that development can move on, do they put their foot down and say "This is the way it's going to be." Reluctance to make decisions by fiat is a trait shared by virtually all successful benevolent dictators; it is one of the reasons they manage to keep the role.*

[32] OSS Watch, How to build an open source community, retrieved May 22, 2012 from http://www.oss-watch.ac.uk/resources/governanceModels.xml

Even large projects can be run according to a BD model. The most cited example is probably that of the Linux Kernel, which has been steered by Linus Torvalds since its inception. As with all dictatorships, however, there are inherent dangers: For one, the BD must hold a number of qualities in order to execute this role successfully (e.g. arbitration skills and the ability to delegate). Should the BD fail to adequately meet such minimum standards, he / she is likely to invoke dissatisfaction and schisms within the community (ultimately provoking a fork*). Second, even if the current BD performs outstandingly, there is the inevitable problem of succession – who / what model will take over when the BD eventually abdicates?

An alternative governance model is meritocracy, as embraced by The Mozilla Foundation[33]:

> *Mozilla is an open source project governed as a meritocracy. Our community is structured as a virtual organization where authority is distributed to both volunteer and employed community members as they show their abilities through contributions to the project.*

In practice, OSS projects do not need to restrict themselves to one single stereotypic model of governance. Rather, they tend to blend democratic, oligarchic, meritocratic and autocratic elements – even when governed by a "self-appointed benevolent dictator for life"[34]. Research (e.g. O'Mahony and Ferraro, 2007) further suggests that the blend of such models may change as OSS projects mature. This makes sense. As scope and complexity grows and individuals raise to prominence, informal changes naturally occur. Formal changes can either follow to reflect informal ones, be based on a realization of an inadequate status quo, or be implemented pro-actively in anticipation of emerging needs. Specialized projects may also be constituted under an umbrella of existing ones, with a limited mandate to adapt inherited government models (e.g. the Apache Software Foundation with its numerous progenies).

In the context of commercial dual licensing, OSS governance becomes a key business management process. As such, it needs to be aligned with other business activities. Governance models of dual-licensed OSS editions may therefore display a tendency towards commercial bias. To prevent the community from being provoked or alienated it may therefore seem imperative to balance commercial inclinations against "open" interests. But, as discussed in later chapters, this is not necessarily the case.

[33] Mozilla Governance, retrieved June 6, 2012 from http://www.mozilla.org/about/governance.html
[34] Which is the case for Ubuntu. See http://www.ubuntu.com/project/about-ubuntu/governance

2.3.4 Toolkits for Innovation and Development

OSS projects rely heavily on technological enablers, most notably the Internet: "The widespread diffusion of Internet access in the early 1990s led to a dramatic acceleration of open source activity. The volume of contributions and diversity of contributors expanded sharply, and numerous new open source projects emerged […]" (Lerner and Tirole 2002, p.202). Through effective utilization of newsgroups, FTP-servers and Email, communication costs were drastically reduced and geographically dispersed actors were able to join (Bonaccorsi and Rossi 2003). To further enhance efficiency, FOSS communities have since devised increasingly advanced ancillary tools[35] for code editing[36], issue tracking, testing, revision control, project management etc. The composition of an OSS project's endorsed toolkit is largely determined by the project maintainer, but may also (depending on the governance model) be influenced by community members. Since technical enhancers embody inscribed imperatives of use and encourage different modes of operation, the choice and facilitation of shared tools is no trivial undertaking. Developers may hold expectations which the project maintainer is not able / willing to meet, or vice versa – the project maintainer may want to impose tools that could alienate the community. Likewise, a lack of competencies or resources (e.g. interest, attention or time) may lead to organizational inertia. But such shortcomings give little excuse nowadays, since services as SourceForge, Google Code and GitHub can provide ample functionality to OSS projects.

OSS projects typically complement generic tools with specialized adaptations and in-house kit, which they offer to developers (e.g. for automation of often incurred and / or precarious tasks) and end-users (e.g. for diagnostics). Franke and von Hippel (2003) also propose user-friendly toolkits to facilitate customization and spur innovation amongst a heterogeneous users base[37]. As such technical aids can lower the barriers to entry and enhance development efficiency, it is important for the project maintainer to be aware of their potential and open to suggestions from the community (exemplified in next subchapter).

[35] Often called CASE tools (computer-aided software engineering).
[36] I.e. integrated development environments (IDEs, i.e. Eclipse), which are integrated with other development tools (e.g. versioning control systems, syntax checkers, bug trackers etc.).
[37] Further, what von Hippel terms *User toolkits for innovation* "allow manufacturers to actually abandon their attempts to understand user needs in detail in favor of transferring need-related aspects of product and service development to users along with an appropriate toolkit." (2001, p.247)

2.3.5 Innovation and development methodology

> *The essence of open source is not the software. It is the process by which the software is created. […] Production processes, or ways of making things, are of far more importance than the artifacts produced because they spread more broadly – Weber 2004, p. 56*

What sets OSS apart from conventional proprietary software is a methodology with an inherent focus on Internet-enabled distributed development and innovation – widely denoted as the "Bazaar" development model. As Raymond (1997) rounds up his landmark essay on the subject[38]:

> *Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software "hoarding" is morally wrong (assuming you believe the latter, which neither Linus [Torvalds] nor I do), but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem.*

What is particularly interesting about Raymond's assumption of methodological supremacy, along with the pragmatic stance towards commercial interests, is the confidence with which it is presented. Clearly, Raymond was able to put forward convincing arguments for why open source, epitomized by the Bazaar model, would put "closed software" development methodologies to a challenge. It is the default methodological choice for OSS projects[39], and has inspired emulation in both academia and the private sector. To be fair, many of its characteristics have always been intrinsic to academic research: open revealing and distribution of results, peer review and distributed collaboration. This is no coincidence. The Bazaar model is based on lessons learned from the making of Linux, which was incepted by Linus Torvalds in 1991 as he was graduate student at the University of Helsinki. There he became familiar with Minix, a minimalistic Unix clone made specifically for teaching purposes. In the absence of a free alternative and his preference for an Unix-like system over MS DOS, he installed it on his computer. At the time, a free alternative[40] had been in the

[38] "*The Cathedral and the Bazaar*", through which the term was popularized. Linus Torvalds, whom he here refers to, is the initiator and project coordinator of the Linux kernel – the core component of the GNU / Linux operating system (commonly referred to as just "Linux").

[39] In various forms – not all OSS projects follow the model "to the letter", or are even aware of it. But as its underlying philosophy is well reflected in the OSI's Open Source Definition's, they are innately bound to its guiding principles. It is important to note, however, that far from all projects manage to attract developer pools to sufficiently support the application of the model. See for example Healy and Schussman (2003).

[40] The GNU operating system (administered by the FSF's GNU Project), with unfinished kernel named "Hurd".

working, but was still missing its most central component – a kernel[41]. Torvalds therefore started to experiment on his own kernel – Linux. In August 1991 he announced the project to the Minix newsgroup[42]:

> *I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) […]. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat […T]hings seem to work. This implies that I'll get something practical within a few months, and I'd like to know what features most people would want. Any suggestions are welcome […]:-)*

About a month later, with a working version now at hand, he posted a follow-up (excerpts from ibid., with correction of minor typographic errors):

> *Do you pine for the nice days of minix-1.1, when men were men and wrote their own device drivers? Are you without a nice project and just dying to cut your teeth on an OS you can try to modify for your needs? Are you finding it frustrating when everything works on minix? No more all-nighters to get a nifty program working? Then this post might be just for you :-) […]*
>
> *I can (well, almost) hear you asking yourselves "why?". Hurd will be out in a year (or two, or next month, who knows), and I've already got minix. This is a program for hackers by a hacker. I've enjoyed doing it, and somebody might enjoy looking at it and even modifying it for their own needs. It is still small enough to understand, use and modify, and I'm looking forward to any comments you might have. I'm also interested in hearing from anybody who has written any of the utilities/library functions for minix. If your efforts are freely distributable […] I'd like to hear from you, so I can add them to the system.*

A community of geographically scattered beta-testers, code donators and co-developers was born, and the project gained traction. Frequent releases and a growing number of peer-review "eyeballs" assured that bugs were spotted quickly and fixes timely distributed back to the community. Torvalds treated community members as an indisputable resource, and they responded in kind by becoming so. Although Torvalds' position as the project's undisputed leader was assured, he handled his co-developers as what they were – free and autonomous agents with aligning interest. Any form of coercion, which is not uncommon in the context of employment, would have been futile. According to Raymond (1997 p.52), thus, "[t]he Linux

[41] Responsible of managing a computer's hardware resources and providing applications access to them.
[42] *LINUX History*, by Linus Torvalds, retrieved June 17, 2012 from http://www.cs.cmu.edu/~awb/linux.history.html

world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved." Yet, Torvalds' role as benevolent dictator was a coordinating prerequisite to this "self-correcting spontaneous order". As Larry McVoy (a community member at the time) put it in an interview of 2002[43]:

> *In my opinion, then and now, Linus [Torvalds] is what makes Linux great. He's the glue that holds it all together. Without him, Linux would splinter […].*

All this was made possible by the Internet's emergent ubiquity (Bonaccorsi and Rossi 2003), which Torvalds effectively utilized for code distribution and coordination. Still, most Internet-enabled "toolkits for innovation" (see chapter 2.3.4), which FOSS developers take for granted today, were relatively crude or non-existent at the time. For example, as Linux project was experiencing growing pains in 1998, the very same McVoy acknowledged[44]:

> *It's clear that our fearless leader is, at the moment, a bit overloaded so patches may be getting lost. There are some of us, myself among them, that have been worried about this for a while and are working on a solution.*

The remedy he prescribed was the BitKeeper, a revision control system he himself had "conceived to keep Linus from getting burnt out"[45]. After testing it, Torvalds recognized that it was indeed "the best tool for the job" (ibid) and adopted it. However, the use of a commercial product by an OSS project was controversial on principle. And after its free provision was withdrawn amongst allegations of reverse-engineering, Torvalds decided to develop his own open source alternative – git. It has since become the revision control system of choice for many software projects, even among commercial ones.

Not only does this chain of events parade the Bazaar model in an explicatory fashion, but it also displays the capabilities of the individuals it attracts and efficiently engages for a shared purpose. For a more detailed understanding of the model, please see Appendix B.

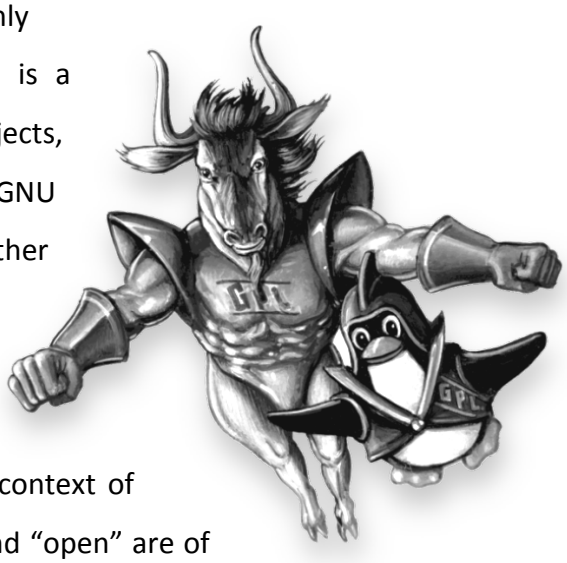
---

[43] *Interview: Larry McVoy* (May 28, 2002), retrieved June 18, 2012 from http://kerneltrap.org/node/222

[44] Post to the Linux Kernel Mailing list (September 30, 1998), titled "*A solution for growing pains*", retrieved June 18, 2012 from https://lkml.org/lkml/1998/9/30/122

[45] *Feature: No More Free BitKeeper* (April 5, 2005), retrieved June 18, 2012 from http://kerneltrap.org/node/4966

At this point it should be clarified that what is commonly referred to as the Linux operating system in fact is a composition of software originating from several projects, among them the Linux kernel and various GNU components. Hence, the FSF insists that it should rather be called GNU/Linux. Even if the latter naming convention is less popular in the OSI camp, it demonstrates how well FOSS software can work together in practice (see adjacent graphic[46]). In the context of integration and distribution, thus, the labels "free" and "open" are of little significance. The only prerequisite is that all involved code bases are governed by compatible legal frameworks, which brings us to the next chapter.



2.3.6 Legal Framework

What Torvalds did not know when he conceived Linux was that another project was pursuing similar Internet-enabled efforts from Berkeley[47]. A major contributing factor to Linux' success, as is widely understood, was the legal limbo which impeded the competing project. While UC Berkeley and AT&T's Bell Labs were entangled in a dispute over code ownership, Torvalds could offer a hospitable legal safe-haven. His decision to distribute Linux under the GPL license[48] was crucial in this regard, and gave him a competitive advantage: It allowed the Linux kernel to be liberally intermixed with legally compatible code from external sources (e.g. GNU), and remixed by anyone into customized "distros". In 1993, when the dust started to settle at Berkeley, Linux had already gained superior momentum. As Torvalds publicly recognized four years later[49]:

Making Linux GPL'd was definitely the best thing I ever did.

[46] FSF image of GNU and Linux penguin, from The GNU web site, entitled: *The Dynamic Duo: The Gnu and the Penguin in flight*. It is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.
[47] 386BSD, also called "Jolix", of which FreeBSD and NetBSD are popular offshoots.
[48] GNU General Public License version 2, released by the FSF in 1991, used by Linux as of version 0.12 (1992).
[49] Hiroo Yamagata (August 5, 1997), *The Pragmatist of Free Software: Linus Torvalds Interview*. Retrieved June 26, 2012 from http://www.tlug.jp/docs/linus.html

Copyright, licensing and collaborative innovation

Copyright poses a challenge to collaborative innovation and development. This is because changes to someone's code (derivative work) cannot be distributed without explicit consent of the original author. One way to circumvent this problem is to develop a patch (i.e. technically compatible complementary code), and distribute it independently from the original work. But even with such a mode of collaboration, challenges remain:

- Other developers cannot modify the patch without its author's explicit consent (i.e. the same problem, but related to a different chunk of code and copyright holder).
- Compatibility issues: Patches that build upon the original software are dependent on a predictable mode of operation. Therefore, changes to the original software pose a risk of rendering patches incompatible.
- Legal uncertainty: Even if all contributors informally agree to let others change and redistribute their code (such as IBM's "non-assertion" pledge), there is no guarantee that all involved parties will waive their copyright indefinitely.

Of course, if all collaboration is done within a legal framework of employment and all collaborators work for the same entity, then copyright ownership is likely assigned to that entity by default. Conversely, in the context of open collaboration, other legal mechanisms have been devised to administer code ownership under a centralized regime (e.g. foundations or corporations serving as *project maintainers*). The most common forms are:

- Contributor License Agreements (CLAs): A CLA is a formalized agreement, signed by copyright holders, which grants rights for contributions to be redistributed by the project maintainer. Further, CLAs "simplify the process of collaborating with partners on projects as they can form part of a broader collaboration agreement."[50] Thus, as commonly defined, CLAs do not involve any formal transfer of copyright ownership, but secure the project maintainer extensive rights of use[51] .

[50] OSS Watch, retrieved Jan 15, 2012 from http://www.oss-watch.ac.uk/resources/cla.xml

[51] E.g., the Individual Contributor License Agreement V2.0 of the Apache Software Foundation states: "*You hereby grant to the Foundation and to recipients of software distributed by the Foundation a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, sublicense, and distribute Your Contributions and such derivative works.*". Retrieved Jan 15, 2012 from http://www.apache.org/licenses/icla.txt

- Copyright Assignment Agreements (CAAs): A more stringent approach is to require collaborators to assign their copyright (or significant parts of it) to a project maintainer. This owning entity "then licences back to the contributor the licence to exercise all rights associated with the copyright material and the contribution"[52].

As copyright ownership is a valuable asset (see next page), CAAs are naturally favored by commercial entities. Even the FSF has encouraged this approach for software that is published under the GPL, claiming procedural benefits should the need for copyright enforcement arise[53]:

> *In order to make sure that all of our copyrights can meet the recordkeeping and other requirements of registration, and in order to be able to enforce the GPL most effectively, FSF requires that each author of code incorporated in FSF projects provide a copyright assignment, and, where appropriate, a disclaimer of any work-for-hire ownership claims by the programmer's employer. That way we can be sure that all the code in FSF projects is free code, whose freedom we can most effectively protect, and therefore on which other developers can completely rely.*

This requirement is only mandatory to FSF projects' contributors – other projects using the GPL are not bound by it. Thus, one of the reasons for not upgrading the Linux Kernel's GPL license from version 2 to 3 is the unfeasibility of obtaining the required consent from all copyright holders of the Linux code repository[54].

Naturally, the legal terms of CLAs, CAAs and the like vary, and cover a wide spectrum of possible terms of use and ownership distribution. Their actual application is prescribed by a projects governance model. Typically, community members are required to give explicit consent before obtaining "commit rights" (i.e. a procedural prediction for their contributions to be included in the code base). This provides the project maintainer exclusive gatekeeper capabilities.

---

[52] GPL software: Ownership, retrieved Jan 15, 2012 from http://bulkzooi.wordpress.com/legal-overview/gpl-software-ownership/

[53] "*Why the FSF gets copyright assignments from contributors*", by Professor Eben Moglen, Columbia University Law School. Retrieved Jan 16 2011 from http://www.gnu.org/licenses/why-assign.html

[54] In 2004 the Linux Kernel project introduced its own variation of an CLA, which requires contributions to be "signed off" by their authors to attest a "Developer's Certificate of Origin" (DCO).

The benefits of copyright ownership

It is the copyright holder's sole privilege to decide under which conditions his source code is to be modified, distributed and used. Even if it is already licensed under the GPL, the owning entity can choose not to apply the same terms to itself. Thus, the owner has the option to release the same source code under several different licenses (dual-/multi-licensing). While CLAs authorize project maintainers to license the collective code repository to third parties, they typically dictate which license categories that can be applied. Such limitations can pose a challenge at a later stage. For example, if the project's code was to be distributed as an integrated whole with other open source software packages, license compatibility issues could arise – i.e. licenses with contradictory requirements that make some integrated distributions unfeasible. Of such reasons, the right to re-license the software can be of great benefit. Although re-licensing is possible with CLAs (Jensen and Scacchi, 2011), the process is far easier with CAAs. Thus, to summarize, CAAs are sought of two purposes (Rosen in ibid., p. 13):

1. So the project can defend itself in court without the participation and approval of its contributors.

2. To give the project (and not the contributor) the sole right to make (re)licensing decisions.

There are also examples of joint license agreements, which provide "the contributor and project […] equal and independent copyright to contributed source" (loc. cit.). However, copyright ownership has limited value to single contributors, since atomic pieces of code have little utility outside the context of peer contributions. The project maintainer, on the other hand, would still hold the aggregate value of all congruent contributions.

FOSS licenses

The Free Software Foundation (FSF) has published a set of criteria for "free software" licenses [55]. Correspondingly, the Open Source Initiative (OSI) has published its set of requirements[56]. Both maintain lists of officially certified licenses, which overlap to a great extent. This is why many open source projects, such as the Linux Kernel (proclaimed as open source), apply FSF -authored licenses.

In essence, *[t]o be OSI [or FSF] certified, the software must be distributed under a license that guarantees the right to read, redistribute, modify, and use the software freely* (Deek and McHugh, 2007, p.244). This is to grant anyone (non-discriminatory) the right to:

- Make copies and distribute the software freely
- Full access to the source code
- Modify the source code and redistribute derivative work

Beyond that, FOSS licenses fall in one of two main categories. Restrictive ("copyleft") licenses (such as the GPL) contain reciprocal ("share-alike") provisions, which dictate any derivative (or integrated) work to be distributed under the same licensing terms. Hence, copyleft licenses are hereditary and self-perpetuating by nature. In contrast, permissive licenses[57] impose few restrictions on how the software is redistributed or integrated with other code[58].

While permissive licenses maximize legal compatibility with other software, copyleft licenses are meant to ensure that the community keeps control over the code and maintains access to external modifications. Proponents of the latter typically base their stance on ethical principles of users' freedoms[59], whereas advocates of permissive licenses stress the practical advantage of widespread software applicability through unrestrained reuse of code.

[55] *The Free Software Definition*, retrieved Jan 13, 2012 from http://www.gnu.org/philosophy/free-sw.html
[56] *The Open Source Definition (Annotated)*, retrieved Jan 13, 2012 from http://www.opensource.org/osd.html
[57] Often referred to as "research" or "academic" style licenses, as they are widely used by academic institutions
[58] I.e., they typically allow for redistribution with additional license terms (even reciprocal or proprietary).
[59] *Freedom or Power?*, by Bradley Kuhn and Richard Stallman. Retrieved Jan 13, 2012 from http://www.gnu.org/philosophy/freedom-or-power.html

License compatibility and license augmentation

One fundamental objective of free- and open source software is "to promote the free exchange of ideas and technology without fear of infringing the rights of others" (Morin, Urban, et al. ,2012, pp. 3-4). As such (loc. cit.),

> code licensed under like-minded FOSS terms should be freely combinable to create new products. Compatibility is the attribute of software licenses that allows combining of program code. To be compatible, license terms must be free of contradictory or mutually exclusive requirements. Alas, some FOSS licenses contain terms "incompatible" with other FOSS licenses, thereby diluting the ability to easily combine code.

Generally speaking, the more permissive a license, the more likely it is to be compatible with other licenses. Thus, source code governed by a truly permissive license can be combined with code administered by virtually any form of license (even proprietary or copyleft). As a result, the combined code base must always adhere to the most restrictive terms of involved licenses – hence the term "viral", which is often applied to copyleft licenses. Naturally, the same logic applies to license augmentation when no code merger is involved: Truly restrictive licenses leave no room for augmentation (unless explicitly stated for specific purposes[60]), while permissive licenses give few reasons to do so – they may even allow binary (closed) code to be redistributed and embedded without disclosing the source (ibid.).

[60] E.g. section 7 ("Additional terms") of the GPL version 3

2.3.7 OSS Business Models

"No matter who you are, most of the smartest people work for someone else." – Joy's Law[61]

While established enterprises utilize OSS to build competitive advantage, OSS startups have engraved it in their very DNA. Depending on their business model, market entrants harness it differently. The most generic OSS business models can be summarized as such:

- Platform providers: This is the business model pioneered by Red Hat. The company assembles and adapts various OSS packages into optimized quality certified software distributions. The firm also benefits from an open source community[62], which supports in-house innovation and development capabilities. Revenue is generated by offering supplementary professional services (migration, integration, management, support, training etc.) that go along with commercial "enterprise editions".

- Consultancy services: This model is employed by companies that do not directly engage in the development of OSS, but focus on providing consultancy services relating to pre-existing software solutions. An example is the identification and integration of fitting OSS offerings based on specific customer needs.

- Proprietary components: Open source licenses allow OSS code to be integrated with proprietary plugins[63]. The rationale is to satisfy specific customer needs that are not met by the open source solution itself. The drawback of this model is that the proprietary software over time may be replaced by a pure open source substitute if this functionality attracts enough attention by open source communities.

- Dual licensing (multi licensing): This strategy can be defined as a meta-business model, which supports the application of parallel approaches for targeted market segments. The main rationale is to draw the benefits of an open source community and brand, whilst at the same time allowing commercial distribution of the software under differentiated non-OSS terms.

Naturally, these stereotypic forms can be intermixed into more complex business models.

[61] Attributed to William Nelson Joy, co-founder of Sun Microsystems
[62] For the OSS Fedora distribution, which has a mutual beneficial relationship with Red Hat.
[63] Permissive licenses allow tight integration, while restrictive licenses may require functional encapsulation (e.g. using a permissively licensed "bridge" module to load proprietary plugins at runtime).

2.4 Commercial Open Source Software

> *I've always wondered where the word "oxymoron" came from. What does "oxy" have to do with "moron"? What about the words "commercial" and "open source"; do these words form an oxymoron when combined in one phrase?* – Sterne and Herring, 2006

In spite (or because) of its semantic ambiguity, the term *commercial open source software* (COSS) is widely used for marketing purposes. Arguably, such protuberant use of "open source" implies a certain appeal amongst investors, customers and OSS community members – or at least that it is perceived as "kosher". However, as COSS may indicate a variety of business models, a clarification is required for academic use. This thesis will apply the term as initially understood by Dirk Riehle, who recently found it necessary to rename the title of his 2009 paper to avoid confusion[64]. What he now calls "single-vendor" COSS entails the following (Riehle 2012, p.5):

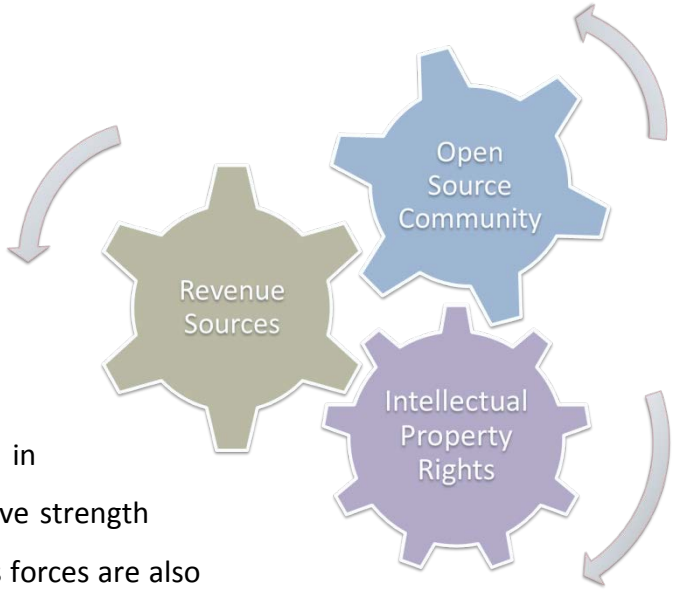
> *Single-vendor commercial open source software projects are open source software projects that are owned by a single firm that derives a direct and significant revenue stream from the software. […] Using a single-vendor open source approach, firms can get to market faster with a superior product at lower cost than possible for traditional competitors.*

Building on our understanding of OSS from the previous chapter, this classification infers (1) that an OSS (preferably OSI-compliant) license governs the source code, which to a certain extent is developed by an OSS community under the auspices of the firm. And (2), that some form of copyright assignment mechanism assures the firm legal ownership of the entire code base[65], and thus the exclusive right to relicense it under proprietary terms. Following, COSS applications are likely to be distributed under at least two licenses – one providing "unrestricted use" (community edition), and one intended for commercial purposes (Capra and Wasserman, 2008). The two versions of the software may differ, "with the commercial version including features that are not present in the unrestricted version. In that situation, the commercial version of the software typically includes some closed source code" (ibid, pp.1-2).

[64] From "*The Commercial Open Source Business Model*" to "*The Single-Vendor Commercial Open Source Business Model*". See Riehle, 2012

[65] Or alternatively, the firm does not accept outside contributions to the core code repository.

To fully understand COSS as a business model, one also needs to comprehend its intrinsic components: Intellectual property rights, revenue sources and a commercially steered open source community[66]:



As the figure shows, these aspects can be interpreted as interlocking cogwheels, which in concert drive the COSS apparatus. On the face of it, the implied dynamics appear simple. In reality, however, the picture is far more complex. It is the COSS firm that acts as the chief mechanic, and thus is in power to determine the shape and relative strength of each component. Yet, other exogenous forces are also at play, and insufficient lubrication of friction points can lead the entire machinery to break down.

This chapter addresses each of the three components in turn, before discussing issues relating to COSS community governance: Fairness, trust and community heterogeneity. With the theoretical foundations of OSS and COSS then firmly established, it will then conclude with an integrated view of the system as a whole.

[66] Dirk Riehle (2009), *The Business Model of Commercial Open Source Software*, p.7. Retrieved Mar 13, 2012 from http://dirkriehle.com/wp-content/uploads/2009/08/Commercial-Open-Source-v2.pdf.

2.4.1 Intellectual Property Rights

According to Riehle (ibid, p.9), the COSS model is founded on two legal preconditions, of which the first should be familiar to the reader:

1. Full control over intellectual property rights (copyright, patents and trademarks), is instrumental in maintaining the exclusive rights for relicensing and branding.

2. An open source license that is suited to "drive adoption" and "stall possible competition".

Riehle argues that reciprocal (copyleft) licenses "like the GPL" are well suited for that purpose: They effectively hamper the reuse of code by competitors under proprietary terms, and require any improvements by others to be disclosed. In fact, a high-quality and well established FOSS-licensed product may by itself have an preemptive effect on competition: Its mere existence rises barriers to entry by being offered at virtually no cost. Meanwhile, the COSS firm can relicense and profitably distribute the software (either as binary- or source code) to *OEM*s*, *ISV*s* and *VAR*s* that do not wish to be subject to a copyleft license (e.g. in order to embed or combine the code in a proprietary product). Likewise, it can sell commercial user licenses of enhanced editions (e.g. "professional" or "enterprise"). Last but not least, it can benefit from providing open APIs[67] or permissively licensed "bridge" modules to facilitate easy integration with complementary offerings.

In academia, the segmentation of code according to legally differentiated boundaries has become known as "IP modularity" (Henkel and Baldwin, 2010). COSS companies may draw the boundaries differently according to individual assessment of tradeoffs: Too restrictive boundaries (i.e. in proprietary terms) are likely to reduce value creation by the help of outsiders. On the other hand, if the boundaries are too lax (i.e. in favor of open source innovation) the firm's prospects of value appropriation are hampered. Likewise, the degree of modularity (in terms of fragmentation and / or overlaps) determines the costs of setting up, monitoring and enforcing the technological-legal IP framework. Hence, "the optimal modular structure for *capturing* value is not necessarily the same as the optimal structure for *creating* value" (ibid. p.14, original emphasis).

[67] Application Programming interfaces, which allow runtime compatibility.

2.4.2 Revenue sources and funding

COSS companies can apply different business models in combination, and draw revenues from a number of sources. These are, according to Riehle (ibid. pp. 9-10, with my own augmentations):

- Core product: Targeted at OEMs, ISVs and VARs, who "may pay for a commercial license to receive certification or indemnification or to embed the software into their products without having their own code touch open source code".

- Value-added product: Targeted at professional / enterprise users that require functionality beyond basic features of the community edition. COSS companies typically offer convenient upgrade from the community edition, which thus serves as a teaser / trial version.

- Operational comfort: To "ensure that the software reliably fulfills its duty" customers may want to pay for additional services, such as technical support, real-time monitoring or regular patching (bug fixes and quality assured upgrades).

- Consulting: To get the software fully operational, customers may want to pay for consulting services for installation, integration, customization and training.

In addition, one could add hosting in cases where the software can be accessed through a client (e.g. a Web browser). Such a "Software as a Service" (SaaS) delivery model typically involves additional technical resources, which can be packaged and sold according to specific customer demands (like storage capacity and provision of complementary software).

The diversity of revenue sources, and particularly high margin licensing, makes COSS companies popular among investors. An open source version (which drives adoption and innovation) and active community (implying reduced development, support and marketing costs) can further increase prospective value. Meanwhile, the 2008 acquisition of MySQL[68] by Sun Microsystems for USD 1bn has demonstrated that highly profitable COSS exits are possible. Accordingly, COSS companies "have attractive valuations and can raise capital at parity with proprietary software vendors" (Olson, 2005, p.79).

[68] MySQL raised USD 1m through venture funding in 2001, USD 19.5m in 2003 and USD 18.5m in 2006.

2.4.3 The role and composition of the COSS community

Riehle (2012, p.11), in reference to an interview[69] with former CEO of MySQL, Mårten Mickos, maintains that "almost all of the core product development work is carried out by the commercial [COSS] firm, with occasional contributions from the community". This claim is substantiated by others, like Capra and Wasserman (2008). They studied three COSS companies, none of which had more than 20% of their code base derived from community contributions[70]. Mickos explains why this is the case for MySQL (in ibid, pp.15-16):

> *The deeper you go into the core of the database engine, the more difficult it is for somebody to contribute because it takes five years to learn. If you build something on the outskirts of the kernel – some tool or function that you add on top of it – then it is much easier because there's less risk that you will mess up the whole product. But something great can emerge out of many tiny-looking contributions.*

Further, volunteers who demonstrate their skill by actually making it into the core and contributing valuable code make likely candidates for employment by the company. In a sense, therefore, the community gets crowded out from the core of the software. That also makes sense from a managerial perspective. The commercial development process is guided by stringent internal roadmaps, with a selective focus on particular milestones. This is unlikely to blend well with informal bazaar-style development (Scacchi, 2002; O'Mahony and Bechky, 2008). Much better, then, to direct the community's attention to peripheral functionality – which at any rate provides more low-hanging fruits for part-time volunteers. Peripheral functionality also gives the most immediate and tangible utility to the user, and is therefore well suited for user-driven innovation. For, as Mickos explains (in loc. cit.): "Their No.1 reason is that they are contributing so that they will get, in return, a better functioning product. They are looking after their own business interests". This understanding is supported by Shah (2006), who distinguishes between "open" and "gated" communities, the latter referring to communities with restricted access to the development process due to COSS companies' patronage and control. She established that gated communities are "populated almost exclusively by need-driven participants" (ibid. p.1004), whereas

[69] Former CEO of MySQL AB. See *An Interview with Marten Mickos, The Oh-So-Practical Magic of Open-Source Innovation*, MIT Sloan Management Review, Vol. 50, No. 1 (Fall 2008), 16.

[70] This percentage does not include contributions by non-employees who have obtained regular compensations by the COSS firm. It may however imply irregular compensations, such as "bounties", which some companies use as an incentive.

"enjoyment" is a more predominant motivational factor in open communities. Further, she suggests that many members of gated communities inhabit an initial reluctance to adopt gated software, and only do so when they cannot find an adequate open alternative. This also resonates with Steward et. al. (2006, p.133), in that "an association with a market organization could dampen enthusiasm among some volunteer developers because certain tenets of the open source culture seem to value independence from organizational constraints and, in some cases, disdain of profit motives" (in reference to DiBona et al. 1999, Stewart and Gosain 2006). As one anonymous respondent puts it (in Shah 2006, p. 1009):

> *I answer questions and stuff, but I don't feel the need to contribute my changes to the community. It's time consuming and I don't know if [the corporate sponsor] will do anything with it... .At the end of the day, they make the decisions with their commercial licensees in mind. (Long-term participant, gated source community, United States).*

And another, in a more resentful tone (loc. cit.):

> *I make the changes that I really need and so does everyone else and we benefit from one another.... There are a lot of things the project still needs that I keep asking [the corporate sponsor] to develop... they are not absolutely critical, but they'd take the software to the next level and expand its capabilities... if I develop it and then [the corporate sponsor] says I can't let others see it or work on it or use it in whatever way that makes sense – now come on! That's not how it works! (Long-term participant, gated source community, United States).*

Although not explicitly stated, both citations raise a commonly professed sensation of COSS software – namely that the community edition is deliberately kept limited in quality, utility and / or performance. This strategy, often termed "damaged good" or "crippleware", allows a COSS company more scope for commercial product differentiation and price discrimination, while it effectively repels cannibalization threats posed by the community edition. On the other hand, it is likely to alienate community members who wish to push the open source code towards its potential. For that reason, COSS companies are unlikely to confess to the pursuit of such a strategy. But one would probably be naïve to believe that they are not inclined to subtly follow it at least to some degree. Even when not the case, mere prejudice, mistrust and suspicion of its application can have significant effects on community recruitment and motivation.

In addition to ex-ante self-selection by users and contributors, migrations may occur when long-term expectations are not sufficiently met by the COSS firm. But given various forms of lock-in[71], which are likely to ensue once the software has been firmly adopted, a desire to leave can be offset by implied switching costs. Disgruntled community members may then become "captive" to the software and reluctantly linger on. In a worst case scenario, an aggrieved community constituency may eventually incite a partial exodus under the flag of a competing OSS project (fork).

At this point, it is important to remember that far from all users and contributors are freewheeling agents of sovereign will. As Mickos implied above, COSS contributions are to a large extent driven by commercial incentives. Customers and partners of the COSS company have vested interests in the software, and their employees represent a considerable share of community activity. Similarly, third party entities contribute complementary modules (e.g. connectors) and concurrent updates to increase the value of their own software. Even employees of COSS firms tend to be represented in communities (e.g. in forums). Whether pecuniary rewarded contributors should qualify as "real" community members remains a question of definition, and depends on the weight one puts on "voluntariness" as an absolute criterion. For all practical reasons, however, they are – which may explain why enjoyment is a relatively underrepresented motivational factor in gated communities.

Summarizing, we can say that COSS (gated) communities generally differ from open communities in at least three regards:

1. Governance and control: The firm sways regulative and operative control over the development process. It serves the function of a "benevolent dictator" not uncommon in conventional OSS projects, albeit with an inherent commercial motive.

2. Area of development: The firm tends to dominate the development of core functionality, while it encourages and benefits from the community's primary focus on peripheral complementary functionality.

3. Motivation: COSS communities predominantly attract utility-motivated users and developers, many of whom represent commercial vested interests in the software.

[71] E.g. acquisition of skills / training, integrated work-flows and complex data storage structures.

2.4.4 Governance: Fairness, trust and community heterogeneity

As just established, COSS projects tend to be firmly controlled and governed by a commercial entity. COSS communities thus seem to be at the mercy of their patrons, yet they in turn rely on the allegiance of their communities. According to Harhoff and Mayrhofer (2010, pp. 141-142), hybrid community-firm modes of collaborative innovation "will be sustainable only if the parties involved receive a sufficiently high benefit from the relationship and as long as the norms of the community, in particular with regard to sentiments of fair treatment, are not violated." This, they argue, is most likely achieved through a stable relationship in which mutual trust is fostered by two types of fairness:

- Distributional Fairness: "when the actual distribution of benefits is perceived as fair."
- Procedural Fairness: "when at least the procedures and criteria by which benefits are being divided are perceived as fair".

The two seem innately related – one might even say that the first is an operational manifestation of the latter (i.e. procedures and criteria as they are actually being applied). But while procedural fairness is reasonably easy to gauge[72], distributional fairness can only be judged by assessing successions of unilateral actions and bilateral interactions over time[73]. This may explain the focal role of "learning experiences", which Harhoff and Mayrhofer attribute to the building of trust (loc. cit.):

> *Hybrid innovation processes […] have a greater chance of becoming stable over time as the parties involved accumulate experience and build trust in each other. Any violation of norms of behavior can disrupt this process.*

Interestingly, Harhoff and Mayrhofer imply copyright appropriation to be potentially regarded as a breach of trust. This reasoning may seem instinctively implausible, since copyright assignment is the norm in COSS projects. Commercial copyright consolidation is an unlikely scenario to incur for previously distributed ownership arrangements, as any substantial community of volunteers would be likely to fend off such a transition.

[72] On the grounds that, in ANT-terminology, procedures and criteria for distribution of ownership / control are represented by the translation of underlying intentions into inscribed artefacts (e.g. licenses and formalized rules). As such, these artefacts constitute actants in their own right, and serve as mediating agents that prescribe fairly predictable patterns and scope for interaction.

[73] Since inscriptions can leave room for varying interpretations and be subjected to opposing interests (or "anti-programs") in the COSS actor-network. Moreover, procedures and criteria for distribution can be altered as a result of constructive negotiation or opportunistic behavior, thereby changing the basis for actual distribution.

Disapproval of commercial ownership can therefore be understood as a motivational ex-ante self-selection factor. Yet, as established in the previous chapter, even those who value independence from organizational constraints or disdain of profit motives find their way into COSS communities. In line with Harhoff and Mayrhofer, therefore, appropriation *also* has an ex-post effect on trust and perceived fairness. This, according to Fogel (2005, p.164) is "bound to raise tensions at some point, at least with some volunteers".

Another trust-related aspect of copyright attribution is a commonly shared fear among voluntary contributors that the commercial entity may eventually "take the project private". This would happen "if the company failed to license its further work as Open Source, and continued development as a proprietary product while abandoning the users and developers in the Open Source community" (Perens, 2011). In this light, copyright assignment is likely perceived as an "unconditional gift to the company with no quid-pro-quo for the Open Source developer" (loc. cit.)[74]. Such perceptions carry implicit connotations of unfair distribution of ownership and control, and may explain observations by Shah (2006, p.1009):

> *In describing their activities within the gated source community, participants voiced blatant disapproval of two elements of the governance structure: The level of code control held by the sponsor (the sponsor is the only actor able to make changes to the source code) and ownership by the corporate sponsor.*

The important question is whether such views are representative within and across COSS communities. Community heterogeneity can be measured along multiple dimensions. In terms of motivation, we have already established that contributions in gated communities are predominantly utility-driven. This does not mean that other forms of motivation cannot possibly be nurtured and utilized, rather that gated communities appear to provide unsuitable environments for them. It may be futile or counterproductive to cater for innately hostile dispositions towards commercial ownership, but it is conceivable that unrealized potentials can be tapped by loosening institutional control and giving more room for diversity. For example, Shah (2006, p.2012) notes that COSS firms may inadvertently dominate the flow of communications "through a desire to influence the direction of the project or because firm employees represent a substantial portion of the participant pool

[74] Note: Guaranteed "user freedoms" to derived work are commonly regarded as an essential quid-pro-quo among voluntary FOSS developers. Although copyleft licenses are intended to insure these conditions, they can be effectively circumvented through consolidated copyright ownership.

[..]. This may act to inhibit developers from voicing heterogeneous views, resulting in decreased volunteer participation." As a known COSS symptom, firms may thus consider to restrain their dominance over communication flows. Similarly, to build trust and dispel fears about projects being "taken private", Perens (ibid) recommends COSS firms to offer their communities a formalized "covenant" in exchange of copyright attribution.

Since managerial trade-offs may entail partial dilution of powers, one needs to be observant about the minimum degree of control required to sustain a COSS business model. Capra and Wasserman (2008) developed a framework[75] to evaluate managerial styles in open source projects, and used it to asses three COSS projects (OpenOffice, MySQL and SugarCRM). They identified four "fundamental governance dimensions"[76] according to which the projects were measured along a continuum between "fully open" and "fully closed" extremities. By implication, all these governance dimensions relate to fairness and control (e.g. by whom and how contributors' code is evaluated and allowed into the official repository). Capra and Wasserman unveiled substantial variety between the studied projects, which suggests that a generic COSS business model indeed allows for a range of managerial styles[77]. As such, community heterogeneity is not merely a phenomenon to observe and handle reactively. Through deliberate management, COSS companies can pro-actively cultivate or mollify community constituencies in order to achieve a suitable fit[78]. This does not only concern motivational factors. Skills, experience and needs also tend to be unevenly distributed among and between communities (Franke and von Hippel 2003, Shah 2006). Franke and von Hippel (ibid.) suggest "user friendly" toolkits as effective measures to address challenges of multiplicity in needs and technical proficiency[79]. Harhoff and Mayrhofer also consider skills as relevant. Based on their assumption that expert users are "least deterred from switching" they go on to infer that "[t]echnical sophistication may be highly correlated with aversion to

[75] The "Software Project Governance Framework" (SPGF)

[76] (1) Contribution (relative amount of voluntary code development), (2) Project leadership (distributed vs. hierarchical decision making), (3) working practices (geographically distributed and virtual communication vs. physical proximity and face-to-face interaction), and (4) testing (community peer-review vs. formalized "in-house" quality assurance of code contributions – i.e. a powerful gatekeeper role).

[77] OpenOffice has after the 2008 study experienced significant turmoil: LibreOffice as a fork in 2010, based on concerns that Oracle would introduce more restrictive terms or discontinue it (in relation to Oracle's acquisition of SUN Microsystems, which was the previous project sponsor); and a transferal of ownership and control from Oracle to the Apache Software Foundation in 2011. One should note that none of these changes are related to SUN's then management of the project.

[78] This concerns both existing and potential community members.

[79] In terms of lowering barriers to entry for unskilled / inexperienced users, catalyzing innovation, and facilitating customization according to unforeseen needs. Also see chapter 1.3.4

commercialization" (ibid p. 142). Interesting as it may be, this conjecture looks overly simplistic and speculative. But the reader should beware of committing an association fallacy by discrediting the authors' paper on this basis – the premise of expert mobility stands in its own right. Already in 2000, the renowned *Orbiten Free Software Survey* (Ghosh and Prakash) suggested high mobility among technology savvy contributors. It showed that a majority of community members participate in only one project, while a minority (some of which are institutions) contribute to numerous ones. Likewise, a mere 10% of authors accounted for over 70% of the combined code base (of the 12,706 surveyed projects). To my knowledge, however, no differentiating composition of motivational factors has been attributed to prolific contributors, nor have any forms of governance that agree particularly well with their (perhaps characteristic) preferences. The same goes for heterogeneity in terms of affiliation to the firm (e.g. partners, customers and community edition users). As a whole, and particularly in the COSS context, community heterogeneity remains an under-studied field that leaves much room for speculation. But the absence of certainty is no reason for managerial complacency or indifference. On the contrary, exactly because so little is known about underlying dynamics and possible ramifications of community heterogeneity, COSS companies are well advised to pay additional attention to it.

Consequences of community mismanagement can be severe. If coordinated community constituencies lose trust or confidence in the commercial leadership, they can establish a pure open source project based on the community edition (fork). Not only does the firm thereby lose valuable parts of its voluntary work force, but the resulting publicity can have high reputational costs. Further, the forked project can turn into a fierce competitor. Therefore, in the words of Harhoff and Mayrhofer (ibid. p.142):

> *While stable relationships can be attained, their management is likely to require constant attention. Firms will need to develop a comprehensive understanding of the community's ecosystem; treating the collaboration as another form of business relationship is likely to lead to failure.*

With this in mind, we are now in the position to conclude the Theory chapter.

2.4.5 An integrated view

As this chapter has established, OSS and COSS projects are complex socio-technical ensembles, comprising a multitude of interdependent components or aspects, or, as it were, "interlocking cogwheels". The ones most relevant to us are summarized in the figure below:

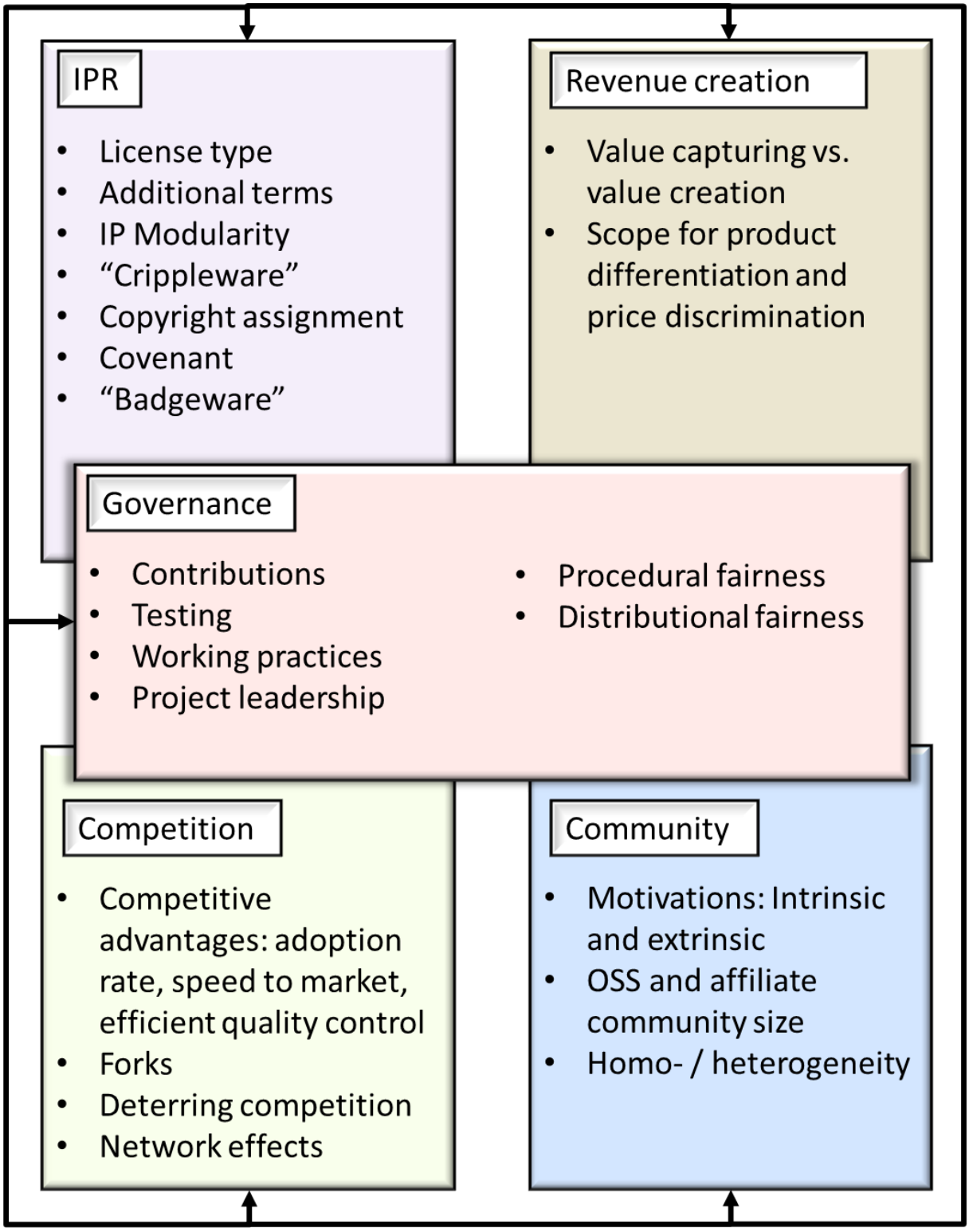


Figure 1: Generic, mutually dependent components / aspects of (C)OSS projects.

Various scholars have studied the inner workings of OSS and COSS projects, and sought to establish the mutual significance of measurable technical, legal and managerial aspects (e.g. Stewart et.al. 2006; Stewart and Gosain, 2006; Capra and Francalanci et al.,2009). Yet, too few studies exist to confidently determine which findings and inferred theories of causality are applicable only to a particular setting, and which can be credibly extrapolated to the general. Since all OSS / FOSS projects display unique endogenous and exogenous particularities, caution is in order when performing deductive or inductive reasoning. Thus, even if generic patterns are readily recognizable across OSS / FOSS projects, one also needs to consider how archetypical templates are adopted to a specific setting, and perhaps even supplemented or customized with innovative new features.

As we delve further into the problem domain of this thesis (chapter 4) and inquire into the three research questions (as posed in chapter 1), the reader should take notice of the methodological choice of grounded theory (presented in the following chapter), which guided the underlying research process. As this methodological approach inherently prescribes, the evolution of theory accompanies the ongoing process of analysis. For a consistent narrative, therefore, this is reflected through further introduction of theory in chapter 4.

3 Methods

This thesis reports on findings from an exploratory case study on SugarCRM, guided by three predefined research questions (see chapter 1). The underlying research followed a grounded theory (GT) approach, in which theory evolved successively as collected data was interpreted (Glaser, 1978; Strauss and Corbin, 1998). As such, the overall process was inherently iterative, in accordance with what Glaser describes as the "sequential, subsequent, simultaneous, serendipitous, and scheduled" nature of GT (1998, p.15). Likewise, in line with Glaser's famous "all is data" maxim (ibid, p.8), I kept an open mind with regards to potentially relevant data sources and data-gathering techniques[80]. Based on conceptual understanding that emerged early in the process, it also appeared fruitful to lend terminology from Actor-Network Theory (for the understanding and description of socio-technical semiotics), and theoretical perspectives from Phenomenology (for the interpretation of stakeholders' subjective motivations and normative positions).

Overall, the research process is summarized by the following figure:

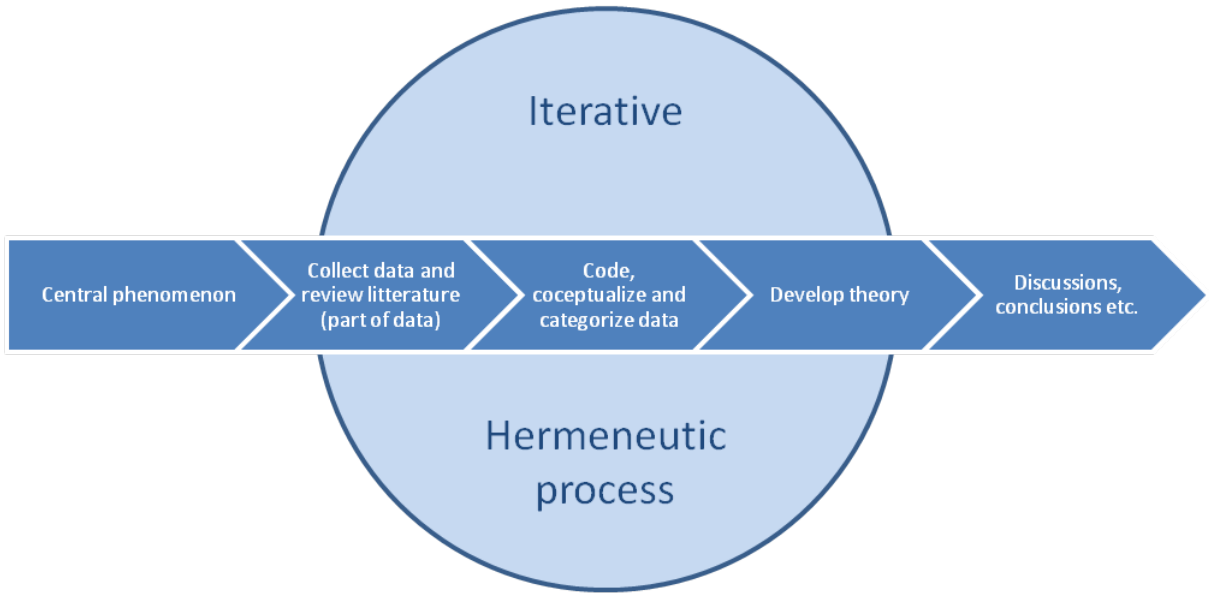


Figure 2: Simple schematic of the research process

[80] E.g. the intended use of a questionnaire for a complementary quantitative analysis.

3.1 Data collection and analysis

As is common for case studies, several data collection approaches were used for triangulation purposes (Maxwell, 1996; Creswell and Miller, 2000). Apart from epistemological considerations, the very nature of the problem domain guided the choosing of methodology:

1. SugarCRM's Community Manager had a tight schedule, and many questions would require some research from his side. Combined with the practical challenge of different time zones, this suggested an asynchronous mode of communication.

2. Community members relate to SugarCRM through web forums and miscellaneous online points of contact (e.g. *SugarForge**, developer blog[81] and webinars). Being publicly available, they are suited for observation to obtain first-hand knowledge. Likewise, a rich body of public commentary (e.g. press releases, forums and mailing lists) provided rich information, suitable for traditional hermeneutic interpretation.

3. The community is made up of heterogeneous constituents, which are not practicably distinguishable through observation. And even though observations deliver circumstantial evidence on participants' backgrounds and motivations, they cannot possibly provide a comprehensive understanding of subjective stakeholder perceptions. This suggested the use of an online questionnaire.

4. There is a wide body of academic literature available on open source in general, and a somewhat narrower selection on commercial open source in particular. Combined with previous research on SugarCRM, this literature provides valuable access to relevant findings and scientific interpretations.

Accordingly, four types of data sources / collection approaches were chosen as the basis for this thesis: (1) e-mail interviews with the Community Manager, (2) observations, (3) an online questionnaire, and (4) review of literature. The iterative process of data collection from these sources, coding / conceptualization of findings, and the advancement of theory was supported by the use of mind maps (MindManager software) and different forms of Rich Picture Diagrams (pen and paper).

[81] SugarCRM Developer Blog: http://developers.sugarcrm.com/wordpress/

3.1.1 E-mail interviews

Multiple rounds of e-mail interviews were conducted with SugarCRM's Community Manager, John Mertic. As noted by Opdenakker (2006), this form of interviewing has its drawbacks[82]. Yet, as pointed out by various scholars (e.g. Bampton and Cowton, 2002; Kivits, 2005), it also has characteristic benefits:

1. Significant savings in terms of required effort / increased feasibility: Asynchronous communication circumvents the challenge of finding mutually convenient times for time-pressured participants, which can be particularly hard to overcome with different time zones. And as a bonus for the researcher: No transcription is required.

2. Higher quality of information (Bampton and Cowton, 2002): "In permitting a lengthy delay between communications, an e-interview gives the interviewee time to construct a response to a particular question. For example, it provides an opportunity to find information which might be required", and "enables interviewees to reflect and then supply a considered reply". Correspondingly, the approach can "reduce the pressure felt by nervous interviewees".

3. Access to more information: Interviewees not only get time to gather and validate information, but also to provide a wider range and depth of information. E-mails are particularly suited for this, since they allow for attachments and hyperlink references.

As these benefits were considered highly relevant in the context of this thesis, they became decisive for my choosing of the e-mail approach.

To assure reliability, I was careful not to formulate the interview questions in a leading or ambiguous manner. Even though every single set of questions could be considered as structured (i.e. in written form), I would argue that the overall interview process was semi-structured (as new questions were raised on the basis of previous answers).

The focus of interviews changed over time: The first rounds were meant to clarify my general understanding of preliminary observations, and to establish the role of the Community Manager. Then, the focus turned towards community governance and implied

[82] In short: (1) Lack of social cues; (2) low chance of spontaneous answers (which in some cases are considered valuable – but not in this case); (3) Risk that the interviewee may forget the "ongoing" interview (which can be counterweighted with reminders – but nevertheless poses a practical challenge).

strategy. As my understanding matured, later interview rounds then became more ad-hoc in nature – based on accumulated questions that arose through my research.

In a sense, the Community Manager also served as a gatekeeper to other valuable data: He provided me a dataset, and facilitated the use of the online questionnaire.

3.1.2 Observation

I performed two forms of observation, which in consecutive order were:

- Exploratory: In order to form a preliminary understanding of the community and its modus operandi, I explored the web sites around which the community evolves: Forums, SugarForge, the Sugar Developer Zone, the Developer Blog etc. Based on these observations, I also looked at external web pages for a deeper understanding (e.g. early postings on vTiger forums – a SugarCRM fork). These observations, along with initial reviews of literature, served as an intermediate foundation for interviews and the online questionnaire.

- Targeted: Based on other findings and emergent understanding thereof, I actively looked for information of relevance. For example, I sampled 20 random projects on SugarForge to determine the degree of collaboration among developers, and studied SugarCRM's Trademark Policy to establish its relevance for strategic governance of the community edition (Sugar CE).

Since the objects of observation varied in form and nature, I had to apply different practical methods to conclude on their meaning and mutual significance. While the overall approach was informally hermeneutic, I sometimes found it necessary to take on more rigorous methods. For example, observations led me into performing document analysis (e.g. for the supplementary terms of the AGPLv3 license, and the SugarCRM Trademark Policy).

3.1.3 Questionnaire

In the absence of readily available data on SugarCRM's community composition and participants' motivations, I found an online questionnaire to be the best suitable approach. My initial aspiration was to perform multivariate analyses to discover covariance across a number of parameters, and collect qualitative data to guide the interpretation of observations. Regrettably, however, the questionnaire did not yield enough responses[83] to provide adequate confidence through statistical significance. I had also hoped that the resulting dataset could be used for comparative purposes[84], but this idea had to be abandoned for the same reason. Hence, the quantitative results of the questionnaire are not presented in this thesis.

The design of the questionnaire was guided by results from preliminary reviews of literature, observations and interview rounds. As with the interviews, I was careful to avoid ambiguous or leading formulations, and consciously adapted the language to my audience. To capture qualitative data, I also used optional, open fields to allow respondents to elaborate or provide otherwise relevant information.

The sample of respondents was naturally determined through self-selection. With the help of the Community Manager, the questionnaire was announced in forums, newsletters, and most importantly: On SugarForge. Although all community constituents thereby should have been equally exposed to its presence, representativeness cannot thereby be guaranteed. In fact, the relative low number of respondents, accompanied by generally high scores on self-attributed contributions, may suggest that the most active community members were overrepresented. This could be explained both by a higher degree of attentiveness among active members[85], and / or a correlated predisposition to participate.

Although the questionnaire failed in its primary intent, it nevertheless produced valuable qualitative data which was used for triangulation purposes. And even if the resulting dataset is unsuited for statistical analysis, I found it useful enough for rudimentary testing against other scholars' hypotheses.

[83] 62 full sets of replies were obtained by the beginning of September 2012, after the survey being prominently announced and marketed on SugarForge for about one month.
[84] The questions on motivation were purposely formulated so they would be compatible with datasets from other studies on open source community motivations.
[85] I.e. likelihood to read newsletter or visit the Web sites where the questionnaire was announced.

3.1.4 Review of literature

Even before beginning with preliminary observations, I started off by reviewing relevant literature. The rationale behind this approach was to (1) build reflective knowledge[86] on issues relating to the problem domain, (2) utilize this knowledge as a cognitive and conceptual foundation[87] for the other applied methods, and (3) share this with the reader through an informed and consistent narrative[88]. As one might expect, my overall focus followed a trajectory from the general (open source software) towards the specific (e.g. business models and SugarCRM itself). Yet, as new understanding emerged around interpreted specifics (e.g. licensing schemes), I was constantly compelled to revisit and readjust my general understanding. As such, the body of reviewed literature provided valuable dialectic contributions to an ever revolving hermeneutic circle[89]. These contributions can be considered as both complementary and supplementary to those provided by observations, interviews and the questionnaire.

The selection of literature was inherently guided by my research questions, along with corresponding search results from Google Scholar and suggestions made by my supervisors. Additionally, I actively looked for research that had not yet been published[90], and even came across some by pure serendipity[91]. The amount of literature was not deliberately determined by choice, but naturally followed from the scope of the thesis and diminishing returns of additional research (saturation).

As noted in the chapter on observation, I also performed document analyses and reviewed public commentaries (news articles, blogs, announcements etc.). In spite of methodological resemblance to literature review, these approaches were integral to exploratory observation efforts.

[86] I.e., in terms of hermeneutics and phenomenology: The review of literature facilitates a preliminary "fusions of horizons" (Gadamer) – that is, to widen my hermeneutical horizon (contextual understanding).
[87] Following from the previous: To use my widened hermeneutical horizon as a basis for observation, interview- and questionnaire design, and thereafter for the hermeneutic interpretation of findings.
[88] Both in the form of a literature review (see Appendix B), and as dialectic components / reference points in the analysis of other literature, secondary data / interpretations and own findings (see Analysis chapter).
[89] Or, rather, a hermeneutic spiral (to avoid implications of infinite regress).
[90] By querying other scholars / authorities in the given field.
[91] E.g. Waltl et al. (2012), which I was given directly by the main author after discovering that he also was doing research on SugarCRM.

4 Analysis

This chapter provides an analytical examination of collected data and reviewed literature (see chapter 3), based upon research questions presented in chapter 1 and theoretical framework of chapter 2. It starts with a general overview of SugarCRM to make the reader familiar with the object of inquiry, SugarCRM (Chapter 4.1). The two first research questions are then addressed in turn, and discussed in the light of my findings (Chapter 4.2 and 4.3, respectively). Building on this, the concluding remarks (Chapter 4.4) then also takes upon the last research question, which is more conceptual in nature and inherently follows from the two previous ones. Finally, a small subchapter (4.5) on future research concludes this thesis.

4.1 SugarCRM – a general overview

SugarCRM–a sweet mix of commercial and open source.[92]

SugarCRM Inc. was founded in 2004, with 10 employees, charging $149 per user per year for support services to its open source customer relationship management (CRM) solution. The company was able to raise $8m within its first year of operation ($2m in 2004, and $6m in 2005), and within 18 months its open source project had attracted a community large enough to support 21 different languages (Sterne and Herring, 2006). After successive rounds of funding and organic growth, the company turned cash flow positive in Q4 of 2010. In 2011 it proclaimed a new record, with a 92% year-on-year billings growth and annual revenue growth of 67%[93]. In April of 2012 it announced[94] it had completed a new $33m financing round (equity and debt) "for further expansion into the enterprise". In terms of open source development, it now claims that "[m]ore than 30,000 registered developers in the Sugar community drive SugarCRM innovation through testing, feedback, extension creation, and complementary applications"[95].

[92] Title of Sterne and Herring, 2006

[93] Press Release (2012-02-22), *SugarCRM Scores Another Banner Year in 2011,* Retrieved Sept. 18, 2012 from http://www.sugarcrm.com/newspress/sugarcrm-scores-another-banner-year-2011-0
Note: As the company is privately held, I wasn't able to obtain numbers for net profit.

[94] Press Release (2012-04-04), *SugarCRM completes $33 million financing round for further expansion into the enterprise*. Retrieved Sept. 18, 2012 from http://www.sugarcrm.com/newspress/sugarcrm-completes-33-million-financing-round-further-expansion-enterprise

[95] Official SugarCRM Community web page: http://www.sugarcrm.com/community. Retrieved Sept. 18, 2012.
Note: Registered developers are a bad indicator for community vitality, as it does not say anything about actual activity of these developers.

4.1.1 Technical foundation

SugarCRM builds on the LAMP solution stack – a widely used open source software bundle consisting of Linux (operating system), Apache (web server), MySQL (database management system) and PHP (server-side scripting language). According to John Roberts[96] (co-founder and CEO until 2009), the choice of LAMP was based on a "conscious decision […] because we felt it was the least complex, most portable, and most cost-efficient platform to run a CRM application on" (ibid). Likewise, as of version 4.5, SugarCRM integrates the open source YUI[97] JavaScript in its distributions. As such, SugarCRM should not only be recognized as a commercial open source project maintainer, but also as an OSS systems integrator and direct beneficiary of a wider open source ecosystem – not the least because of low barriers to entry and a familiar technological OSS environment for potential community members.

4.1.2 Deployment

SugarCRM allows for four different deployment and hosting options to customers:

1. On-premise: The customer uses own hardware and network resources, and deploys the software stack behind a firewall. This gives more control and flexibility for integration with legacy software, but raises costs for infrastructure, integration and maintenance.

2. On-demand, "SugarCRM cloud": Hosed by SugarCRM based on a predefined Service Level Agreement (SLA), backup and secure FTP access. This Software as a Service (SaaS) option provides the inverse benefits and drawbacks as the above option.

3. On-demand, SugarCRM partners: Similar to the above option, but with a partner of SugarCRM. Terms and conditions may vary between providers.

4. Public Cloud: This is a compromise between the first and two latter options. The SugarCRM stack can be deployed on public (commercial) clouds, such as Amazon Elastic Compute Cloud (EC2), Windows Azure, Rackspace Cloud or IBM SmartCloud.

Users of the Sugar Community Edition (CE) are likely to fall into the on-premise category.

[96] It was named "Project of the Month" on SourceForge in October 2004, see http://sourceforge.net/potm/potm-2004-10.php

[97] Yahoo! User Interface Library, which uses the BSD (permissive) license.

4.1.3 SugarCRM's legal framework

The legal framework of SugarCRM is quite typical for that of COSS firms (see chapter 2.4.1). Yet, in some respects, the company has broken the mold, with new innovative ways to license its open source version – the Sugar Community Edition (CE)[98]. Although other COSS firms have followed its example, SugarCRM has gained particular attention for its lead role.

4.1.3.1 Exploring troubled waters

SugarCRM pioneered the controversial "Exhibit B" clause, later imitated by other COSS companies. It entails a potent modification of the Mozilla Public License 1.1 (MPL 1.1), which requires all derivate work to conspicuously display a specified logo in all interactive user interfaces[99]. While proponents defend this as a measure to assure attribution to original authors[100], opponents denote it as a "badgeware" clause that is in clear breach of fundamental FOSS principles. Apparently, so the latter camp argues, the primary rationale behind this provision is not attribution, but to advertise for the commercial product, prevent forking and inhibit external reuse of code. As a result, SugarCRM received considerable critique and bad publicity for claiming to be open source, while its altered MPL license had not even been approved by the OSI[101]:

> *"it is the first time we have [..] to confront agents who fly our flag as their actions serve to corrupt our movement. The time has come to bring the matter into the open, and to let the democratic light of the open source community illuminate for all of us the proper answer."*

Thus, SugarCRM was hailed as a hero when, in 2007, it announced the release of the Community Edition 5.0 under the GPLv3[102]:

> *SugarCRM has shot right past the badgeware debate by embracing GPL V3. This updated version of the GPL has been portrayed by open source rivals as a threat against business – gwana, gwana, gwana. Now, we find a leading software maker going whole hog with the license.*

---

[98] Previously known as Sugar Open Source.
[99] In this case saying "Powered by SugarCRM", with a hyperlink to the company's website.
[100] Note: The "Software as a Service" (SaaS) delivery model allows hosted software to be commercially used without offering access to the underlying code - the so-called SaaS "loophole", also known as the application service provider (ASP) loophole. The "Exhibit B" clause closes this loophole.
[101] E.g. Michael Tiemann (2007-06-21), *Will The Real Open Source CRM Please Stand Up?* Retrieved Sept. 18, 2012 from the blog of Michael Tiemann (then president of the OSI): http://opensource.org/node/163
[102] Ashlee Vance (2007-07-25), *SugarCRM trades badgeware for GPL 3: A hero is born*. Retrieved Sept. 18, 2012 from http://www.theregister.co.uk/2007/07/25/sugarcrm_gpl3/

But the euphoria soon settled, as it became clear that SugarCRM had merely translated its "Exhibit B" clause into a different legal shape. As one disgruntled contributor to a mailing list noted to general acclamation[103]:

> *I personally consider most licence requirements for 'visual display of OSS attributions' to be at minimum a bit obnoxious […]. SugarCRM's subsequent torturing of GPLv3 into a particularly bad badgeware licence through stitching their earlier badgeware requirement into GPLv3 via a hook in GPLv3 section 7 -- covering SugarCRM Community Edition 5.0 -- was IMO outright deceptive, particularly the way they ballyhooed that move in public pronouncements, implying they were using stock GPLv3 when that was not the case […].*

After outcries of foul play, the company in 2010 moved on to the AGPLv3 license[104] with SugarCRM CE 6.0. But the new version stirred up just as much controversy, albeit for a different reason: New major features (notably the user interface) were withheld from the Community Edition, thereby widening the functionality gap to commercial editions[105]. Now SugarCRM was accused of flirting with the newly termed "open core" business model[106], which had received extensive negative attention among FOSS supporters. SugarCRM refuted the accusations, pointing out that all (even commercial) editions contain the entire source code[107], and stated that "[o]pen source doesn't mean free and was never really meant to mean free"[108]. Although indisputably correct, this argument somewhat misses the point by ignoring key aspects of the Open Source Definition – the rights to modify the relevant code and distribute derivative work. Thus, the enduring disagreement seems to circle around the very definition and use of the term "open source". According to an anonymous blog comment[109]:

---

[103] Rick Moen (2012-02-01), message to the License-discuss mailing list (with minor corrections), Retrieved Sept. 18, 2012 from http://projects.opensource.org/pipermail/license-discuss/2012-February/000180.html

[104] The AGPL (Affero GPL) license is specifically designed to fill the above mentioned SaaS / ASP loophole.

[105] The move also spurred rumors among community members that SugarCRM would put less effort in the CE edition, or indeed be orphan it. See http://forums.sugarcrm.com/f22/sugarcrm-ce-being-left-out-loop-65740/

[106] "Open core" is a relatively new and ambiguous term. As often used, it only subtly differs from (or is a subset of) a COSS / dual licensing model. It puts more emphasis on a feature limited "basic" open source version (strongly implying "crippleware") with optional additional commercial features.

[107] Note: The source code of SugarCRM is written in PHP – an interpreted language that is inherently unsuited for compilation into binary code (although some recent PHP compilers have had limited success).

[108] Martin Schneider, senior director of communications at SugarCRM, cited from InternetNews.com in article by S.M. Kerner (2010-07-13), *SugarCRM 6 Debuts with Open Source and Commercial Features*, retrieved Sept. 22, 2012 from http://www.enterpriseappstoday.com/article.php/3892731

[109] Comment to article by S. Dean (2010-07-14), *Has SugarCRM Violated Open Source Principles?* Retrieved Sept. 22, 2012 from http://ostatic.com/blog/has-sugarcrm-violated-open-source-principles

> *As far as I can tell, the complaints are not about SugarCRM's business strategy or distribution terms, but only around their insistence on using the term "open source" to describe what they're doing. That's what the controversy is about: the dilution of a term with a very specific and well-understood meaning. How SugarCRM generates their revenue is not at issue. Accuracy in labeling is.*

In this context, the OSI's formal criteria of "open source" licenses – which Sugar CE appears to satisfy – seem secondary to normative perceptions of what they *should* entail. In an interview[110] on the occasion of the official 10th anniversary of the OSS movement, Bruce Perens[111] expressed regret of only one thing: That the OSI's license ratification process had not been implemented with clearer guidelines. According to him, the proliferation in OSI-approved licenses, and particularly the emergence of attribution ("badgeware") licenses, is "dangerous" and puts open source on a "slippery slope". Already before SugarCRM migrated to the GPL, he is to have said "that the minute a license becomes so prescriptive about what developers can and cannot do with a software's user interface, it's no longer open source because of how one of open source's most fundamental tenets is the developers' freedom to control the user interface"[112]. Similarly, Eben Moglen, general counsel to the FSF, "agreed with other critics that the requirements limit developer freedom and that developer freedom is one of the things that free software is all about" (loc. cit). Apparently, they are not alone in such thinking. Some FOSS pundits are already exploring new legal instruments to counter what they consider commercially compromised licenses[113]. Meanwhile, SugarCRM stands its ground and remains in the limelight of critics.

[110] G. Clarke (2008-02-11), *Perens: 'Badgeware' threat to open source's next decade.* Retrieved Sept. 22, 2012 from http://www.theregister.co.uk/2008/02/11/open_source_at_ten/

[111] Author of the Open Source Definition and co-founder of the OSI

[112] David Berlind (2006-12-06), *Podcast: SugarCRM CEO says attribution in open source licenses is about fairness*, Retrieved Sept. 22, 2012 from http://www.zdnet.com/blog/berlind/podcast-sugarcrm-ceo-says-attribution-in-open-source-licenses-is-about-fairness/211

[113] E.g. "GPL.next" a fork of GPLv3 initiated by Richard Fontana (co-author of the original GPLv3).

4.1.3.2 IP modularity

A recent study (Waltl, Henkel, et al., 2012, p.95) confirms that SugarCRM faces "conflicting goals of openness toward users and complement developers for the purpose of distributed value creation, and of maintaining exclusivity of essential modules in order to appropriate value". IP modularity is therefore of "highest strategic relevance for SugarCRM and its ecosystem". In the words of Clint Oram, co-founder and CTO of the firm (loc. cit.):

> *I would say it's one the most fundamental aspects of our entire business model strategy that we purposely keep the modularity in such a way that we can easily create different [open source and proprietary] editions. What we sell is based on IP modularity.*

Not surprisingly then, the study finds that the platform architecture and business model are inherently interrelated: "The business model is enabled by a product architecture that separates the IP elements for the community edition from the elements required for the commercial editions" (ibid. pp. 98-99)[114]. SugarCRM's basic product architecture is depicted in a simplified schematic as follows[115]:

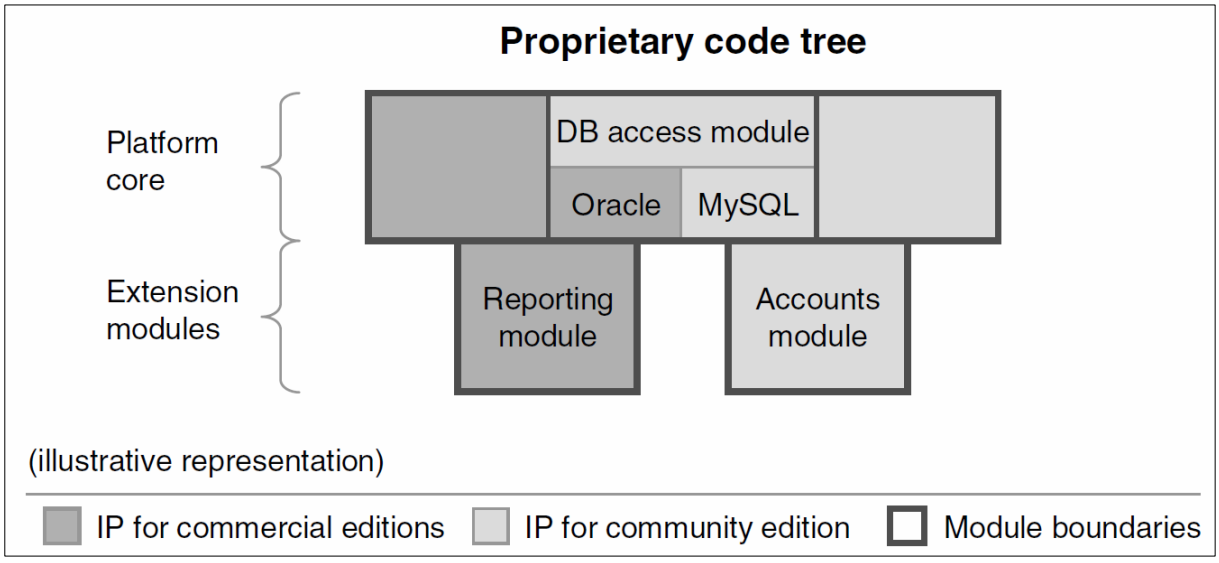


Figure 3: Schematic Architecture Overview for SugarCRM

Interestingly, the figure implies code IP heterogeneity even for the platform core. SugarCRM has indeed some core components exclusively dedicated for proprietary distributions. This increases complexity and may pose a risk of code incompatibility. SugarCRM ensures compatibility by using the same code tree for all distributions, and through the build process

[114] Note: Commercial editions are governed by the , SugarCRM Master Subscription Agreement. See http://www.sugarcrm.com/master-subscription-agreement

[115] Fig. 1 in ibid. p. 99: *Schematic Architecture Overview*

segregating the code according to preset tags. And while such granular IP modularity rises development costs, it is justified by an increased "strategic control of value appropriation" (ibid p. 104). The community is not included in this process. It has access to the Sugar CE code[116], but almost exclusively works on optional extension modules, language packs and themes. The legal terms for these peripheral components may vary, depending on what the authors choose[117]. To the limited degree that community members actually contribute to core functionality (mostly bug fixes), they are first required to sign the *SugarCRM Contributor Agreement*[118]. This leads us to the subject of project governance.

[116] Hosted on GitHub: https://github.com/sugarcrm/sugarcrm_dev
[117] According to John Mertic, Community Manager at SugarCRM
[118] Which states that "You assign all right, title and interest worldwide in all patents, inventions copyrights and related moral rights ("IP Rights") for the full term of their existence in and to Your Contributions to SugarCRM"

4.1.4 Governance

The SugarCRM community is governed as a commercial "benevolent dictatorship", wherein the corporate sponsor wields exclusive control over the code base, the project's evolution strategy, and the implementation roadmap. Since the community in essence is relegated to peripheral extension modules, it does not naturally evolve around a shared code base (the platform core), as is the case in conventional OSS projects[119]. On the contrary, the SugarCRM community appears to be a fragmented cluster of entities, held together by the gravitational pull of web forums[120] and a distribution platform for complementary software modules (*SugarForge**[121]). Many modules found here cannot be considered as open source, since authors may share them under proprietary terms[122]. Moreover, there is little evidence to suggest that collaborative development on these peripherals is the norm. Even if registered SugarForge members may "request to join" projects of others, all sampled projects were provided by only one single entity (mostly commercial). As such, the SugerCRM community does not seem to engage in Bazaar style collaborative development – a guiding principle of OSS.

To reward and incentivize partners who engage in the community, SugarCRM in late 2010 started its "Open+ Developer Program". As members, they can "gain access to emerging SugarCRM features and the ability to provide feedback earlier in the development cycle"[123]. Although the role of corporate participation in COSS projects is "a consolidated and generally accepted fact" (Capra, Francalanci, et al.: 2009, p.225), this move is emblematic of a strictly controlled development process that evidently tilts in favor of commercial

[119] As of August 2012, about two thirds into the year, SugarCRM had registered less than 80 code contributions for the platform core (including bug fixes) – which was hailed as nearly the double rate from the previous year. An overwhelming majority of these contributions were made by commercial partners (or employees thereof). See SugarCRM Developer Blog (2012-08-10), *Calling out some top contributors in the community*, Retrieved Sept. 28, 2012 from http://developers.sugarcrm.com/wordpress/2012/08/10/calling-out-some-top-contributors-in-the-community/

[120] See http://forums.sugarcrm.com/. Help forums attract most activity.

[121] SugarForge (http://www.sugarforge.org/) serves as SugarCRM's community portal, where optional extension modules (i.e. complementaries such as language packs, skins and add-ons) are shared (for both Sugar CE and commercial editions). Also see Appendix A. Although marketed as a "factory", there is little evidence of collaborative work on SugarForge.

[122] See Terms & Conditions ("SugarCRM On-Line Terms Of Use") required to be approved for registration on SugarForge: http://www.sugarcrm.com/page/sugarcrm-line-terms-use/en . Of the sampled projects 20% (2) explicitly stated to be of "Other / Proprietary" nature.

[123] Poer, M. (2011-07-11), *Profiling Solutions joins SugarCRM Open+ Developer Program*, Retrieved Sept. 28, 2012 from http://www.profilingsolutions.com/archive/profiling-solutions-joins-sugarcrm-open-developer-program/

affiliation. Governance thus gets condensed to a series of entrenched institutionalized sluices, over which independent community members have little real influence. This observation is substantiated by the already mentioned study by Capra and Wasserman (2008), which identifies SugarCRM as the only "fully closed" project when measured against all the four "fundamental governance dimensions"[124]. See figure:[125]

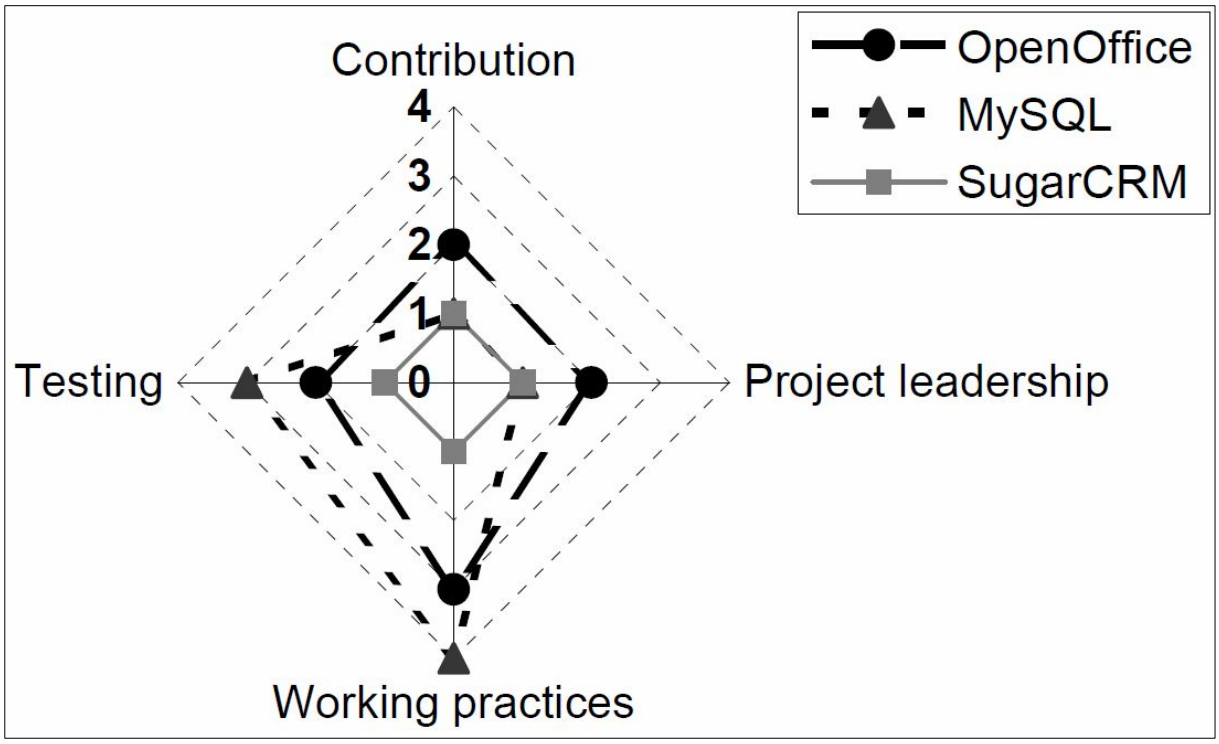


Figure 4: Visualization of comparative assessment of four COSS projects' governance model

As the study concludes (ibid. p.12, my emphasis): "SugarCRM governance and managerial styles are actually very similar to those of a traditional closed software project, with the only exceptions that most of the code is open and that external people *can* contribute code". Naturally, such observations provide ammunition to critics who already question SugarCRM's open source credentials. This does not make Sugar CE less OSI-compliant in formal terms. But it may create false expectations among those that share a more conventional understanding of the term "open source", thus spurring mistrust and notions of unfairness.

[124] I.e. as stated in chapter 2.4.4: (1) Contribution (relative amount of voluntary code development), (2) Project leadership (distributed vs. hierarchical decision making), (3) working practices (geographically distributed and virtual communication vs. physical proximity and face-to-face interaction), and (4) testing (community peer-review vs. formalized "in-house" quality assurance of code contributions – i.e. a powerful gatekeeper role). My observations and interviews confirm that these findings are still valid.

[125] Figure 2 in ibid: *Graphical representation of the assessment of a project according to the framework*.

4.2 Stakeholders and their incentives for participation

The parties involved in the development of SugarCRM software fall into one or several of four main categories:

1. SugarCRM employees: These are hired software developers whose primary focus is on the AGPLv3-lincensed platform core and proprietary modules for the commercial product range. SugarCRM employees are also encouraged to participate in community forums, where they mainly answer technical questions. Naturally, this category directly involves the CTO[126] and community manager[127], and by implication also shareholders and the remaining C-suite.

2. Affiliates: SugarCRM has several standardized partner programs[128]. Many companies also have an implicit affiliation with SugarCRM[129] and use SugarForge to distribute extension modules. According to the Community Manager, many companies "start in the SugarForge community, and as their practice grows and matures they will look to be part of the Partner programs" (which also give access to *SugarExchange**). This category is dominated by VARs and ISVs that provide complementary functionality.

3. Customers: Customers of SugarCRM and its affiliates (e.g. resellers) play a more peripheral role in the development process. But particularly in the case of On-premise deployment, customers need to relate to SugarCRM code for integration and customization. They are well represented in forums, where they post questions and receive guidance. A special case is IBM, to which SugarCRM also relates through a global partnership program.

4. Sugar CE users: The open source edition is downloaded by both individuals and institutions (SMEs, start-ups and non-profits), many of which later become paying customers in order to get access to more advanced features (e.g. team management and mobile support). Sugar CE users make use of extension modules from SugarForge, and occasionally contribute own modules.

[126] Clint Oram (CTO and co-founder), who leads product strategy and product management.
[127] John Mertic (Community Manager since 2011), who is an authority on PHP and open source CRM. He coordinates external contributions with internal development and represents SugarCRM at OSS conferences.
[128] See Appendix E.
[129] E.g. third party software providers, who integrate their software with SugarCRM.

Members of all stakeholder categories are well represented in community forums. Beyond that, participation focus in the development varies (see Table 2):

	SugarCRM employees	Affiliates	Customers	Sugar CE users
Predominant work ethic / incentive to contribute	Salary-based	Salary-based	Salary-based	Mixed salary based and voluntary
Primary contributions to the development process	Platform core, proprietary module extensions, bug r.+f.	Proprietary and OSS module extensions. Bug r.+f.	Bug r.	Bug r.+f., OSS extensions

Table 2: Stakeholders incentives and development focus (r = reporting, f = fixing).

The survey results clearly identified "need / wish for software improvements" as the overall principal reason for contributing. Other dominating motivational factors are "the widespread use of the software"[130] and "attractiveness of the software itself". Interestingly, individuals who are also active in other open source projects[131] rank the motivational factors differently for their *general* participation in OSS projects than they do for their SugarCRM participation[132]. Here, and particularly amongst those who use Sugar CE personally[133], motivational factors associated with hacker culture play a somewhat larger role[134]. This highlights a distinction between dispositional and contextual motivation – i.e. the same individual with given motivational dispositions may "play out" different forms of motivation in different contexts (Oreg and Nov, 2008). Or seen from a different perspective: An unfavorable context may dampen enjoyment-driven enthusiasm (Stewart and Gosain 2006). Both interpretations seem plausible in the case of SugarCRM.

Overall, we can conclude that the stakeholders' incentives of cooperation in the SugarCRM development process are of an utilitarian, commercially-oriented nature. Even among a minority of purely voluntary contributors[135] motivational factors are clearly tilted towards the utility spectrum. This substantiates qualitative findings by Shah (2006), which predict such a trend for gated OSS communities.

[130] A proxy for networks effects based on community- and user pool size (i.e. factors that increase the value of complements and indicate the overall health of the project).
[131] Among those that also participate in other OSS projects (57% of all respondents), Sugar CE users are clearly overrepresented.
[132] I.e. including OSS projects besides SugarCRM.
[133] Note: About half of those using Sugar CE personally are also employed by SugarCRM, one of its affiliates, customers and / or an institutions using Sugar CE.
[134] Fun, commitment to the community, reciprocity, meet like-minded people and community spirit.
[135] I.e. those not receiving pecuniary rewards for contributions.

4.3 Strategies

SugarCRM employs a number of strategies to safeguard shareholder interests. This subchapter explores how these align with the needs of other stakeholders within the COSS ecosystem. It starts off with SugarCRM's overarching market orientation, which becomes defining for the relative strategic relevance of different community constituencies. Second, it turns to a number of seemingly unproblematic measures and technological enhancers that SugarCRM employs to spur innovation, increase productivity and rouse a shared community spirit. This includes ancillary tools and platforms, which empower all parties in the development process. Finally, it revisits the legal framework of SugarCRM, before it closes with a discussion of general implications.

4.3.1 Market and sales orientation by proxy

Larry Augustin[136], CEO of SugarCRM since 2009, believes that "SugarCRM's product is a process sale, not a technology sale".[137] This has lent value-added resellers (VARs) an increasingly strategic role. "The traditional theory", so it goes, is that the open source model bypasses VARs "since customers are able to download and try out free software directly; if they require support or customisation, they would contact the software company directly" (loc. cit.). But SugarCRM cannot possibly satisfy heterogeneous demands from diverse vertical markets simultaneously (Brownstein et. al., 2006). VARs, on the other hand, have intimate knowledge of their market segments. And as SugarCRM source code is readily available to them, they stand in an ideal position to offer tailored support, customizations and "narrow" complementary functionality. The comprehension of this fact probably contributed to the establishment of the mentioned *SugarCRM Open+ Developer Program*[138], "to provide more collaboration and a tighter feedback loop with customers and the Sugar business partner community that writes add-ons"[139]. On a nonspecific "getting started" page for community developers, the program is advertised specifically to partner employees[140]:

---

[136] Larry Augustin is also known as one of the contributing theorists behind the Open Source movement, prolific OSS entrepreneur and venture capitalist.

[137] Moody, G. (2010-10-10), *(Finally) Meeting Mr. Open Source Business*, Blog entry retrieved Oct. 10, 2012 from http://blogs.computerworlduk.com/open-enterprise/2010/10/finally-meeting-mr-open-source-business/

[138] See http://www.sugarcrm.com/partner/sugarcrm-open-partner-program. The page states that "As a channel focused company, SugarCRM is 100% committed to our partner's success". Retrieved Oct. 10, 2012.

[139] Woody, A. (2011-11-08), *SugarCRM Bolsters Development, Admin Features with Version 6.3*, Retrieved Oct. 10, 2012 from http://www.itjungle.com/fhs/fhs110811-printer09.html

[140] Retrieved Oct. 10, 2012 from http://support.sugarcrm.com/01_Get_Started/03_Developers/03_Community

If you work for a partner of SugarCRM, then we highly encourage you to consider becoming part of the Open+ program. Being an Open+ developer gives you the inside track on upcoming features and designs for Sugar with the added benefit of allowing you to contribute to the success of the product. In addition, you will engage with the SugarCRM engineering team to help guide you on complex customizations and integrations that you deliver for your customers.

Apparently, SugarCRM goes a long way to align itself with the needs of its partners and customers, and reserves an exclusive position to its affiliates in the developer community. In the same spirit, it holds annual conferences (*SugarCon*), specifically targeted at employees of partners and third party affiliates[141]. Any corresponding push to accommodate voluntary contributors could not be observed. Of course, most of the Open+ program's offerings would not be suitable to Sugar CE developers – nor would the annual conferences, since they are primarily relevant for commercial editions. And it seems nothing but reasonable, from a commercial perspective, that SugarCRM prioritizes its revenue generating editions and sales channels over Sugar CE. Further, one can plausibly argue that the community edition directly benefits from the commercial ones, as the improvement of the latter also relate to the shared platform core and common basic modules[142]. Nevertheless, even if there is no official segmentation of the community, the sum of preferential affiliate initiatives amount to a de facto layering of it. In ANT terminology, they constitute a powerful translation of commercially guided interests into organizational inscriptions, which in turn influence the nature of the community. Whether this amounts to a conscious strategy to steer the community towards commercial hegemony or homogeneity could not be empirically established. But it can be assumed that independent volunteers may perceive this as procedural and / or distributional unfairness (see chapter 2.4.4)[143].

The pertinent question is whether SugarCRM would have anything to gain from diminishing the role of its voluntary CE contributors. The company benefits from a sizable CE constituency to uphold its open source credentials, but its relative size may not be important in this regard. Then there is the implicit danger of OSS forks deriving from independent

[141] In 2012 held at the Waldorf Astoria, New York, with an agenda that "provides strategic and practical insight on how to engage your entire business to effectively and consistently interact with your customers." Retrieved Oct. 11, 2012 from http://sugarcon.sugarcrm.com/ (under "about" tab).

[142] According to the Community Manager (interview), the "top end Ultimate edition has around 80% code commonality with the CE edition".

[143] Note: In this context, it is not as much a question of differentiation, as it is of subjective perceptions of a viable and fair balance between volunteers and commercially employed developers.

community members, which one may construe as less likely if the ranks of volunteers dwindle. This is an intriguing line of thought, but again, neither possible to confirm nor disprove as an actual managerial doctrine. Above all: Keeping an independent Sugar CE community constituency alive and well seems intuitively rational, since it represents a valuable business asset. Yet the seeming shortage of unambiguous measures to reaffirm SugarCRM's commitment to CE developers is bound to raise questions. One appropriate measure could be a formalized "covenant" in exchange of copyright attribution (as Perens suggests, see chapter 2.4.4). Another to allow the community more influence or slack in the governance of the Sugar CE (see chapter 4.1.4).

4.3.2 Win-win

Preferential treatment aside, SugarCRM also has a palette of offerings that are accessible to the wider community ecosystem. For example, it holds regular "code sprints", to which CE community *may* also get invited[144]. These appear to be popular happenings, which not only provide developers an opportunity to meet like-minded peers face-to-face and take on stimulating challenges, but also spur innovative enhancements that benefit SugarCRM, partners and users[145]. Another noticeable initiative is the so-called "Project of the Month" (POTM)[146], similar to that of SourceForge. It allows registered SugarForge developers to nominate candidates and vote for "the most outstanding and innovative add-ons, plugins, and extensions". This is likely to foster peer recognition, which scholars link to motivational factors like ego gratification, self-marketing and community identification (Hars and Shaosong, 2001; Lerner and Tirole, 2005).

To empower the development process, SugarCRM offers and promotes a number of technical enhancers. Besides the provision of online documentation, hosting of code[147], discussion forums, a bug tracker[148] and generic tools (e.g. PHPUnit for unit testing), it also offers tools that are custom made for SugarCRM development[149]:

---

[144] John Mertic (2012-03-02), *Code sprinting in Raleigh = success*, SugarCRM Developer blog entry, retrieved Oct. 11, 2012 from http://developers.sugarcrm.com/wordpress/2012/03/02/code-sprinting-in-raleigh-success/

[145] Sonja Fridell (2012-01-23), *SugarCRM Code Sprint, An Engineer's Point of View*, Retrieved Oct. 11, 2012 from http://www.brainsell.net/blog/2012/01/sugarcrm-code-sprint-an-engineers-point-of-view/

[146] See http://www.sugarforge.org/content/project-of-the-month/

[147] GitHub for the CE core, and SugarForge for modular community extensions

[148] See http://www.sugarcrm.com/support/bugs.html

[149] Other tools, like MeterMaid and SugarMMM, have been discontinued according to the Community Manager (interview), but are still advertised in the *Sugar Developer Zone* as of early October 2012.

- *SODA* and *SodaMachine* for functional testing
- *TidBit* to generate large amounts of data for scalability and performance testing

Although many regard these offerings as adequate, some survey respondents like to point out specific issues that leave room for improvement – the use of Git for modular extensions being the most recurring issue, e.g. [150](minor typographic and grammatical corrections):

> *Git availability on SugarForge is something that I've been expecting for a long time, it would definitely ease the process of project submission. And considering it also being used by Sugar to manage the core product and that it's the most used version control system nowadays, I believe it would make sense. Exploring the possibility of "hosting" remote projects (github, google code, etc) would also avoid people having the same project on two different places.*

Other issues concern a lack of proper documentation of "internals", the professed absence of a quality control scheme, more advanced search functionality, a way to easily test modules prior to installation, the archiving of outdated / unsupported modules[151], and a more intuitive organization of modular extensions. Given the importance of satisfactory developer toolkits (see chapter 2.3.4), SugarCRM is well advised to consider options that address these professed shortcomings.

There are some noticeable outliers who used the opportunity to voice blatant criticism of, for example, SugarForge in general ("abandoned site!"[152]) and the Community Manager (should do "some work instead of plugging the commercial editions"[153]). Even if overly tendentious and unjustified, such strong negative sentiments raise concerns over potentially wider discontent among Sugar CE developers. Unfortunately, the survey did not include any questions to explore this issue in further detail. Regardless, SugarCRM could benefit from commissioning an authoritative unpartisan third-party assessment to dispel misgivings and identify potential measures to overcome underlying causes. Along the same lines, a census could establish the actual number of *active* developers[154] within the different community constituencies. If the results were encouraging, they could then be used by SugarCRM to parade a harmonious relationship with a vibrant community.

[150] Respondent ID: 3389715 (from Portugal, partner employee and personal user of the Sugar CE).
[151] Which my observations confirm to represent a large share of SugarForge extensions.
[152] Respondent ID: 3424602 (from Costa Rica, employed by institution using Sugar CE, and also personal user of Sugar CE).
[153] Respondent ID: 3390324 (from Chile, personal user of the Sugar CE).
[154] In contrast to all *registered* developers on SugarForge, the number of which is used for marketing purposes.

4.3.3 Legal assurances

SugarCRM employs the AGPLv3[155] license for its CE edition. In legal terms, Sugar CE can thus be considered as *free software*[156]. Yet, as earlier explained, the firm's consolidated copyright ownership allows it to reemploy the same code under commercial terms. Through a deliberate IP-modular product architecture SugarCRM thus achieves strategic product differentiation and price discrimination. Meanwhile, the AGPL license effectively deters commercial competitors from reusing code from the Sugar CE. Interestingly, though, SugarCRM does not appear to consider the generic AGPL terms as sufficient – it apparently found it necessary to apply far-reaching augmentations to safeguard its commercial interests (see Appendix D). Of these, one deserves particular attention – the requirement of user interfaces of derivate work to display the "Powered by SugarCRM" logo as part of "Appropriate Legal Notices":

> *The interactive user interfaces in modified source and object code versions of this program must display Appropriate Legal Notices, as required under Section 5 of the GNU Affero General Public License version 3.*
> *In accordance with Section 7(b) of the GNU Affero General Public License version 3, these Appropriate Legal Notices must retain the display of the "Powered by SugarCRM" logo. If the display of the logo is not reasonably feasible for technical reasons, the Appropriate Legal Notices must display the words "Powered by SugarCRM".*

This corresponds with "attribution" provisions from earlier versions (see chapter 4.1.3.1), and remains equally controversial today[157]. As one contributor to the *legal-debian* mailing list puts it[158]:

> *I think this goes beyond (A)GPLv3 §7(b): "Powered by SugarCRM" is neither an Appropriate Legal Notice (defined in the AGPLv3 as a copyright notice, statement of no warranty, or statement that it's under the AGPLv3) nor an author attribution (author attributions look like this: "based on software written by John Doe, Jane Smith and FooCorp, Inc."). It's also not necessarily true, or advantageous to the authors of SugarCRM. If I extracted a module from SugarCRM to use in my new webapp, it looks as though the authors of SugarCRM are trying to require me to say*

[155] The *GNU Affero General Public License*, version 3. See http://www.gnu.org/licenses/agpl-3.0.html
[156] As additional terms explicitly state: "This program is free software" (see Appendix D).
[157] Making Sugar CE to fit into the category of what is popularly known as "badgeware".
[158] Simon McVittie (2011-12-14), Subject: *Re: Thoughts on GPL's Appropriate Legal Notices? or the CPAL?* Retrieved Oct. 14, 2012 from http://lists.debian.org/debian-legal/2011/12/msg00034.html

"Powered by SugarCRM" - even if the rest of my webapp is so bad that doing so could damage SugarCRM's reputation! (By contrast, "incorporates code from SugarCRM" would remain a true fact, and is nicely neutral: a non-binding request to acknowledge use of SugarCRM is much less adversarial and probably has about the same practical effect.)

Reasonably enough, this argument recognizes a potentially detrimental boomerang effect. But it also implies a likely dissuading effect on partial or wholesale (fork) reuse of code, since a mandatory "Powered by SugarCRM" logo (henceforth *The Logo*) on "interactive user interfaces"[159] could deteriorate the appeal of any resulting software. Apart from branding, the prevention of external code reuse is therefore a probable rationale behind the scheme. This seems all the more likely if we consider the relevance of SugarCRM's Trademark Policy (TP)[160]: Among several "SugarCRM Marks", it identifies *The Logo* as the "SugarCRM Conditional Use Logo", and explicitly threatens with legal action for unauthorized or "detrimental" use. As such, it actually preempts the above noted detrimental boomerang effects. But it also poses a catch-22-like paradox with Kafkaesque connotations to would-be adopters of Sugar CE code:

- Premise 1, *axiom*: Sugar CE is licensed under the AGPLv3, which is "intended to guarantee your freedom to share and change" any code that is governed by it.
- Premise 2: The use of *The Logo* is mandated by SugarCRM additional terms, in professed accordance with "Appropriate Legal Notices" of the AGPLv3 (Section 5 and 7b).
- Premise 3 (TP, Section 1A): Under certain conditions, *The Logo* may be "used without specific written permission from SugarCRM". Amongst others, these conditions prohibit "detrimental" use, and require compliancy with Section 2 of the TP. But "SugarCRM reserves the right to revoke this authorization at any time in its sole discretion for any reason" – for example, if SugarCRM deems the use as detrimental.
- Premise 4 (TP, Section 2A): "Unauthorized use of the SugarCRM Marks or marks that are confusingly similar thereto constitutes an infringement of SugarCRM's trademark rights and is strictly prohibited".
- Paradoxical conclusion: The "freedoms" that the AGPLv3 is supposed to guarantee do not appear to apply to Sugar CE – although it actually is governed by the AGPLv3.

[159] Which in effect refers to all interfaces of the software, since CRM systems are throughout interactive.
[160] See http://www.sugarcrm.com/about/trademark-information (Last modified: 24 February 2012). Retrieved Oct. 14, 2012

Since the use of *The Logo* is required, but its use simultaneously rests on authorization, a discretional revocation of authorization could effectively render an entire derived software distribution useless. Further, the TP (Section 1A) threatens with intimidating reprimands:

> *Upon revocation of this authorization by SugarCRM, you shall immediately cease using any and all SugarCRM "Conditional Use" Logo [sic]. If you do not immediately cease using all SugarCRM "Conditional Use" Logos upon revocation [..], SugarCRM may take whatever action it deems necessary to protect its rights and interests, including but not limited to seeking injunctive relief, actual damages, disgorgement of profits, trebling of any monetary award and attorneys' fees.*

Unsurprisingly, SugarCRM has experienced few forking attempts after it first introduced such a scheme into its legal framework. Interestingly, it was "compelled" to take this step in late 2004[161], shortly after vTiger CRM appeared as a SugarCRM fork[162]. The fork was instigated under a confrontational "honest open source" banner, just as *SugarCRM was starting to "close" some of its source code for commercial gain* (Rossi, 2011). A short but fierce verbal clash ensued. SugarCRM denounced vTiger as "a lie"[163] and accused it of hijacking SugarCRM's "hard work (code)" by replacing its logos with vTiger's[164]. Meanwhile, vTiger lectured SugarCRM on the meaning of open source, and appealed to Eric Raymond and the Open Source Development Labs for support. The dispute silently settled in vTiger's favor. In a historical light, SugarCRM's consistent application of attribution ("badgeware") schemes thus appears as a strategic safeguarding measure against further forking attempts. But it remains to be seen whether the latest legal incarnation is watertight. Given its paradoxical inconsistencies, one cannot help but wonder whether...

1) the mandated use of the "Powered by SugarCRM" logo in interactive user interfaces has a lacking legal basis in the AGPLv3,
2) or SugarCRM has found a legal loophole in the AGPLv3, or the FSF actually intended this form of use (the latter would seem to be in conflict with its stated intentions of freedoms),
3) and / or that the innovative coupling to the Trademark Policy allows SugarCRM to circumvents the freedoms that the AGPLv3 is intended to guarantee.

[161] John Roberts – then CEO of SugarCRM (2006-11-27), message sent to *license-discuss* mailing list of opensource.org. Subject: *ZDNet article - why attribution matters*. Retrieved Oct. 14, 2012 from http://lwn.net/Articles/211814/

[162] In 2004, based on version 1.5 of SugarCRM Open Source (the predecessor of Sugar CE), which was governed by a modified version of the Mozilla Public License 1.1 – the SugarCRM Public License (SPL) 1.1.2. The "Powered by SugarCRM" provisions was added to the "Exhibit B" clause as of SPL 1.1.3. Also see chapter 4.1.3.1.

[163] Forum message by John Roberts (2004-08-26): https://forums.vtiger.com/viewtopic.php?t=7

[164] Forum message by John Roberts (2004-08-27): https://forums.vtiger.com/viewtopic.php?t=11

Since none of these alternative explanations have yet been tested in court (to my knowledge), no definite answer is available. If one were to judge merely by popular opinion, as widely expressed in sampled web forums, blogs and mailing lists, alternative 1 would be the winning candidate. Yet, SugarCRM seems to rely on a combination of the other two – or at least on signaling an assertive and deterring intent to defend this legal position. Regrettably, it was not possible to obtain any official statement by the FSF or its affiliated Software Freedom Law Center (SFLC). However, an unofficial statement by Aaron Williamson, Senior Staff Counsel at the SFLC, sheds some light on the issue:

> *I'm afraid that no "authoritative answer" to your question exists; there is sufficient ambiguity that one cannot predict with certainty how a court would rule. I agree with you that there is a potential for conflict of terms where a trademark policy separate from the GPL imposes restrictions beyond what the GPL allows. Where the licensor has retained broad authority to object to "harmful" uses, it's not clear to me that there is a definite conflict; it may depend upon how the licensor enforces the trademark policy. If enforcement was particularly aggressive, i.e. if the licensor objected to a use that was clearly permitted by the GPL, it is possible that a court would apply equitable estoppel, i.e. rule that fairness dictates that the licensor can't enforce against a use its license seems clearly to permit.*
>
> *Perhaps more importantly, the GPL itself includes a "safety valve" to prevent the use of trademarks to unduly restrict GPL licensees: "If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term." In other words, if a licensor adds a restriction that does not fall into one of the categories of permissible restrictions enumerated in Section 7, it is moot.*

In this light, SugarCRM's ostensible position seems vulnerable to attack. Hence, the firm is likely to benefit from a protracted state of legal ambiguity. For, coupled with offsetting costs and the potential ramifications of a trial (see previous page), one can reasonably expect an assertive legal posture to be just as a deterring. Of the same reason, risk-averse would-be adopters of Sugar CE code[165] could benefit from a conclusive evaluation.

[165] E.g. for use of Sugar CE (or adapted parts thereof) in a SaaS solution. See for example "Rebrand" thread on a SugarCRM forum, which raises the question of limited rebranding, and includes posts by SugarCRM CTO and co-founder, Clint Oram: http://forums.sugarcrm.com/f134/rebrand-63132/. See also http://liveweb.archive.org/http://forums.sugarcrm.com/f134/rebrand-63132/ (Wayback Machine).

If SugarCRM's apparent position were to be confirmed as legally impermeable, the firm would possess a very potent juridical weapon against any form of unsolicited Sugar CE code reuse and redistribution. But this legal interpretation would also imply that SugarCRM was, in effect, in obvious breach of fundamental FOSS principles[166]. If, on the other hand, the more popular interpretation were to prevail, SugarCRM would be in a significantly weaker juridical position to defend itself against FOSS forks. This does not mean, however, that SugarCRM necessarily would become critically vulnerable to competition from potential Sugar CE forks (see following chapter).

[166] As defined in The Free Software Definition (FSD) and Open Source Definition (OSD).

4.4 Synthesis and concluding remarks

This thesis has sought to shed light on the particularities of Commercial Open Source as a phenomenon, exemplified through a case study of SugarCRM. Based on empirical findings and analysis thereof, we are now in the position to revisit the initial research questions from chapter 1.2 and draw our conclusions:

1. Stakeholder motivation: Who are the involved parties in the software development process, and what are their primary incentives for participation?

As identified in chapter 4.2, the parties involved in the development of SugarCRM software fall into one or several of four main categories: 1) SugarCRM employees, 2) Affiliates (partners and third party complement providers), 3) Customers and 4) Sugar CE users. SugarCRM developers focus their concerted effort on the platform core and standard components for the commercial product range. Meanwhile, members of the three latter categories, who by definition constitute the SugarCRM community, are encouraged to contribute customizations and complementary modular functionality[167]. My findings suggest, in agreement with contemporary research on "gated communities", that participation is principally utility-driven. On the individual level, this is reflected in the prevalence of pecuniary incentives through employment[168]. Alternative motives for participation, as typically found amongst "hobbyist" and "hackers" in more conventional FOSS communities, play a correspondingly less significant role in this context[169]. On an institutional level, software utility, vested interests and profit motives serve as inherent incentives for participation[170].

To further incentivize its community constituencies and steer efforts according to own interests, SugarCRM employs a number of strategies – which are addressed by the next research question:

[167] Language packs, themes and various forms of modular extensions (including connectors to third party offerings). These are shared under self-chosen licensing terms on SugarForge and SugarExchange.
[168] Note: A significant share of Sugar CE users are employed by SMEs, start-ups and non-profits etc.
[169] Resulting from ex-ante self-selection and dampened enthusiasm among intrinsically motivated users / developers (see chapter 2.3.1).
[170] Software utility: The direct value of the SugarCRM use, and anticipated marginal value of added functionality (of which the contribution to a wider audience does not impose any significant marginal cost, but can have marketing benefits); Vested interests: Commitment to the SugarCRM platform for marketing purposes and sunk costs in one particular architecture; Profit motives: Expected derived value one can capture as a result of contributing (e.g. potential network effects triggered by contributed connectors to one's own complementary offerings).

2. Strategies: Which strategies do the commercial entity employ to satisfy the stakeholders' expectations and needs, whilst at the same time safeguarding shareholder interests?

SugarCRM pursuits an unmistakable market- and sales orientation, which puts strong emphasis on its commercial affiliates. An extensive partner network serves as sales channels to a variety of market segments, and is actively engaged to provide complementary offerings and customizations. This is achieved through targeted partner programs[171] that forge intimate coordinative links within SugarCRM's value network. Privileged access to SugarCRM's internal development process, along with the general availability of a shared source code, enables partners to address self-identified needs directly and assures that SugarCRM's does not sidetrack from its technical core competency – an open product platform and optional proprietary functionality that facilitates vertical product differentiation and price discrimination. This, in turn, is made possible through a granular IP-modular product architecture, deliberately managed to achieve strategic control over value appropriation and deter open source cannibalization.

To empower the development process and lower the threshold for participation, SugarCRM offers a series of ancillary tools and shared online platforms (see chapter 4.3.2). Although many regard these offerings as adequate, my findings suggest there still remains room for significant improvements. According to interviews with the Community Manager, SugarCRM is aware of at least one of the reported shortcomings[172] and has already included it in its roadmap. Given the importance of satisfactory developer toolkits (see chapter 2.3.4), SugarCRM is well advised to also consider the remainder of professed shortcomings.

Although not officially acknowledged as such, SugarCRM appears to have built an elaborate legal framework to shield itself against unsolicited forking and partial external reuse of Sugar CE code (see chapter 4.3.3). In the absence of an authoritative evaluation of its juridical soundness, the actual strength of this formal inscription remains to be determined. If it were to be proven weaker than it apparently intends to be, then SugarCRM could still rely on other defensive mechanisms against potential forks:

[171] Including the "Open+ Developer Program" and annual SugarCon conference. See also Appendix E.
[172] Git-enabled hosting of projects on SugarForge, or the use of GitHub to host SugarForge projects.

- SugarCRM already has assured itself a pre-emptive competitive advantage through strategic management of its IP-modular product architecture, which entails deliberate withholding of valuable functionality from the Sugar CE.

- SugarCRM holds standard-setting power over its code, including application programming interfaces (APIs)[173], on which extra functionality must rely for compatibility. Combined with a well-established partner network that contributes to an ever growing pool of complementary functionality and services, and a mature user base that benefits therefrom, SugarCRM enjoys control over a two-sided market that spurs network effects and lends it market power.

Yet, as Brownstein et. al. (2006, p.15) notes, SugarCRM forks can arise to serve "different vertical markets." Even with SugarCRM's noticeable market- and sales orientation, with a large partner network to drive customizations and sales, there may still be small but lucrative niche segments that remain underserved. The same could be true for the low-end CRM market, which a no-frills fork could attempt to penetrate by expanding and improving the existing palette of generic open source functionality (corresponding to some of what SugarCRM reserves for its proprietary editions). And even if the generic low-end market is already well saturated and unattractive as it is, the current installed base of Sugar CE could serve as a potential target market in its own right – one in which a FOSS fork could conceivably compete with Sugar CE for users and voluntary talent. Given this hypothetical outlook, and considering an amalgamated relationship with commercially affiliated community constituencies, a seemingly sensible strategy for SugarCRM would be to emasculate an already weak and fragmented constituency of independent community members. But even if this would help to stifle potential forking attempts, the actual implementation of such a strategy could have offsetting side-effects. For one, it is likely to undermine the overall value of SugarCRM software[174] and deprive the firm of a valuable asset[175]. Moreover, it is likely to weaken SugarCRM's residual "open source goodwill" by stripping it of already contested open credentials. This leads us to the last research question.

[173] Comprising its so-called "extension framework" which facilitates "upgrade safe" extensions and adaptations to SugarCRM code.
[174] Based on the assumption that a sizable and healthy community of volunteers increases the value of corresponding software.
[175] Based on the assumption that non-affiliated (Sugar CE) community members actually constitute a valuable asset by virtue of driving adoption, contributing bug reports, bug fixes, modular extensions etc.

3. Friction points: Following from the previous – are there notable friction points between involved stakeholders, which could undermine the viability of SugarCRM's business model? If yes, how does the firm handle them?

There are no noticeable direct friction points between the stakeholders represented in the community (Affiliates, Customers, Sugar CE users). Admittedly, this thesis has not delved into direct bilateral relations between these stakeholders, as the collection of such data would push the thesis' scope beyond what is practically feasible[176]. But it is unlikely that relevant friction points therefore have been overlooked. This is because any direct relations can be expected to be of mundane nature and limited only to a subset of SugarCRM affiliates (e.g. resellers) and their customers. Thus, it seems fruitful first to take a step back and recapitulate the role these stakeholders actually play in SugarCRM's business model.

SugarCRM is not only a software manufacturer and vendor, but also a systems integrator and platform provider. Affiliates, customers and Sugar CE users contribute and derive utility from a mutually beneficial ecosystem that is built by SugarCRM around its open product platform – that is, a multifaceted constellation that in many respects resembles a *value network*[177]. As such, SugarCRM commands a multisided market that spurs both same- and cross-side network-effects[178]. For this to work, as is presupposed by SugarCRM's business model, it must be in all the involved parties' self-interest to be part of the network. From this perspective, the autarkic value of SugarCRM's product range is only relevant to the extent that it channels new members into the network – be it paying customers or voluntary code contributors. Conversely, as one would expect from a product labeled as "open source", the autarkic value should in turn be determined by the quantity and quality of contributions derived from the network. In this sense, open source projects resemble workshops[179], which "mobilize resources from different actors to solve clearly defined problems" (Berdal and Føleide 2008, p.17). This is only partly true for SugarCRM, since it has in effect monopolized the development process of standard components and their underlying core. For, as Capra

[176] In this context, one also needs to consider that these categories are anything but discrete. That is, one can belong to one or several of them at once. E.g.: Sugar CE users can also be customers if they buy proprietary complementary services or software extensions from affiliates.
[177] As understood by Stabell and Fjeldstad (1998).
[178] E.g. Sugar CE users benefit from being many in numbers, since they provide mutual assistance (help forums) and contribute to the development Sugar CE and compatible modular extensions (same-side). Meanwhile, Customers benefit from core code contributions and modular extension provided by both Sugar CE users and SugarCRM affiliates (cross-side).
[179] In reference to "value shops", as understood by Stabell and Fjeldstad (1998).

and Wasserman (2008, p.12, my emphasis) already noted: "SugarCRM governance and managerial styles are actually very similar to those of a traditional closed software project, with the only exceptions that most of the code is open and that external people *can* contribute code". That is not to say that open access to the code has been rendered irrelevant. On the contrary, the availability of source code (even for proprietary components) is a decisive value driver. But even if this openness is benign and noteworthy, it does not quite seem to qualify as "open source" – at least not in the sense the term is commonly construed. Yes, the Sugar CE edition is licensed under the OSI-approved AGPLv3, which in the eyes of most FOSS supporters is laudable. And, yes, one *can* contribute to a shared and open code repository – albeit under the condition of copyright assignment. But then again, restrictive managerial practices and innovative legal acrobatics[180] seem to defy several implicit promises of open source. Even if a comparative "openness deficit" can be expected for COSS projects, SugarCRM stands out with a remarkably constrictive approach. According to a taxonomy proposed by Dahalnder and Magnusson (2005), this makes SugarCRM a probable candidate for the "parasitic" category, characterized by "violation of basic norms, values and principles" or perceptions of being a "free rider". Herein lies a conundrum comprising a whole set of related friction points (see table on next page):

[180] I.e., far-reaching supplementary licensing terms that render the software "badgeware" in the eyes of critics. And, if seen together with SugarCRM's Trademark Policy, appear to circumvent the FSF's freedoms / OSI's criteria relating to free distribution of derived works (see chapter 4.3.3).

Friction point	COSS / SugarCRM perspectives	Opposing FOSS perspectives
Copyright assignment	Precondition for dual licensing, without which the business model would not be commercially sustainable.	Disincentive to contribute because of fears of going (partially) private. Free Software purists: "Immoral".
Withholding of value driving functionality from Sugar CE	1) Pre-emptive competitive advantage against OSS clones, 2) wider scope for price discrimination and product differentiation for commercial editions, and 3) stronger incentives for Sugar CE users to migrate to a commercial edition.	"Crippleware" / damaged good, "open core". Disincentive to contribute because of lacking assurances against potentially exclusive proprietary use.
"Powered by SugarCRM" logo	1) Official stance: Legitimate author attribution in recognition of invested work. Not confirmed, but highly plausible: 2) brand promotion, and 3) thwart forking attempts / stifle unsolicited external code reuse.	"Badgeware". Violation of basic FOSS principles, especially when coupled with the SugarCRM Trademark Policy.
"closed" governance practices, even restrictive by COSS standards	1) Need for managerial control to ensure that customers' needs are efficiently met. 2) Speculative: Diminish the influence of FOSS enthusiasts and vigilantes, who could interfere with a commercially guided development process.	Overly restrictive, lack of procedural fairness. No real influence over shared Sugar CE code base. De-facto relegation to work on small-scale peripheral complements, which not need to be open source.
Preferential treatment of commercially affiliated community constituents and third parties	Reasonable supplementary approach of differentiation to utilize and enhance commercially vested interests in SugarCRM's product platform. This is 1) to strengthen the firm's sales channels through a co-evolution of capabilities with partners, and to 2) stimulate demand driven customization and development of modular complements (extensions, plug-ins etc.), 3) triggering network effects which increase the overall value of the product platform.	Deficient distributional fairness (in terms of underprovided focus and priority). Perception of being kept out of the loop (the latter not directly observed, but implied through qualitative responses to the questionnaire).

Table 3: Friction points between SugarCRM and FOSS community.

It is probably because SugarCRM has sought to counterbalance inherent weaknesses of its COSS business model, that it has followed an increasingly constrictive trajectory. Ever since vTiger appeared as fork of its Open Source edition in late 2004, and SugarCRM's leadership felt compelled to introduce attribution ("badgeware") terms to its infamous "Exhibit B" clause, the company has remained at odds with the wider FOSS community. This appears to be expected from a theoretical perspective (Osterloh and Rota, 2007 p.163): "once a technology enters the commercial phase of competition, one would expect the collective invention mode to break down (Meyer, 2003). […] Appropriation concerns start to dominate; exploration of the potential of a technology is replaced by exploitation of its commercial value". Hence, SugarCRM is likely to have sought to avoid open source cannibalization of its proprietary editions by opportunistically changing its sharing behavior. Such cannibalization would incur if (1) the open source edition left too little utility scope for feasible product differentiation and price discrimination[181], and / or (2) if competitive forks were to appear and deprive SugarCRM's value network of development externalities and customers. In a worst case scenario, a potent fork would capture large swathes of the Sugar CE installed base and soon be in a position to compete directly against SugarCRM's commercial editions.

[181] I.e. by virtue of providing extensive value that cannot be monetarily appropriated under open source terms, and lack of enough marginal proprietary value to maintain commercial sustainability.

In the event that this fork also managed to offer low switching costs[182] to SugerCRM's affiliates and customers, negative feedback loops could lead an already weakened value network to implode. And one should not necessarily discard FOSS projects as cacophonic hydras unable to muster or coordinate the required efforts. For history has shown that, if led by a mobilizing entity, FOSS projects can even outmatch their proprietary counterparts[183].

On the face of it, if one were to judge simply by official announcements, the company remains dedicated to the open source spirit. But as a result of putting safe value appropriation over risky value creation, it has in effect undercut the relevance of a voluntary FOSS community and compromised its open credentials. On the other hand, it has managed to mobilize its commercial affiliates – whose contributions now dwarf those of Sugar CE users. So, one may ask, why not go "the whole nine yards" and take the project private? If the open source edition represents a "wild card" liability and voluntary developers can be crowded out by employed ones, would this not be an appropriate move? Surely, even if confirming their suspicions, FOSS "purists" would applaud it as honest. Sugar CE could be transformed into proprietary freeware, the source code kept available for inspection and mundane non-redistributable customizations[184], and SugarForge rebranded as what it essentially is: a distribution platform for complementary software components. Of course, SugarCRM would then also need to substitute its "Commercial Open Source CRM" slogan, preferably with one that retains attuned connotations[185]. And it could still encourage development of open source peripheral extensions. With prudent community management, SugarCRM might more or less continue business as usual. And it would no longer be exposed to accusations of operating under a false flag, nor fear inadvertent cannibalization from an open source edition. However, this line of thought implies that open source has become a strategic anachronism for SugarCRM, at a time when it has already played out its "disruptive innovation" purpose. Further, it presupposes that the "Commercial Open Source" brand is non-essential for marketing purposes, and that a transition would not indeed provoke a fork from the last Sugar CE distribution. Last but not least, it disregards vested interest in the status quo. With an eminent open source serial-entrepreneur at the helm, it would undoubtedly seem ironic if SugarCRM took this symbolically strong step.

[182] E.g. through a support framework, technical compatibility and / or convenient migration offerings.
[183] E.g. The Mozilla Foundation (Firefox) vs. Microsoft (Internet Explorer).
[184] Under terms that are often referred to as "shared source" or "source-available".
[185] E.g. "Community Driven CRM".

SugarCRM appears to have found a pragmatic middle ground. Rather than being twofaced or ambivalent, as some interpretations may suggest, this proposition infers that the company realistically sees and treats its open source edition according to what it plausibly can be construed as – an emotionally labile hobgoblin that serves and dwells under the same roof as the commercial editions. Although essentially friendly and helpful, this troublesome creature has an unpredictable temperament and can become mischievous if aroused. Specimens of this particular breed are known for their ability to work their masters' land and scare off intruders from what they unwittingly confuse as their own dominion. Conversely, they also display an inherent disposition to opportunistically switch their allegiance. In fact, it is known that some have served under several masters simultaneously – which can have disturbing ramifications. Exactly that happened in late 2004, when our exemplar got tempted into helping a stranger to encroach onto Mr. Sugar's very own territory. Angered and disappointed, the old master resolutely burned his distinctive mark onto the hobgoblin's forehead, later followed by a nose ring to better control the incensed beast. Because of its lacking trustworthiness and natural propensity to interfere in the work on the land, it was confined to the narrow perimeters of the garden. The master's sons had by now grown old and strong enough to plow, while their reliable friends helped out with sowing and fertilizing the fields. And, as reaffirmed by the master's interpretation of Sun-Tzu and Machiavelli, one is wise to keep enemies even closer than friends. Not that this hobgoblin was considered an adversary as such, but it could certainly be enticed into becoming one. This is realpolitik, so the master reasoned. Never mind those eccentric animal rights activists who keep complaining about cruelty and unnaturally constrained living conditions. Surely, they are misguided. After all, he knows this breed better than anyone, and the law does not require him to make any concessions. Or does it? Well, if so, he could always try to put his servant out of its misery[186]. But then he would also lose a deterrent to trespassers. Even worse, that creature would be all the more likely to turn against him if he failed in his first attempt. Not to mention the odd gardening work it still performs, which is so nicely depicted on the labels of the land's sweet produce. Best not to feed the labile friend too much muscle building proteins then, just in case a judge came along and forced him to reconsider his options.

[186] I.e. take Sugar CE private (discontinue the development or distribute further versions under proprietary terms). This effort would not be successful if a potent and competitive fork appeared.

With reference to Creswell (2007, p.67), this somewhat caricatured analogy[187] is meant to "visually portray a conditional matrix that elucidates the social, historical, and economic conditions influencing the central phenomenon". Although subjective in tone, it effectively illustrates the issue at hand – friction points that could undermine the viability of SugarCRM's business model. And regardless of how well it reflects actual intentions and actions, it succeeds in pinning down the underlying dynamics. The hobgoblin is emblematic of all COSS projects, but the story plays out differently according to varying exogenous and endogenous factors. As exemplified by MySQL, it can end well for all involved parties. In another version[188], a disgruntled and determined hobgoblin finds considerable support in the activist, and eventually manages to become its own master.

The ending of SugarCRM's story remains to be written. Based on observations put forward in this thesis, one could just as well have interpreted Sugar CE as a working horse with a detrimental predisposition to metamorphose into a Trojan one. Whichever metaphor one chooses, SugarCRM's business model has innately steered the firm into a problematic *zugzwang* position, which it could only escape by making significant concessions. So far, it seems to have succeeded through a consistent strategy of containment and control – albeit with accompanying accusations of foul play. To strengthen its defense, it has sought to reinterpret and augment the rules of the game. If SugarCRM's underlying legal stance were to endure the scrutiny of an independent judge, then it would have achieved the feat of dodging a potentially dangerous genetic weakness in its business model. Even if this effectively renders Sugar CE's labor potential trifling, then at least it would no longer pose a credible threat. Alternatively, if a judge were to rule SugarCRM's apparent legal position as inconsistent with the applied AGPLv3 license, the company might have to consider the prospect of transmuting its business model into a "source available"-freeware variant. Yet given SugarCRM's preeminent market position and preemptive application of supplementary tactics, this thesis suggests that such a controversial and risky step would still be avoidable. It remains the privilege of a complex SugarCRM actor-network to play out its own story through inscribed proxies and direct interplay. One can attempt to intervene from the outside, but it is up to SugarCRM to determine the general direction. For, after all, the company now stands as the undisputed master of its "Commercial Open Source CRM" turf.

[187] Reminiscent to a narrative tone found among FOSS pundits: Sarcastic, speculative and tendentious.
[188] One well known example is that of Joomla, an open source CMS that became a successful fork of Mambo.

4.5 Further research

Although there already exists a considerable body of accumulated research on open source, the field of single-vendor commercial open source remains noticeably understudied. I therefore hope that my thesis has contributed some useful findings and new conceptual angles to this particular academic niche. However, I realize that I have only scraped on the surface of what could, and should, be investigated further. The following areas of inquiry appear of particular interest for further research:

- Retrieval of concealed data: What is, for example, the number of *active* community members, and how can one attain related information (e.g. average churn rate) for different community constituencies?

- A multivariate and comparative approach: How do variables such as FOSS community size, legal frameworks (restrictive-permissive), project maturity (basic-intermediate-advanced), predominant motivational factors (intrinsic-extrinsic) and governance models (open-closed) correlate with one another?

- Other theoretical frameworks: In a community with mutually competing commercial affiliates of a single governing COSS entity, how can one assure that valuable information (code) is shared with the community and not hoarded? Such questions could be enlightened through the use of game theory. Similarly, systems dynamics, agent-based modeling and scenario analysis appear to have great potential in this line of inquiry.

Bibliography

Akrich, M. (1992*) The De-Scription of Technical Objects* in Bijker and Law (eds.) *Shaping Technology / Building Society: Studies in Sociotechnical Change*, Cambridge, MA, The MIT Press.

Akrich, M. and Latour, B. (1992) *A Summary of a Convenient Vocabulary for the Semiotics of Human and Nonhuman Assemblies*, In Bijker, W. E. and Law J. (eds.) *Shaping Technology / Building Society: Studies in Sociotechnical Change*, Cambridge, MA, The MIT Press.

Amor, J. J., Robles, G., et.al. (2009) *Measuring Lenny: the size of Debian 5.0*, Retrieved June 18, 2012 from http://gsyc.es/~frivas/paper.pdf.

Bampton, R., and Cowton, C. (2002) *The E-Interview*. Forum: Qualitative Social Research, 3(2): http://www.qualitative-research.net/index.php/fqs/article/view/848/1842

Berdal, S. R. B. (2004) *Public deliberation on the Web: A Habermasian inquiry into online discourse*. Hovedfag Thesis. Dept. of Informatics, University of Oslo.

Berdal, S. R. B. and Føleide, L.R. (2008) *Networked business models and strategies for modern e-commerce: Case study of a generic business concept*. Master Thesis. Centre for Entrepreneurship, University of Oslo.

Bonaccorsi, A. and C. Rossi (2003), *Why Open Source software can succeed.* Research Policy 32(7): 1243-1258.

Brooks, F.P, Jr. (1975) *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA

Brownstein, M., Vining, A, et. al. (2006), *Can Open Source Disrupt Enterprise Software? The Case of CRM*. Proceedings of the Annual Conference of the Administrative Sciences Association of Canada, Banff, Alberta June 3-6, 2006, pp.221-237.

Callon, M. (1991) *Techno-economic networks and Irreversibility*, In Law, J. (ed.) *A Sociology of Monsters. Essays on Power, Technology and Domination*. London: Routledge.

Capek, P. G., S. P. Frank, et al. (2005), *A history of IBM's open-source involvement and strategy*. IBM Systems Journal 44(2): 249-257.

Capra, E. and A. Wasserman (2008). *A Framework for Evaluating Managerial Styles in Open Source Projects*. In IFIP International Federation for Information Processing, Volume 275; *Open Source Development, Communities and Quality*. B. Russo, E. Damiani, S. Hissam, B. Lundell and G. Succi, Springer Boston, pp. 1-14.

Capra, E., C. Francalanci, et al. (2009), *A Survey on Firms' Participation in Open Source Community Projects,* in *Open Source Ecosystems: Diverse Communities Interacting*. C. Boldyreff, K. Crowston, B. Lundell and A. Wasserman, Springer Boston. 299: 225-236.

Charles, John (1998), *Open Source: Netscape Pops the Hood*, in *In the News*. IEEE Software, 15(4), 79-82. Retrieved November 6, 2011, from ABI/INFORM Global. (Document ID: 1750960911).

Creswell, J. W. and Miller, D. L. (2000), *Determining Validity in Qualitative Inquiry*. Theory Into Practice 39(3): 124-130.

Creswell, J.W. (2007) *Qualitative inquiry and research design: Choosing among five approaches*, Thousand Oaks: Sage.

Dahlander, L. and Magnusson, M. G. (2005), *Relationships between open source software companies and communities: Observations from Nordic firms*. Research Policy 34(4): 481-493.

Deek, F. P. and McHugh, J. A. M. (2007), Open source: Technology and policy. Cambridge: Cambridge University Press.

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

Fitzgerald, B. (2006), *The Transformation of Open Source Software*. MIS Quarterly, 30(3), 587-598.

Fogel, K. (2005), *Producing Open Source Software: How to Run a Successful Free Software Project*, O'Reilly Media.

Garzarelli, G., Y. R. Limam, et al. (2008) *Open source software and economic growth: A classical division of labor perspective*. Information Technology for Development 14(2): 116-135.

Glass, R. L. (2003). *Facts and Fallacies of Software Engineering*. Boston, MA: Addison-Wesley.

Ghosh, R. A. and Prakash, V. V., (2000), *The Oribiten Free Software Survey*, First Monday 5(7). Retrieved Sept. 5th 2012 from http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/article/view/769/678.

Glaser, B. G. (1978) *Theoretical sensitivity*. Mill Valley, CA: Sociology Press.

Glaser, B. G. (1998). *Doing Grounded Theory: Issues and Discussions*. Mill Valley, CA: Sociology Press.

Hanseth, O. and Monteiro, E. (1998), chapter 6 in *Understanding Information Infrastructure*. Manuscript of August 27th, 1998, Retrieved March 8th, 2012 from http://heim.ifi.uio.no/~oleha/Publications/bok.html.

Harhoff, D., Henkel, J., et al. (2003), *Profiting from voluntary information spillovers: how users benefit by freely revealing their innovations*. Research Policy 32(10): 1753-1769.

Harhoff, D. and Mayrhofer, P. (2010). *Managing User Communities and Hybrid Innovation Processes: Concepts and Design Implications*. Organizational Dynamics 39(2): 137-144

Hars, A. and Shaosong, O. (2001) *Working for free? Motivations of participating in open source projects*. Proceedings of the 34th Annual Hawaii International Conference on System Sciences.

Healy, K. and Schussman, A. (2003) *The Ecology of Open Source Software Development*, Working Paper, Department of Sociology, University of Arizona, Tucson, AZ (available online at http://www.kieranhealy.org/files/drafts/oss-activity.pdf).

Henkel, J. (2006). *Selective revealing in open innovation processes: The case of embedded Linux.* Research Policy 35(7): 953-969.

Henkel, J., Baldwin, C.Y. (2010) *Modularity for Value Appropriation - How to Draw the Boundaries of Intellectual Property*. Harvard Business School Finance Working Paper, No. 11-054, http://dx.doi.org/10.2139/ssrn.1340445

Jensen, C. and W. Scacchi (2011) *License Update and Migration Processes in Open Source Software Projects Open Source Systems: Grounding Research*. S. Hissam, B. Russo, M. de Mendonça Neto and F. Kon, Springer Boston. 365: 177-195.

Kivits, J. (2005) *Online interviewing and the research relationship*. In C. Hine (Ed.), *Virtual methods: Issues in social research on the Internet* (pp.35-50). Oxford: Berg.

Lakhani, K. R. and E. von Hippel (2003). *How open source software works: "free" user-to-user assistance.* Research Policy 32(6): 923-943.

Lakhani, K. R., and Wolf, R. G. (2005). *Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects*. In J. Feller, B. Fitzgerald, S. A. Hissam, & Karim R Lakhani (Eds.), Perspectives on Free and Open Source Software (pp. 3-21). MIT Press.

Latour, B. (1987) *Science in Action: How to follow scientists and Engineers through society*. Milton Keynes, UK: Open University Press

Latour, B. (1991) *Technology is Society Made Durable*, In Law, J. (ed.) *A Sociology of Monsters. Essays on Power, Technology and Domination*. London: Routledge.

Law, J. (1992) *Notes on the Theory of the Actor-Network: Ordering, Strategy and Heterogeneity*. Systems Practice, 5(4): 379-393.

Lerner, J. and Tirole, J. (2001), *The open source movement: Key research questions.* European Economic Review 45(4-6): 819-826.

Lerner, J. and Tirole, J. (2002), *Some Simple Economics of Open Source*. The Journal of Industrial Economics 50(2): 197-234.

Lerner, J. and Tirole, J. (2005). *The Economics of Technology Sharing: Open Source and Beyond*. The Journal of Economic Perspectives 19(2): 99-120.

Markus, M. (2007), *The governance of free/open source software projects: monolithic, multidimensional, or configurational?*. Journal of Management and Governance 11(2): 151-163.

Maxwell, J.A. (1996), *Qualitative Research Design: An interactive approach*. Sage Publications, Thousand Oaks, CA.

Meyer, P.B. (2003), *Episodes of Collective Invention*. US BLS working paper no. 368. Available at http://ifipwg213.org/system/files/meyer.pdf (Sept. 2012).

Midha, V. and P. Palvia (2012) *Factors affecting the success of Open Source Software*. Journal of Systems and Software 85(4): 895-905.

Mikkonen, T., Vainio N. and Vadén T. (2006), *Survey on four OSS communities: description, analysis and typology*. Empirical Insights on Open Source Business. Helander N. & Mäntymäki, M. (eds.) Tampere: Tampere University of Technology and University of Tampere.

Mikkonen, T. and Vadén T. (2009). *The Anatomy of Sustainable Open Source Community Building: the Cultural Point of View*. Teoksessa Imed Hammouda, Timo Aaltonen, Andrea Capiluppi (toim.) Proceedings of the First International Workshop on Building Sustainable Open Source Communities. Tampere: Tampereen Teknillinen Yliopisto, 14-20. (Tampere University of Technology, Department of Software Systems, Report 3). Retrieved from http://tutopen.cs.tut.fi/oscomm09/program.shtml

Monteiro, E. (1999). *Monsters*, In *Planet Internet* , K. Braa, B. Dahlbom and C. Sørensen (eds.), Studentlitteratur, Lund, pp. 239 - 249.

Morin, A., J. Urban, et al. (2012). *A Quick Guide to Software Licensing for the Scientist-Programmer*. PLoS Comput Biol 8(7): e1002598, doi:10.1371/journal.pcbi.1002598

O'Mahony, S. and F. Ferraro (2007), *The Emergence of Governance in an Open Source Community*, The Academy of Management Journal ARCHIVE 50(5): 1079-1106.

O'Mahony, S. and Bechky, B. A. (2008). *Boundary Organizations: Enabling Collaboration among Unexpected Allies*. Administrative Science Quarterly 53(3): 422-459.

Olson, M. (2005), *Dual Licensing*, chapter 5 in *Open Sources 2.0: The Continuing Evolution* (Ed. C., DiBona, D., Cooper and M., Stone), O'Reilly.

Opdenakker, R. (2006), *Advantages and Disadvantages of Four Interview Techniques in Qualitative Research*. Forum: Qualitative Social Research, 7(4). Retrieved from http://www.qualitative-research.net/index.php/fqs/article/view/175/391

Oreg, S. and Nov, O. (2008), *Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values.* Computers in Human Behavior 24(5): 2055-2073

Osterloh, M. and Rota, S. (2007), *Open source software development—Just another case of collective invention?* Research Policy 36(2): 157-171.

Perens, B. (1999), *The Open Source Definition*, Chapter 12 in DiBona C, Ockman S and Stone M, eds. (1999), *Open Sources: Voices from the Open Source Revolution*, Sebastopol, CA: O'Reilly & Associates, Retrieved November 3, 2011, from http://oreilly.com/catalog/opensources/book/perens.html

Perens, B. (2011), *The Covenant - A New Approach to Open Source Cooperation*. White paper on website of HPCC Systems. Retrieved August 19, 2012, from http://hpccsystems.com/community/white-papers/the_covenant_bperens

Raymond, E. S. (1997), *The Cathedral and the Bazaar*, Retrieved November 3, 2011 from http://www.catb.org/~esr/writings/cathedral-bazaar. Page references are to the revised 2001 book edition (Sebastopol, CA: O'Reilly).

Raymond, E. S. (1999a), *A Brief History of Hackerdom*, Chapter 2 in DiBona C, Ockman S and Stone M, eds. (1999), *Open Sources: Voices from the Open Source Revolution*, Sebastopol, CA: O'Reilly & Associates, Retrieved November 3, 2011, from http://oreilly.com/openbook/opensources/book/raymond.html

Raymond, E. S. (1999b), *The Magic Cauldron*, Retrieved May 19, 2011 from http://www.catb.org/esr/writings/magic-cauldron/ (version 1.17).

Riehle, D. (2012), *The single-vendor commercial open course business model*. Information Systems and E-Business Management 10(1): 5-17.

Rosen, L. (2005), *Open Source Licensing: Software Freedom and Intellectual Property Law*. Upper Saddle River, NJ: Prentice Hall.

Rossi, I.D. (2011), *vTiger: CRM Beginner's Guide*, Birmingham, UK: Packt Publishing Ltd.

Sauer, R. M. (2007) *Why develop open-source software? The role of non-pecuniary benefits, monetary rewards, and open-source licence type*. Oxford Review of Economic Policy 23(4): 605-619.

Scacchi, W. (2002). *Understanding the requirements for developing open source software systems*. IEE Proc. Software 149(1) 24–39.

Shah, S. K. (2006), *Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development*. Management Science 52(7): 1000-1014.

Stabell, C. B. and Fjeldstad, Ø. D. (1998), *Configuring value for competitive advantage: on chains, shops, and networks.* Strategic Management Journal 19(5): 413-437.

Sterne, P. and Herring, N. (Nov. 16, 2006), *SugarCRM - A Sweet Mix of Commercial and Open Source in Linux*, retrieved Aug. 11 from http://linux.sys-con.com/node/173436

Stewart, K. J., Ammeter, A. P. , et al. (2006). *Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects*. Info. Sys. Research 17(2): 126-144.

Stewart, K. J., and Gosain, S. (2006), *The Impact of Ideology on Effectiveness in Open Source Software Development Teams*, MIS Quarterly (30:2), pp. 291-314.

Strauss, A. and Corbin, J. (1998), *Basics of qualitative research: Techniques and procedures for developing grounded theory*. Thousand Oaks, CA: Sage.

Torvalds, L. (1999) The Linux Edge, in DiBona, Ockman and Stone (eds.), Open Sources: Voices from the Open Source Revolution. O'Reilly,Sebastopol, CA: 101-11. See also http://oreilly.com/openbook/opensources/book/

Tatnall, Arthur (2003), *Actor-Network Theory as a socio-technical approach to Information Systems Research*, In Clarke, S., Coakes, E., Hunter G.M. and Wenn, A. (eds.) *Actor-Network Theory and human cognition elements of Information Systems*, Hershey, PA: Idea Group Publishing, pp.266-283.

Tuomi, I. (2001), *Internet, Innovation, and Open Source: Actors in the Network*, FirstMonday, vol. 6, no. 1.

von Hippel, E. (2001) *PERSPECTIVE: User toolkits for innovation*. Journal of Product Innovation Management 18(4): 247-257.

von Hippel, E. and Katz, R. (2002) *Shifting innovation to users via toolkits*. Management Science, 48, 7, 821–833.

von Hippel, E. and G. von Krogh (2006), *Free revealing and the private-collective model for innovation incentives*. R&D Management 36(3): 295-306

von Krogh, G., Haefliger S., Spaeth, S., Wallin M.W. (2012), *Carrots and Rainbows: Motivation and Social Practice in Open Source Software Development*, MIS Quarterly, (36: 2) pp.649-676.

Waltl, J., Henkel, J. et al. (2012). IP Modularity in Software Ecosystems: *How SugarCRM's IP and Business Model Shape Its Product Architecture Software Business*. M. A. Cusumano, B. Iyer, N. Venkatraman et al., Springer Berlin Heidelberg. 114: 94-106.

Weber, S. (2004), *The Success of Open Source*, Harvard University Press.

Appendix A: Terminology

This appendix clarifies my understanding of the terms marked with asterisks (*) throughout this thesis.

Fork / forking

A fork is source code that is copied from an original project, and used as the foundation of a another. The term is also applied on resulting new FOSS projects (e.g. LibreOffice as a fork of OpenOffice.org). The legal basis for forking are freedoms / criteria[189] of the Free Software Foundation (FSF) and Open Source Initiative (OSI), which are reflected in officially certified licenses.

Forking (also used as a verb) is understood to be detrimental to the originating project, as it creates a schism amongst its user base and developer community. It is also likely to result in duplication of effort, and to undermine the trust and authority of the originating project maintainer. When FOSS software is forked, the new fork inherits the license that governed the originating source code.

FOSS

A generic term used to refer to software that is "free" (as defined by the Free Software Foundation) and / or "open source" (as defined by the Open Source Initiative).

ISV

Independent Software Vendor. To SugarCRM, ISVs fall into one of the main categories of partners (in addition to VARs).

OEM

Original Equipment Manufacturer. Refers to a manufacturer of products that are purchased and typically branded by other companies (value-added resellers / VARs). OEM is sometimes also used to refer to the product itself – i.e. as original equipment from manufacturer. SugarCRM offers an OEM version of its software (*Sugar Platform Edition* with its own license terms, through the "Powered By SugarCRM OEM Program")[190].

[189] By the Free Software Definition (FSD) and Open Source Definition (OSD), respectively.
[190] *Sugar Platform Edition FAQs*. Retrieved May 28, 2012, from http://www.sugarcrm.com/page/sugar-platform-edition-faqs/en

Project maintainer

A project maintainer (PM) is the entity, typically a single person, which holds the formal leadership position of a FOSS project. Also see chapter 2.3.3.

SugarExchange

Similar to *SugarForge*, it is a distribution platform for complementary software components. But it also serves a "market place" (sales channel) for commercial complement "providers" (payments are facilitated through PayPal), and offers official certification of modular extensions ("Certified by SugarCRM"). However, a large proportion of the offerings here are free and uncertified, and overlap with the ones found on *SugarForge*. As with *SugarForge*, a large share of the offerings also appear outdated (not supporting the latest versions, and / or with reviews that are several years old).

SugarForge

According to SugarCRM itself, "SugarForge is the destination for the Sugar community to collaborate and develop Sugar extensions"[191]. Essentially, it is a distribution platform for complementary software components, just as *SugarExchange*, albeit with collaborative features and a more "open" look. As with *SugarExchange*, a large share of the offerings also appear outdated (not supporting the latest versions, with reviews that are several years old, and / or no notable project activity). And, although it is meant as a platform for collaboration, my observations cannot confirm any noteworthy collaboration amongst community members. See chapter 4.3.2 for more findings.

VAR

Value-Added Reseller. A company that adds new features or services (i.e. value) to an existing product, and typically resells it under its own brand. In the software industry, VARs often add features to a product to target specific customers with corresponding needs. Thus, to SugarCRM, VARs are one of two main categories of partners (in addition to ISVs).

[191] See http://www.sugarforge.org/

Appendix B: Review of *The Cathedral and the Bazaar*

This appendix contains my assessment of Eric S. Raymond's nineteen "lessons" presented in "*The Cathedral and the Bazaar*" [192]. Each listed item refers to one lesson (original enumeration and naming), with my own reflections stated below. All citations refer to this particular (official) online edition, unless stated otherwise. In the closing, this Appendix also touches upon some of Raymond's other related work.

1. *Every good work of software starts by scratching a developer's personal itch.*

   This is another way to say that "Necessity is the mother of invention". In the context of OSS, however, this statement has particular relevance: A piece of software needs not be a new stand-alone application, but could also be a patch or fork of existing OSS (i.e. the "barriers to entry" are significantly lower when existing code can be adopted and adapted).

2. *Good programmers know what to write. Great ones know what to rewrite (and reuse).*

   Building on lesson 1, this is a normative advice to rewrite and reuse code, rather than reinventing the wheel (or parts of it). The rationale is that it is better to be a smaller contributor to something bigger than writing something from scratch. Not only does this provide a higher probability of widespread use, but also assures better quality (wisdom of the crowds, see lesson 8) and overall increased utilization of limited resources.

3. *"Plan to throw one away; you will, anyhow." (Fred Brooks, The Mythical Man-Month)*

   As stated, this is a renowned lesson already learned by Fred Brooks (which states that you should plan to build a pilot, since you will need to do so anyhow). In the context of OSS, this is one way of saying that it is never too late to jump ship – to realize that a project has severe limitations (sunk costs), but your experience (along with parts of the code) could be beneficial for another OSS project. This is a complementary interpretation to the straight-forward one (i.e. to foresee the need for a pilot), which is just as relevant for OSS.

[192] Retrieved Jan 24, 2012, from http://catb.org/~esr/writings/homesteading/cathedral-bazaar/index.html Version 3, revision 1.57, dated Sept. 11, 2000

4. *If you have the right attitude, interesting problems will find you.*

   This is, as I interpret it, the expected result of other lessons being followed (both by oneself, and by others in the wider OSS realm). Raymond exemplifies this through his experience: He started off by joining one project, realized its limitations, found a better alternative and "jumped ship". After he had sent his first patches to the new project, he noticed that its maintainer had lost interest in it – which leads us to the next lesson:

5. *When you lose interest in a program, your last duty to it is to hand it off to a competent successor.*

   This is as much a lesson as an embracement of a theretofore tacit OSS convention of good conduct. Also, one might expect that a disinterested project maintainer recognizes the benefit of handing over his project, rather than seeing it fall into obsolescence or being forked.

6. *Treating your users as co-developers is your least-hassle route to rapid code improvement and effective debugging.*

   As Raymond states: "Properly cultivated, [… the users] can become co-developers", and further: "In fact, I think Linus's cleverest and most consequential hack was not the construction of the Linux kernel itself, but rather his invention of the Linux development model." Clearly, this lesson is as relevant to commercial OSS, as it is to conventional OSS. By listening to users, giving feedback, adapting the software according to their wishes etc., users will be more likely to join and stay in the community of co-developers. This matters, because an active and sizable community is an indispensable precondition for effective collective innovation and development. The ideal is that as many of the software's users contribute to the development process (i.e. as long as the project maintainer holds a certain minimum of administrative capabilities, and the community of users and developers is internally aligned). Thus, lowering the threshold for participation is fundamental. One way to achieve this is by offering "toolkits for user innovation" (von Hippel and Katz, 2002).

7. *Release early. Release often [(RERO)]. And listen to your customers.*

   In contrast to the traditional Cathedral model, the Bazaar model relies on an agile development philosophy that emphasizes rapid iterations of new releases and small incremental improvements[193]. This ensures a tight feedback loop between users and developers (i.e. identification of bugs / popular features by users, and fixes / enhancement by developers). As pointed out in lesson 6, users can also be developers (and developers are almost always users). But even if a developer reports a bug, it does not need to be this individual that is most competent or motivated to fix it (see next lesson). RERO also ensures to keep the "temperature high" by showing constant and steady progress. As Raymond explains: "Linus [Torvalds] was keeping his hacker/users constantly stimulated and rewarded—stimulated by the prospect of having an ego-satisfying piece of the action, rewarded by the sight of constant (even daily) improvement in their work." Likewise, one can plausibly argue, the piecemeal approach of RERO reduces overall complexity of the project by focusing attention on a manageable scope of collectively identified set of bugs / feature requests at any given time.

   The drawback, some may say, is that the software is undergoing constant change (not all users like that), and new releases are likely to contain more bugs than otherwise. But the remedy is simple: The offering of "stable releases" with predictable intervals (i.e. when the developers deem the provided features to be ready for a wider audience). The developers, meanwhile, can continue to work on interim releases at the very "bleeding edge" of code evolution. In the context of dual-licensing, it is intuitively beneficial to harmonize the stable releases across different editions (concurrently containing the same versions of core features), to encourage as much user-developer synchronization as possible. By having a shared reference point to stable releases, it is easier to utilize the community for support and bug-identification across different editions. The approach can also serve as an effective barrier against unintended or excessive divergence between different editions, which would increase complexity and reduce the very relevance of an OSS community.

[193] It should be noted that the benefits of short incremental iterations have also been recognized for proprietary software development, where agile methodologies / development frameworks (e.g. Scrum and Kanban) have since become widely adopted.

8. *Given a large enough beta-tester and co-developer base, almost every problem will be characterized quickly and the fix obvious to someone.*

The rationale behind this statement[194], which Raymond dubbed "Linus's Law", can be deconstructed as such:

Effective division of labor and a transparent, self-regulating marked: As Adam Smith firmly established as an axiom of capitalism, division of labor results in increased overall productivity and quality. The same general logic applies to OSS[195]. Raymond refers to Linus Torvalds saying that "the person who understands and fixes the problem is not necessarily or even usually the person who first characterizes it", and further: "Somebody finds the problem […] and somebody else understands it". Thus, the picking of tasks is done by a benign mixture of induced chance (someone among many beta-testers and developers discovers a bug) and skills / creativity / inclination (someone else understands and fixes it). A transparent market[196] allows tasks to be distributed efficiently by stimulating "individuals [to] spontaneously align with the problems they are most inclined to solve anyway" (Garzarelli et al., 2008, p.120) and thereupon voluntarily self-select tasks without much need for centralized coordination or interference[197]. Seen in relation to the previous lesson (RERO), it becomes apparent that the near-to self-regulating process of identification, comprehension and code submission is rapid and perpetually ongoing.

Reduction of complexity: Raymond states that "[i]n the cathedral-builder view of programming, bugs and development problems are tricky, insidious, deep phenomena. It takes months of scrutiny by a dedicated few to develop confidence that you've winkled them all out. […] In the bazaar view, on the other hand, you assume that bugs are generally shallow phenomena—or, at least, that they turn shallow pretty quickly when exposed to a thousand eager co-developers pounding on every single new release." The contrast between these two development stereotypes is stark on this point, as are the implications. Development of software may require a wide skill set by its creators. Even if

[194] More colloquially formulated: "Given enough eyeballs, all bugs are shallow".
[195] Albeit in a non-traditional sense (neither based on pure vertical or horizontal specialization of work, but on ad-hoc self-selection of atomic tasks by a pool of loosely associated volunteers and enthusiasts).
[196] Here understood as the socio-technological construct of an Internet aided community, including toolkits for bug-reporting and revision control, which distribute information openly and effectively to all involved parties.
[197] I.e. other than the general procedural framework that is established by the project-/ package maintainer, and the process of choosing and including contributed modifications for upcoming releases.

all anticipated skills needed for a particular software feature or workflow interval are assigned to a dedicated team of specialists, there is no guarantee that the matching of actual capabilities[198] and (often unforeseen) challenges is optimal. The bazaar model, on the other hand, seems to suffer from redundancy (several volunteers occasionally taking on the same challenges simultaneously). However (Garzarelli et al., 2008, p.127, original emphasis): "The redundancy is not a shortcoming […]. It engenders economies that have the ability to capitalize on multiple, intersecting knowledge combinatorics. These redundant economies from parallel, overlapping inputs encourage a production mode whereby being specialized in a particular task is not a *conditio sine qua non* to contribute: A contributor may still be a programmer in the formal sense or a final user with programming skills, but his input(s) may not always directly reflect his primary specialty. What matters is the spontaneity of the contribution because the shared belief is that there's potentially something to learn from everyone. In fact […] the central tenet of the bazaar is that the benefits of tapping from a virtually unlimited knowledge pool through a redundant division of labor outweigh all other costs, including coordination costs." This is a diametric opposite understanding of Brook's Law[199], conceived on the basis of closed-source development, which "predicts that the complexity and communication costs of a project rise with the square of the number of developers, while work done only rises linearly."

After the publication of "The Cathedral and the Bazaar", this lesson was one of the most hotly debated. Gates (2003, pp.174-75) referred to it as a "one of the mantras of the open-source community", and pointed out that it remained to be confirmed by empirical studies. Since then, several studies have assessed the relationship between OSS projects' community size and the projects' technical success. The results have been mixed, and suggest that other (correlated) variables are at play – such as maturity and novelty of the project, the ratio between core- and periphery developers / beta testers, composition and distribution of capabilities, and modularity of the code (e.g. Midha and Palvia, 2012). But none of the studies I found seem to be conclusive on the matter, either because of limited samples or narrow methodological approaches. Clearly, more thorough research is required.

[198] E.g. wider set of relevant skills and experience, creativity, attention and motivation.
[199] Original formulation: "Adding manpower to a late software project makes it later" (Brooks 1975, p.25)

However, one variable that seems to be conclusively important is modularity of the code: As mentioned by Torvalds (1999), "for without it [modularity], you cannot have people working in parallel". With a modular design, multiple programmers can work to build new functions into the same module (ibid, p.898). This brings us to the next lesson.

9. *Smart data structures and dumb code works a lot better than the other way around.*

This lesson may not be intuitive to someone without software development experience. One can think of a pool of builders and craftsmen, collaborating to build a new house. Data structures represent the commonly understood principles of sound building practices, and the blueprints based thereon. The code, meanwhile, may represent actual implementations of the house (e.g. a corridor with doors to adjacent rooms). If the house were a patchwork of excellent craftsmanship, you could get a corridor with splendid doors and floors. But the floor might be at a different level than of the adjacent rooms, and the door frames (designed for the rooms) too small or wide to fit with the doors. With some ad-hoc tweaking, one could perhaps get it to work, but the overall process would become more expensive and is likely to trigger conflicts among stakeholders. On the other hand, if the house had been built according to sound building practices (e.g. modularity with standardized interfaces for different segments) the house would get a consistent design even without splendid floors. And since the different segments have standardized and predictable interfaces (e.g. doors and door frames, toilets and plumbing), one could easily upgrade them independently at a later stage.

Modularity is a particularly important aspect, since it allows participants to work in parallel (see previous lesson), to encapsulate complexity and abstract / assign specialized functionality to dedicated modules. Thus, as Torvalds claims[200], "the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships."

[200] Email from Linus Trovalds to John Smirl (27 July 2006), available on LWN.net: http://lwn.net/Articles/193245/

10. *If you treat your beta-testers as if they're your most valuable resource, they will respond by becoming your most valuable resource.*

    In short, Raymond's reasoning on this point can be summarized as such:

    - Keep your users up to date, and encourage them to participate.
    - Ask users for advice, listen to them, and "stroke" those that provide feedback / patches.
    - Release early and often (RERO) - i.e. lesson 7.

    Raymond could have been clearer on what he meant by "encouraging people to participate" and "stroking them" as a reward. This is because non-pecuniary incentives play an essential role in OSS development (e.g. Sauer 2007). Such incentives can be provided either directly (such as bilateral recognition) and indirectly (e.g. offering an open forum through which multilateral peer recognition can be exchanged and stimulated). See also chapter 2.3.1 on motivations.

11. *The next best thing to having good ideas is recognizing good ideas from your users. Sometimes the latter is better.*

    This attitude / realization is an effective antidote to any form of the so-called NIH ("not invented here") syndrome, and an obvious precondition to any form for open and collective mode of innovation. What sets the Bazaar model apart from conventional corporate innovation is that it has translated this understanding into the core of its culture and modus operandi. However, this is less a question of "either-or" than of "to which extent". Commercial OSS projects in particular, can run a risk of selective attention towards different stakeholders.

12. *Often, the most striking and innovative solutions come from realizing that your concept of the problem was wrong.*

    As Raymond puts it: “When you hit a wall in development—when you find yourself hard put to think past the next patch—it's often time to ask not whether you've got the right answer, but whether you're asking the right question. Perhaps the problem needs to be reframed”. This is intuitive, and just as relevant to commercial projects. But it is, in fact, more than a reframing of the problem per se. If time and effort have been put into solving the wrong problem, it is just as much a question of ignoring incurred sunk costs and handling vested interests when considering the degree of reframing.

    In the context of COSS, the decision to reframe a problem or act upon the resulting implications could be more complex than for pure OSS projects. Say, such implications would entail a change in offered features or functionality (albeit superior), the COSS entity may be legally committed to offer bug-fixing, support and updates for the current version. Thus, unless it is possible to convince all customers to “upgrade”, parallel development and support paths may be required.

13. *“Perfection (in design) is achieved not when there is nothing more to add, but rather when there is nothing more to take away.”*

    Paraphrasing Antoine de Saint-Exupéry, Raymond reformulates what is commonly known as the KISS principle (“Keep it simple, stupid”, or “keep it short and simple”). The underlying belief is that complexity can be reduced (and often performance improved) by promoting simplicity and minimalism. For OSS in particular, this maxim is of highest relevance. Code that is maintained and augmented by a diverse set of contributors must be comprehensible and easy to build upon.

14. *Any tool should be useful in the expected way, but a truly great tool lends itself to uses you never expected.*

    This point is of no particular relevance in the context of (C)OSS.

15. *When writing gateway software of any kind, take pains to disturb the data stream as little as possible—and never throw away information unless the recipient forces you to!*

    This point is of no particular relevance in the context of (C)OSS.

16. *When your language is nowhere near Turing-complete, syntactic sugar can be your friend.*

    This point is of no particular relevance in the context of (C)OSS.

17. *A security system is only as secure as its secret. Beware of pseudo-secrets.*

    This point is of no particular relevance in the context of (C)OSS.

18. *To solve an interesting problem, start by finding a problem that is interesting to you.*

    This is a rephrasing of lesson 1, as Raymond puts it, “in a perhaps more useful way”.

19. *Provided the development coordinator has a communications medium at least as good as the Internet, and knows how to lead without coercion, many heads are inevitably better than one.*

    Raymond cites the anarchist Kropotkin's Memoirs of a Revolutionist to explain the nature of the Internet, and thus identifies the compatible mode of planning and action as the “principle of common understanding”[201]. As a result, in Raymond’s words, “[t]he Linux world behaves in many respects like a free market or an ecology, a collection of selfish agents attempting to maximize utility which in the process produces a self-correcting spontaneous order more elaborate and efficient than any amount of central planning could have achieved.”

    It is interesting to see Raymond use the term “selfish” in this context, as the Open Source movement is often conceived and depicted as the opposite: Idealistic, almost to the extent of altruistic. Although this idealism may be a contributing factor of motivation in parts of the movement, Raymond seems to have a point in that most OSS developers are driven by some form of self-interest. In his words, “[t]he ``utility function'' Linux hackers are maximizing is not classically economic, but is the intangible of their own ego satisfaction and reputation among other hackers. Or more specifically, in the words of Lakhani and von Hippel (2003): “motives used to explain why users would voluntarily work on these basic tasks include: (1) a user’s direct need for the software and software improvements worked upon; (2) enjoyment of the work itself; and (3) the enhanced

[201] The aim of which “can be achieved only through the severe effort of many converging wills”. This stands in contrast to the “principle of command”, which is “effectively impossible to apply among volunteers in the anarchist's paradise we call the Internet”.

reputation that may flow from making high-quality contributions to an open source project." Even a "mundane but necessary task", such as support, can be explained by such motives[202] (ibid). See also chapter 2.3.1 on motivations.

As much as the Internet defines the viable palette of leadership types, it facilitates and stimulates certain forms of communication and cooperation. Indeed, the Internet is a precondition for the Bazaar model to work, as it enables the rapid circulation of information, automation of tasks, and custom made toolkits for distributed development.

Raymond thus concludes:

> *Perhaps in the end the open-source culture will triumph not because cooperation is morally right or software ``hoarding'' is morally wrong […], but simply because the closed-source world cannot win an evolutionary arms race with open-source communities that can put orders of magnitude more skilled time into a problem.*

In hindsight, this somewhat presumptuous postulation deserves to be taken seriously. Indeed, if not constituting a self-fulfilling prophecy, then at least *The Cathedral and The Bazaar* served a catalyst by spreading the word and encouraging mainstream adoption[203].

In a less cited publication, Raymond (1999b) augments his argumentation with economic reasoning. Building on game-theory models, he presents an analysis of the stability of open source cooperation, and proposes models "for sustainable funding of open-source development". Interestingly, and perhaps not surprisingly, he suggests that network effects and commoditization inevitably will force open source into hitherto proprietary market segments[204], and further asserts that such trends are already visible (in 1999) – even for "the heaviest business applications, such as ERPs"[205]. As he continues (in chapter 10.4):

> *If present trends continue, the central challenge of software technology and product management in the next century will be knowing when to let go -- when to allow closed code to pass into the open-source infrastructure in order to exploit the peer-review effect and capture higher returns in service and other secondary markets. There are obvious revenue incentives not*

[202] Here also understood to include reciprocal helping and benefits of attaining knowledge / information.
[203] E.g., as seemed to be the case with Netscape.
[204] Given higher "payoffs", e.g. through comparative advantages in competitive markets: Labor externalities that spur production of complements and drive adoption of a given software.
[205] I.e. Enterprise Resource Planning software. Interestingly, this is a good parallel to the CRM market.

to miss the crossover point too far in either direction. Beyond that, there's a serious opportunity risk in waiting too long -- you could get scooped by a competitor going open-source in the same market niche. The reason this is a serious issue is that both the pool of users and the pool of talent available to be recruited into open-source cooperation for any given product category is limited, and recruitment tends to stick. If two producers are the first and second to open-source competing code of roughly equal function, the first is likely to attract the most users and the most and best-motivated co-developers; the second will have to take leavings. Recruitment tends to stick, as users gain familiarity and developers sink time investments in the code itself.

Again, Raymond concludes his work with "some tentative predictions of the future":

In a future that includes competition from open source, we can expect that the eventual destiny of any software technology will be to either die or become part of the open infrastructure itself. While this is hardly happy news for entrepreneurs who would like to collect rent on closed software forever, it does suggest that the software industry as a whole will remain entrepreneurial, with new niches constantly opening up at the upper (application) end and a limited lifespan for closed-IP monopolies as their product categories fall into infrastructure[206]*.*

In spite of building on a number speculative conjectures, Raymond's appealing reasoning has stood the test of time. And even if some critique is warranted, as also proven by this appendix, its mere historic influence makes it an obvious academic reference point. Thus, as perhaps most open source scholars, I have found it useful to start my research on open source by delving into probably the most cited work in the field. It has provided me a valuable conceptual foundation, which later helped me in the process of interpreting findings from my problem domain – SugarCRM. As such, it should be recognized, for better or worse, as an early influence on my research.

[206] Implying commoditized market segments governed by strong standards.

Appendix C: FOSS motivations, "social practice view"

This is a summary (quoting) of theoretical conjectures and propositions made by von Krogh, Haefliger, Spaeth and Wallin, 2012 pp.666-8 ("motivation–practice framework" based on their "social practice view", which is complementary to the more common "self-determination view" – see chapter 2.3.1 on motivation). Two concept lent from MacIntyre require explanation for full understanding (ibid., p.660): "*External goods* (external to the social practice) include capital, status, or power, which are the property of individuals and/or institutions. *Internal goods* are defined by the social practice and are public goods that benefit all participants in the social practice and the wider community."

Question 1:

How and why do OSS developers produce high-quality software (goods) when they do?

Theoretical Conjecture 1:

OSS developers contribute to the production of internal goods (e.g. high-quality software) when their actions follow the standards of excellence of the social practice. At the same time, the internal good produced by the social practice impacts on standards of excellence pursued in the social practice.

Proposition 1:

Developers in a social practice create a sense of timing and developmental interactions with peers that improves the social practice.

Proposition 2:

The software product impacts on the standards of excellence in OSS development.

Question 2:

Why do OSS developers change institutions?

Theoretical Conjecture 2:

OSS developers change institutions when and where these institutions no longer protect sufficiently the standards of excellence of the social practice. In addition, institutions are changed in order to provide external goods that support the social practice.

Proposition 3:

Institutions that offer external goods to OSS developers (such as firms participating in OSS and hiring developers) are judged as supportive or constraining by the developers according to the institutions' adherence to the standards of excellence defined in the social practice.

Proposition 4:

Under certain conditions, developers of OSS are prepared to sacrifice potential rewards (external goods offered by a current institution) in order to make an institution compatible with the standards of excellence defined by the social practice of OSS.

Question 3:

Why do developers sustain the social practice of OSS development?

Theoretical Conjecture 3:

OSS developers sustain the social practice of OSS development because social practice instills the motivation to uphold its standards of excellence over time.

Proposition 5:

The motivation to contribute becomes stronger during developers' tenure in an OSS community and by their contribution to the social practice.

Proposition 6:

Through sustained contributions to the social practice, developers become motivated to contribute beyond code patches to educate and help others, and take on tasks that support the internal good of the social practice.

Note: In combination with the more traditional "self-determination view", the tenets of the "social practice view" have guided my evolving understanding of SugarCRM's open source community (e.g. under the process of observation and evaluation of qualitative questionnaire results).

Appendix D: SugarCRM's additional terms to the AGPLv3

This is the legal content found in the *install.php* file (among other files) of Sugar CE, which augments the standard terms of the AGPLv3 license:

```
/*********************************************************************************
 * SugarCRM Community Edition is a customer relationship management program developed by
 * SugarCRM, Inc. Copyright (C) 2004-2012 SugarCRM Inc.
 *
 * This program is free software; you can redistribute it and/or modify it under
 * the terms of the GNU Affero General Public License version 3 as published by the
 * Free Software Foundation with the addition of the following permission added
 * to Section 15 as permitted in Section 7(a): FOR ANY PART OF THE COVERED WORK
 * IN WHICH THE COPYRIGHT IS OWNED BY SUGARCRM, SUGARCRM DISCLAIMS THE WARRANTY
 * OF NON INFRINGEMENT OF THIRD PARTY RIGHTS.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
 * FOR A PARTICULAR PURPOSE.  See the GNU Affero General Public License for more
 * details.
 *
 * You should have received a copy of the GNU Affero General Public License along with
 * this program; if not, see http://www.gnu.org/licenses or write to the Free
 * Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA
 * 02110-1301 USA.
 *
 * You can contact SugarCRM, Inc. headquarters at 10050 North Wolfe Road,
 * SW2-130, Cupertino, CA 95014, USA. or at email address contact@sugarcrm.com.
 *
 * The interactive user interfaces in modified source and object code versions
 * of this program must display Appropriate Legal Notices, as required under
 * Section 5 of the GNU Affero General Public License version 3.
 *
 * In accordance with Section 7(b) of the GNU Affero General Public License version 3,
 * these Appropriate Legal Notices must retain the display of the "Powered by
 * SugarCRM" logo. If the display of the logo is not reasonably feasible for
 * technical reasons, the Appropriate Legal Notices must display the words
 * "Powered by SugarCRM".
 ********************************************************************************/
```

Note that the herein mentioned logo is covered by SugarCRM's Trademark Policy, as discussed in chapter 4.3.3. The latest version of this policy (or "Information" thereof) can be found at http://www.sugarcrm.com/about/trademark-information (at the time of writing, this web page states that it was last modified the 24th of February, 2012)[207].

[207] See also http://liveweb.archive.org/http://www.sugarcrm.com/about/trademark-information

Appendix E: SugarCRM partnership types and programs

SugarCRM's classifies and markets its partnership programs as such:

Channel and Reseller Partners[208]: A three-tiered channel and reseller program (Bronze, Silver and Gold) offers a catalog of training, marketing and sales resources. One has to fulfill a list of requirements to be eligible.

Authorized Learning Partners[209]: "Authorized Learning Partners are chosen for their ability to deliver quality training. Instructors must complete certification process where they learn SugarCRM's education methodology and course materials."

OEM Partners[210]**:** "You set the price and terms for your product while Sugar helps power your solution. It is up to you how much [..] the customer knows about the partnership. […] OEM partners can modify Sugar code while retaining your own IP, rebrand Sugar as your own and distribute Sugar code through direct or indirect channels. [sic.]"

Technology Partners[211]**:** "Technology partners integrate their existing applications and solutions into the SugarCRM platform. Technology providers from across a range of industries create and develop innovative solutions for SugarCRM customers and for promotion on SugarExchange helping drive customer adoption of the SugarCRM platform." Note: Technology partners do not need to integrate a whole solution into SugarCRM – many of the offerings on SugarExchange are connectors to external proprietary complements.

One can take part in several of these partnership programs in parallel. For example, Redpill Linpro (SugarCRM's "most important collaborator in the Nordics", according to itself) is both a Gold (reseller) partner, and Authorized Learning Partner[212]. It has also taken part in the community, and claims that it "on a number of occasions" has "delivered debugging services and new functions that were adopted as a part of the standard software" (ibid).

---

[208] *SugarCRM Channel and Reseller Partners*,
Retrieved May 28, 2012, from http://www.sugarcrm.com/partner/sugarcrm-reseller-levels-and-benefits
[209] *SugarCRM Authorized Learning Program*,
Retrieved May 28, 2012, from http://www.sugarcrm.com/partner/sugarcrm-authorized-learning-program
[210] *Powered By SugarCRM OEM Program*,
Retrieved May 28, 2012, from http://www.sugarcrm.com/partner/powered-sugarcrm-oem-program
[211] *Technology Partners*,
Retrieved May 28, 2012, from http://www.sugarcrm.com/partner/technology-partners
[212] *SugarCRM, our partner*, Retrieved May 28, 2012, from http://www.linpro.no/da/node/167