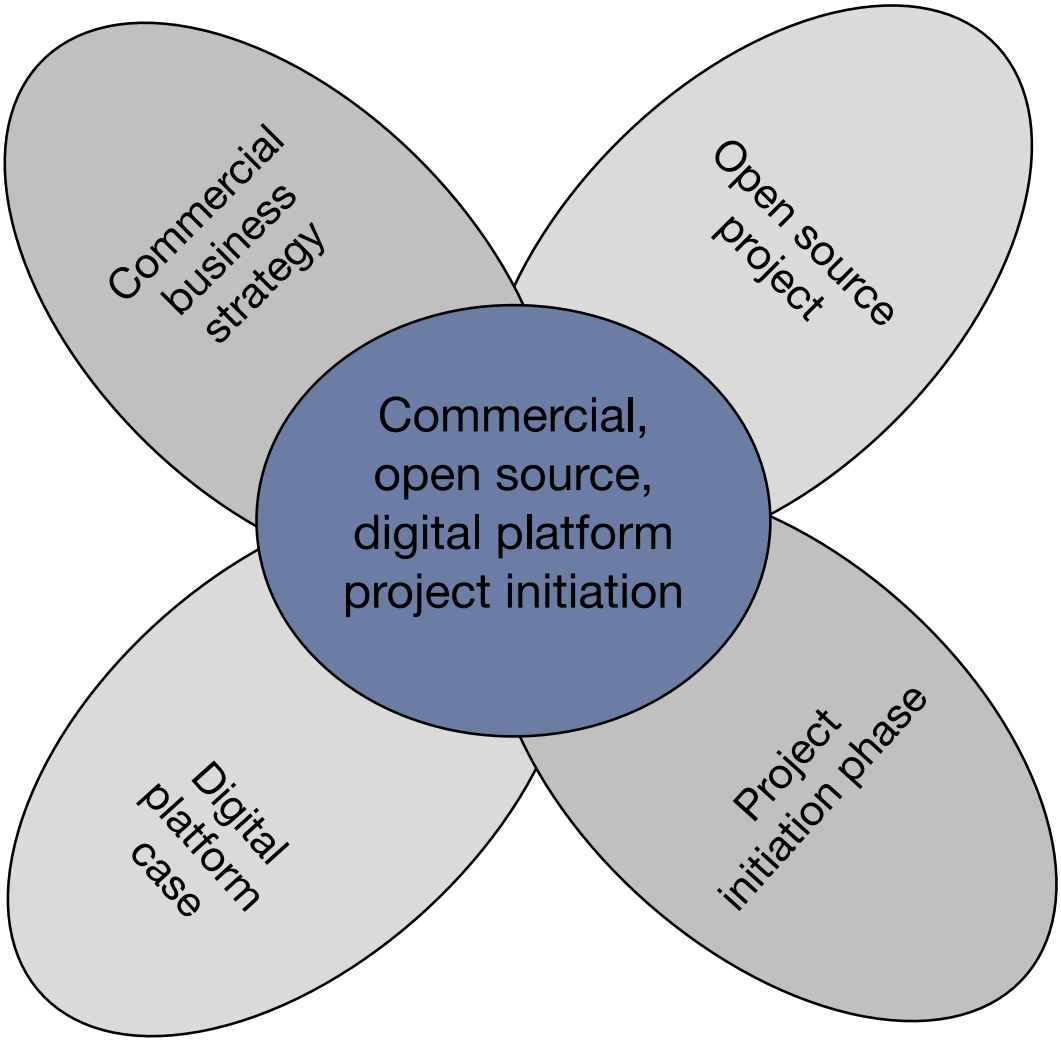
Initiating a commercial open source digital platform



Final project (15 ECTS) for
Master of IT Leadership and Management - Strategy & Architecture
IT University of Copenhagen, Denmark (ITU)

Author: Morten M. Christensen
Advisors: John Gøtze and Ahmad Ghazawneh

Date: 28 May 2016

NB: Public version with confidential case material removed!

Abstract (English)

This thesis is about starting a new commercial open source software project in the digital platform category. The focus is on project initiation (before a community of external contributors exists), where I will cover the concerns that a firm must address right from the start if the project should have a good chance of eventual success. I define success as depending on fulfilment of two conflicting goals: Obtaining a significant community of contributors to the project AND profiting from the resulting software.

For its research this thesis uses a wide range of existing literature from different academic areas and sources. For structure I use a new holistic approach looking at my research subjects from 3 perspectives: a business perspective, a community perspective and a technical perspective. A significant output of my research is a new holistic model for project attractiveness that firms can use to evaluate the ability for a project to potentially attract (rather then repel) a community of contributors.

Keywords: Open source, holistic, vendor sponsored, project initialisation, platform, commercial, monetisation, business, strategy, IP, licensing, community, motivation, self-determination theory, project attractiveness, barriers, marketing, plausible promise, MVP, software architecture, plug-ins, benefits, challenges, control, dilemmas, case.

Resumé (Danish)

Denne afhandling omhandler opstart af et kommercielt open source software projekt i digital platform kategorien. Fokus er på projekt initialisering (tiden før et fælleskab med eksterne udviklere), hvor jeg vil beskrive de forhold, som et firma skal adressere fra starten af projektet, hvis der skal eksistere en god chance for succes på sigt. Jeg definerer succes som betinget af opnåelse af 2 modstridende mål: At få et signifikant fællesskab af bidragsydere til projektet OG opnå profit fra den resulterede software.

Som undersøgelsesgrundlag bruger denne afhandling en bred vifte af eksisterende litteratur fra forskellige fagområder og kilder. Som struktur bruger jeg en ny holistisk tilgang hvor jeg ser på de forskellige emner fra 3 perspektiver: et forretningsperspektiv, et fælleskabsperspektiv og et teknisk perspektiv. Et væsentligt resultat af mine undersøgelser er en ny holistisk model af projekt attraktivitet, som virksomheder kan bruge til at evaluere om et projekt potentielt kan tiltrække (frem for at bortskræmme) et fællesskab af bidragsydere.

Nøgleord: Open source, holistisk, firma sponsoreret, projekt initialisering, platform, kommerciel, monetarisering, virksomhed, strategi, IP, licensering, fællesskab, motivation, selvbestemmelseteori, projekt attraktivitet, barrierer, marketing, plausibelt løfte, MVP, software arkitektur, plug-ins, fordele, ulemper, kontrol, dilemmaer, eksempel.

Contents

Glossary of key terms

Term	Description	References
API	API is short for "Application Programing Interface" and represents "a set of routines, protocols, and tools for building software and applications".	Wikipedia
Business model	"The rationale of how an organization creates, delivers and captures value."	(Osterwalder & Pigneur, 2010)
Commercial	"Concerned with earning money".	Merriam-Webster dictionary.
Digital platform	A digital platform (aka software platform) is "The extensible code-base of a software-based system that provides core functionality shared by the modules that interoperate with it and the interfaces through which they interoperate". Examples include operating systems such as Linux or Windows, mobile platforms such as iOS or Android and software development environments such as Eclipse or Visual Studio etc.	(Ghazawne, 2015)
Open source	A form of open innovation where a community of individual developers, organisations or firms volunteer to work together in creating software, which is shared at essentially no cost. Open source consists of a development/production model, a licensing model and a distribution model.	(Germain, 2015; Meeker, 2015; The451group, 2008)
(Open source) Governance	"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)
Participation architecture	"The socio-technical framework that extends opportunities to external participants and integrates their contributions."	(West & O'mahony, 2008)
(Platform) Ecosystem	"A collective of organizations having a common interest in the prosperity of a digital platform for leveraging their application development."	(Ghazawne, 2015)
Plug-in	A plug-in (also called extension or add-on) is an independently developed software module that provides additional functionality to a core host system such as a digital platform, application or framework. Plug-ins are similar to Apps in iOS and Android but typically less self-contained in functionality and do not necessarily have an associated user-interface.	(Lokhman, Mikkonen, Hammouda, Kazman, & Chen, 2013; Wikipedia, 2016b)
Project initiation	The first phase of a new software project. In an open source context this phase occurs before the project is announced to the world and before outside contributors join. For a firm, the project initiation phase are staffed by their own resources only (there is no community yet). What this phase contains is a subject of this thesis.	My definition
SDT	SDT is short for "Self-determination theory". A theory of motivation predominantly used in open source studies.	(Deci, 2012; R. M. Ryan & Deci, 2000):
(Single-) Vendor sponsored project	"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" (aka commercial open source project)	(Riehle, 2012)

1. Introduction

This thesis is about project start of an **open source digital platform** viewed from a **commercial** angle.

Open source is a form of **open innovation** where a community of individual developers, organisations or firms volunteer to work together in creating software, which is shared at essentially no cost. Prominent examples of large and successful **open source software projects** include the **Linux** operating system, the **Apache** web server and the **Eclipse** Integrated Development Environment (**IDE**).

In the physical world, platforms are something to stand on or build on. In a similar way, a **digital platform** (aka **software platform**) provides common functionality in the form of a common code base (aka core or foundation) that can be extended with new software **modules** that provide new functionality (Economist, 2014; Ghazawne, 2015). Besides being open source, Linux, Apache and Eclipse are all examples of open software platforms that a user can extend with new functionality coming from any 3rd party. The new functionality comes in the form of new applications/drivers in Linux, new modules in Apache and new **plug-ins** in Eclipse.

In opposition to open source, a traditional business model used by vendors in the software business is based on **closed innovation**. Internally created **closed-source** (aka proprietary) software that is licensed to customers for a fee. Strategically and in regard to **economic rents**, this can be an attractive[1] business model for software firms because **proprietary software** is generally well suited for high yield **differentiation** strategies and because of the good **defensibility** of software products derived from legal protection of source code for software that is secret, expensive and time-consuming to reproduce for a competitor.

While closed innovation has **strategic** and **monetary** benefits, it can also take an undue toll on a firm's finances and resources. A small start-up company may not have the time, resources or technical skills to develop (or buy) all its software. Even for established, well-financed companies, some software projects are so massive measured in man-hours or require so diverse a set of capabilities to complete, that few (if any) companies can finish them by themselves. As exemplified by Vice President, Irving Wladawsky-Berge from IBM, "a skyscraper is never built by a single company. Legions of small companies with specific expertise work together under the guidance of a project manager to coordinate and execute their specific tasks in the right order" (Merrill, 2011).

For a firm, commercial benefits of going open source may include increased rate of **diffusion** by the market as well as sharing the costs of development and maintenance (Stürmer & Myrach, 2015; The451group, 2008; West, 2003). Creating an open source project can be a way to leverage a community for both increased adoption and for vast outside investments in excess of its own initial investment. For example, IBM as the original creator of the Java-based Eclipse IDE was able to amplify an initial internal investment of $40 million to a community investment of estimated $1.7 billion by making Eclipse open source and attracting outside innovators (Stürmer, 2009, p. 28). By 2005, after 4 years of being open source, adoption of

[1] According to Michael E. Porter's generic strategies model and five forces framework.

Eclipse by users had skyrocketed, changing its market position from an unknown among multiple proprietary incumbents to being the dominant Java IDE on the market (Geer, 2005).

For a firm, the main commercial challenge of open source seems to be related to strategy and economic rents. On the surface, it appears challenging for a software vendor to **monetise** something that is free, and even more difficult to get attractive economic rents from investments without barriers for competitors that are all providing the same undifferentiated software.

The reality is more complex, though, as I will cover in this thesis. For the well-informed and well-prepared firm, there are various ways to monetise open source software, differentiate and build competitive advantages. As will be discussed, the real challenge is to actually grow a thriving **open source community** of significant size and to balance the need of that community with the strategic/monetisation interests of the project-initiating firm.

2. Problem formulation

The motivation for this thesis is partly born out of intellectual curiosity on the subject matter and partly because of a concrete case. The case, which will be summarised in chapter 5, serves as context and a concrete example only. The thesis is not a case study as such.

Generally, I believe that a company **sponsoring** a new **open source** based product has two essential needs that must both be fulfilled as a condition for **commercial** success:

1. A viable **open source community** for the open source project.
2. **Profit** from the resulting open sourced software product.

As it turns out, the critical phase in an open source project, that determines a firm's ability to pursue both the above goals, is **project initiation**. More precisely the initial phase before the project is even announced to the world and before outside contributors join. Decisions made at this initiation phase are deterministic for the ability to attract a community and deterministic for which **business models** and which sources of **revenue** are possible later on.

Therefore, the focus of this thesis is what a firm must do in the project initiation phase of an open source project. In addition, I want to limit the scope of products to one specific category that is **digital platforms** as in my case. Hence, the generalized **research question** is:

- **What are the general concerns for a firm when initiating an open source digital platform project?**

2.1. Delimitation

This thesis is focused on the project initiation phase. Consequently, this thesis will not look at how to actually manage an open source project or produce its software.

The dominant theme of this thesis will be open source. In that regard, the topic of digital platforms plays a more limited supporting role as the product category used in my case.

I focus on code contributors, but open source communities have many forms of contribution beyond coding, such as testing, localisation, documentation, evangelism, marketing and even financing[2] (OSSS.io, 2014a)

A specific case with a product idea is provided as in in chapter 5. It is intended for context and example only. I will not analyse the product/business of the case strategically etc.

There are many subjects that can be discussed regarding business strategies but only those that are strongly related to commercial open source and digital platforms will be covered in this thesis. Consequently, this thesis will not look at general elements of strategic planning such as for example SWOT-models, industry positioning, competitors, company resources, detailed business plans etc.

[2] For the Mozilla Firefox browser project, the community collectively financed a large ad in New York Times at the launch of the product (OSSS.io, 2014a)

3. Approach and methodology

This study will answer the research question using existing literature and a **holistic** approach involving a wide range of topics in different academic areas.

I had few pieces of relevant literature available in advance from previous ITU classes and from my advisors. Therefore, my first major objective has been to research, identify and gather potentially relevant literature.

I have prioritized academic literature, but in line with my holistic approach I also looked at references from business literature, business professionals, market surveys, analysts/researchers, venture capitalists, the open source community itself etc.

3.1. A holistic approach with three perspectives

In order to better structure and communicate my research, I have arranged the main research topics and related analysis/discussions into 3 areas or **perspectives** as shown in Table 1.

Perspectives	Main topics related to project initialization covered in this thesis
Business	Strategy, Market, Innovation, Monetisation/Economics, Intellectual property rights and Licensing, Marketing, Vendor benefits/challenges, Attraction of developers in a commercial context, Dilemmas, Digital platforms.
Community	Intro to open source and open source communities, Social aspects, Social participation architecture, Motivation of developers, Social attraction of contributors.
Technical	Technology, product, pre-community production and a "plausible promise", Minimum Viable Product (MVP), Technical software architecture for participation, Technical view of attraction of contributors.

Table 1 The three research perspectives

The exact design of the 3 perspectives are my own choice, but the overall approach is heavily inspired by a mix of the holistic approach of Enterprise Architecture (EA) and the categorisation used in Andrea Bonaccorsi & Cristina Rossi's motivation study (Bonaccorsi & Rossi, 2006).

3.2. Identifying literature

I have used a **hermeneutic method** for research of available literature starting with my initial preconceptions of open source which where translated into initial search terms for various search engines. Reading (abstracts) or browsing the literature provided me with a new understanding of the individual topics and an increased understanding of the subject area as a whole, which lead to a new iteration of search terms and new literature. E.g. forming a **hermeneutic circle** of deeper understanding and also an increased amount of material (Boel & Cecez-Kecmanovic, 2010).

As search engines I used the ITU digital library, Google Scholar, plain Google and eventually also YouTube. By combining my search terms with "open source" as a logical AND search operator I was able to get workable results from search engines with an acceptable degree of noise. The search keywords I used are shown in Table 2, which simplifies the many search iterations by collapsing them into 4 stages.

Stages	Driver	Search keywords (used in logical AND with "open source")
1st	Initial terms known to me.	Commercial, motivation, business models, strategy, community, licensing, collaboration, governance, fork, digital platform, marketing, success, failure, starting, book, thesis, survey, study, report.
2nd	New terms suggested by literature.	Monetisation, value extraction, ip modularity, dual licensing, gpl, self-determination theory, attractiveness, trust, transparency, github, bootstrapping, onboarding, network effect, open innovation, bazar, platform economics.
3rd	Information overload.	Literature review
4th	Specifics	Adoption, diffusion, statistics, eclipse, "motivation of firms", video.

Table 2 Search terms and stages

Unlike a systematic literature review my general goal has not been to objectively identify all material written – just the more recent, relevant, accessible and prevalent material… That said, in a few instances where literature is very sparse, such as project attractiveness and motivation of firms, I have attempted to identify all available literature.

Regardless of my general goal of finding just the most applicable literature, at the end of stage two, I still had about 400 literature references collected, which I tagged, scanned looking at abstracts and filtered according to apparent relevance. At this time I found that the important topic of motivation had simply too much source material for this thesis. Hence, at stage 3, I started a dedicated and successful search for literature reviews[3] looking for an existing meta-analysis of motivation studies etc.

Later, during the writing process, I occasionally needed additional specific literature outside my collection, which I added during stage 4 until I had about 650 references tagged and indexed (for searching) in my document manager[4]. Of these, I have for mundane reasons only fully read and used a fraction. Key references are listed in chapter 4 and the full list is listed in chapter 13.

[3] Which in retrospect is something I should have done much earlier as literature reviews can serve as good introductions to a research area and to its concepts. Reviews also provide a vocabulary that is useful for searching (Boel & Cecez-Kecmanovic, 2010).

[4] I have used the free "Mandeley reference manager", which has allowed me to tag and highlight documents, share them among devices and very importantly also search inside my own document base for relevant material.

4. Overview of literature and theories

Georg von Krogh and Georg Eric von Hippel write that the success of open source innovation is an eye-opener for academic researchers, similar to the discovery of life under seemingly impossible deep-sea environmental conditions (von Krogh & von Hippel, 2006).

Consequently, open source has attracted an abundance of research which they (as of 2006) divide into three categories: motivation, governance/organisation/innovation, and competitive dynamics (von Krogh & von Hippel, 2006). By including new literature I found however, that in addition to these 3 mentioned categories, there are now also significant amounts of literature that can be categorised as being about new categories such as licensing, community building and adoption by end users etc.

In some areas of open source I have found that there is very little academic literature available. Such areas include motivation of commercial firms, community marketing and what makes an open source project attractive to contribute to. Here I could find mostly non-academic literature.

Finally, in addition to academic literature mentioned above, I also found an abundance of business literature, reports from analysts, articles in popular magazines, open source insider articles, blog entries on the web and even relevant YouTube videos such as presentations or panel discussions from open source conferences.

Table 3 lists only what I consider the most important literature used in the thesis. For a complete list of my 140+ references refer to chapter 13.

Primary literature and theory (not a complete list)**:**
Case
Strategic analysis of case: • Prior ITU strategy & governance course paper "Strategiudvikling med open source" (strategy development for open source) (Christensen & Jensen, 2015).
General (all perspectives/subjects)
About open source projects and communities and touches many relevant subjects: • Book "Producing Open Source Software - How To Run A Successful Free Software Project", latest version as of December 2015 written by a well-regarded open source industry veteran (Fogel, 2015) • The paper "Cathedral and the Bazar", revision 3 of his famous/influential original essay from 1998. (Raymond, 2002). • Thesis "Open source community building" (Stürmer, 2005) • Conference panel discussion videos from OSSS.io, 2014 - Open Source Startup Summit (OSSS.io, 2014a, 2014b, 2014c) Participation architecture: • Paper "The Role of Participation Architecture in Growing Sponsored Open Source Communities" (West & O'mahony, 2008)
Business perspective
General forecasts, statistics and market research: • Study "2015 - the future of OPEN source" (North bridge and Black Duck software, 2015) and "Open source by the numbers" by Black Duck Software (Rich Sands, 2012) • Presentations and conference keynote by Forrester research about open source (Hammond, 2009, 2010, 2014a, 2014b) • Blog entries by data analyst Donnie Berkholz (Berkholz, 2013, 2014) Innovation, attractiveness and commercial side of open source etc.: • PhD. Thesis "How firms make friends: Communities in private-collective innovation" (Stürmer, 2009)

Strategy, business models: • Research rapport "Open Source – Is not a business model" and the framework "Elements of an open source business strategy" by 451 research group (Aslett, 2010, 2011; The451group, 2008) Copyrights and licensing: • The book "A practical Guide to Open Source Software Licensing" (Meeker, 2015) • Presentation "Software licences" from Bird & Bird (Harris, 2015). Marketing: • Blog posting "Viewing Communities as Funnels". (Lars Kurth, 2010) Platform: • Presentation "Digital Platforms & Ecosystems" (Ghazawne, 2015) • Presentation "Platform Shift: How New Biz Models Are Changing the Shape of Industry" (M. Van Alstyne, 2015) • Paper "Platform Strategy Survey" (Parker & Alstyne, 2014)
Community perspective
Demographics • Paper **"**Open Source Participation Behavior-A Review and Introduction of a Participation Lifecycle Model" (Ehls, 2013) Motivation: • Self-determination motivational theory (Deci, 2012; R. M. Ryan & Deci, 2000) • Meta-studies: "What Do We Know about Developer Motivation" (Hall, Sharp, Beecham, Baddoo, & Robinson, 2008) and first part of "Carrots and rainbows: Motivation and social practice in open source software development" (Krogh, Haefliger, Spaeth, & Wallin, 2012) • Paper: "Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business" (Bonaccorsi & Rossi, 2006). Attraction: • "The attraction of contributors in free and open source software projects" (Santos, Kuk, Kon, & Pearson, 2012) Participation architecture: • Conference keynote video "Open Source Community Building" (Eaves, 2011)
Technical perspective
Up-front development before going open source and MVP: • Key concept of "plausible promise" (Raymond, 2002) • Paper "Performance of Open Source Projects" (Weiss, 2009) • Summary of book "Lean Startup" (Ries & Hartman, 2011) Participation architecture: • Paper "The Impact of Modularity on Intellectual Property and Value Appropriation The Impact of Modularity on Intellectual Property and Value Appropriation" (Baldwin & Henkel, 2012) • Conference video: "Creating a Developer Community" about the Jenkins project experiences (Kawaguchi, 2012) Attraction: • Conference video and blog on "Patterns for Open Source Success" (S. R. Walli, 2013a, 2013b)

Table 3 Main literature and theory

5. Digital platform case

This chapter is confidential and has been removed from the public version. Contact the author for access to the content (may require signing a NDA).

6. Open source from a community perspective

Open source software projects are popular by users, businesses and developers. There is currently estimated to exist 1.5 million[5] open source software projects and the number of projects is growing exponentially (North bridge and Black Duck software, 2015). By 2014, 80% of active software developers of all types used open source in their work (Hammond, 2014a).

As for what open source means it depends on who one asks. Because of politics there are many possible definitions and variants. I will not enumerate the many different views here. Instead, I will loosely refer to open source as a combination of an open **development/production model**, an open **licensing model** and an open **distribution model** (Germain, 2015; Meeker, 2015; The451group, 2008).

As a development model (aka production model) open source is about developers from many places and organisations working together for a common goal. Ideally without central control or planning as further described by Eric Raymond in his influential work "Cathedral and the Bazar" (Raymond, 2002). As a licensing model, open source is about indiscriminative rights to use, examine, write and modify source code as specified in detail by the **Open Source Initiative (OSI)** (Open Source Initiative, 2007). Finally, as a distribution model, open source is about access to source code at zero costs while allowing intermediaries to build, repackage and distribute binaries for different target systems at a (nominal) charge.

6.1. Who are the open source contributors

Early on, the originators of open source and their motivations have puzzled observers such as Glass who wrote in IEEE "I don't know who these crazy people are who want to write, read and even revise all that code without being paid anything for it at all," (Ehls, 2013, p. 4 Citing Robert L. Glass, 1999)

The majority of contributors to most open source projects are **individual volunteers** (Krogh et al., 2012). Forrester has found that consistently, across their surveys, 70-75% of developers write software in their own free time and of those 24% uses their free time to contribute to open source projects (Hammond, 2014b). Their motivations will be covered extensively in chapter 6.4. Demographics vary but, in general, studies suggest that individual contributors are predominantly male aged 14-73, with a mean age of 27-32 years (Ehls, 2013). Studies also suggest that a majority have a university degree (Ehls, 2013; Ke, 2010).

Beyond individual developers, a significant and increasing minority of contributors to open source projects are from **commercial firms**. Leading IT companies such as IBM, Oracle, Google, Apple and Microsoft[6] are all involved in open source projects. 64% of large[7]

[5] As discussed in chapter 8.11, I believe that statistics like this tend to be a bit overstated.
[6] Microsoft has done a remarkable U-turn in regard to open source. In 2001 their previous CEO, Steve Ballmer, associated open source with "cancer". Now they have declared that "we love open source" (Brodkin, 2010), they contribute heavily to Linux and they release key technology of their own as open source (e.g. .NET Core, Visual Code etc.).
[7] Defined here as having over 5000 employees.

companies already participate in open source projects and 87% expect to increase their contributions (North bridge and Black Duck software, 2015). Relative participation of firms in open source projects is 40% on average. For some projects even higher, like the Linux kernel where 70% of project contributions now comes from paid developers (Kroah-Hartman, Corbet, & McPherson, 2009; Krogh et al., 2012) or the Eclipse project where about 90% of changes comes from paid developers (Watson & Boudreau, 2015). Motivations of firms to contribute will be covered in in chapter 6.5.

The significant investment of firms in open source has changed the perception of open source. As the analyst firm the451group, write in their open source report: "the idea of a community of individuals sharing the development of software projects for the greater good has been superseded by the image of a community of vendor employees sharing the development of software projects to increase code quality and lower production costs" (The451group, 2008)

6.2. Types of open source communities

Open source communities can be divided into two categories: **Autonomous** (**organic**) communities founded by individuals that evolve naturally around a software product, and corporately **sponsored** (**non-organic**) communities, which are initiated, structured and managed/dominated by a single vendor (The451group, 2008; West & O'mahony, 2008). In terms of governance, autonomous projects are almost exclusively open, while firm sponsored projects are mostly closed according to research done by 451group. (Ingo, 2011)

Because this thesis has a focus on initiation of open source projects by firms for commercial purposes, my focus will be on **sponsored communities**. Examples of such corporately sponsored communities include MySQL, Alfresco and SugarCRM (The451group, 2008; West & O'mahony, 2008).

An extreme analogy of a community sponsored by a vendor is described by James Dixon, who describes the vendor's role as similar to the role of a "beekeeper tending to a community of bees to ensure that that they produce honey that can be processed and then sold to paying customers". Just as "bees can leave the hive at any time", developers can **fork** or abandon a project at any time. The vendor must therefore balance the need for monetization with keeping the community happy (The451group, 2008, p. 38). The illustrative example is however only partially correct as a beekeeper has more control over a beehive then a vendor has over a project's external community.

6.3. Social participation architecture of communities

According to Joel West and Siobhán O'Mahony contributors' participation in open source projects correlates with **openness**. They define two distinct components of openness: Transparency and accessibility. **Transparency** allows the open source community members to access source code and watch communication and discussions being made. **Accessibility** allows the open source community members to influence the project at the cost of control by the sponsor (West & O'mahony, 2008).

Architecture of participation touches all 3 perspectives of this thesis but in a social context, the architecture of participation is about open source **production processes** and the **governance** of them. For production, transparency is "the ability to read code and observe or

follow" processes and accessibility is about the "ability to change code directly". For governance, transparency is when "observers can understand how decisions are made" and accessibility is about the "ability to participate in governance" (West & O'mahony, 2008, fig. Table 2). A summary of open source governance is provided in Appendix D.

For a company sponsor, the choice of the right participation architecture depends on what benefits the company wants to gain. If it is outside contributions to development, the more open the better. If it is marketing and **adoption** benefits the company wants to gain, then openness is not so critical. (West & O'mahony, 2008)

Another more socio-technical aspect of participation architecture is that (open source) software projects should **design for cooperation** rather then collaboration as formulated by David Eaves. **Collaboration** requires different parties working closely together in a coordinated fashion and that the all parties can come to a common agreement. Eaves states that collaboration is suited to solve complex problems that no party could solve themselves, but is difficult, has high transaction costs and is time consuming. **Cooperation** is when work can be partitioned so individuals can work independently, in a simple and efficient way that has low transaction costs. (Eaves, 2011, pt. 19:30)

When architecting problems for cooperation, one accomplishes what Eaves describes as the "genius of open source" where individuals can pick up a piece of a problem, solve it, easily put their solution back into the software and make it all work - without the need for working together with other people and without asking for permission. Architecting for cooperation is what makes open source work. The alternative, that everybody works on the same thing or agrees on everything, will never work (Eaves, 2011, pt. 20:00). The technical side of how to architect for cooperation is detailed in chapter 7.2.

6.4. Motivation of individual OSS contributors

There are many definitions and no consensus in literature about the concept of **motivation**, but if a synthesis is attempted, the various definitions are according to Steers "all principally concerned with factors or events that energize, channel, and sustain human behaviour over time" (César, 2014, p. 21). Alberto César argues that (work) motivation is distinct from, but related to, (job) **satisfaction**, which can happen after an isolated action while motivation occurs before the action. Motivation is thus future oriented and is about an individual's perceptions of work and its characteristics. This is unlike satisfaction, which is about the past and is based on a broader set of elements that is not limited to the work itself. Both concepts reinforce each other. Past satisfaction form people's perception about their world-view and their prior experiences which in turn influence their future motivation (César, 2014).

Because this thesis has a focus on project initiation rather than project management, my emphasis is on motivation rather then satisfaction. It should be noted, however, that while a precise definition of the concepts of motivation and satisfaction is beneficial, much literature is rather imprecise in this regard. For instance in some literature the concept of (work) motivation overlaps with (job) satisfaction.

Regardless of the context in which software developers work, such as open source, traditional commercial setting etc., understanding what motivates developers is a key success factor for projects and companies. Only motivated people will do their best. Alberto César writes (citing

reports from Tracy Hall and others) that motivation of software engineers have the "single largest impact on productivity and software quality management" of all human aspects (César, 2014, p. 15). Conversely, lack of motivation is a major reason for software projects to fail: As Ikram Asghar and Muhammad Usman writes, "research shows that (poor) motivation is amongst the most frequently highlighted causes of software projects failure." (Asghar & Usman, 2013, p. 1)

Specifically for an open source context, Karl Fogel also stresses the importance of motivation and writes that "understanding people's true motivations will help you arrange things so as to attract and keep (contributors)" (Fogel, 2015, p. 160) and that "understanding developers' motivations is the best way—in some sense, the only way—to manage a project." (Fogel, 2015, p. 9)

6.4.1. Theoretical motivation theories

Two major types of general motivation theories exist. **Content theories**, also called **need theories**, focus on the individual's needs, their relative importance and how their goals are for seeking satisfaction of their needs. **Process theories** are about the processes of how motivation actually happens and are concerned with behaviour and dynamic components of motivation. (Hall, 2008; Soós, Takács, Krasz, & Villám, 2013). Reward-wise, process theories are mostly about **external rewards** (salary, praise etc.) while content theories are non-monetary and about a person's **internal rewards** (personal growth etc.).

An overview of the general motivation theories covered by this chapter (and this thesis) is provided in Figure 1. Shapes with dotted lines indicate that the theory will be mentioned but not detailed. Emphasis will be on **self-determination theory** (**SDT**), as this theory translates well to an open source context and is commonly used in open source motivation studies.

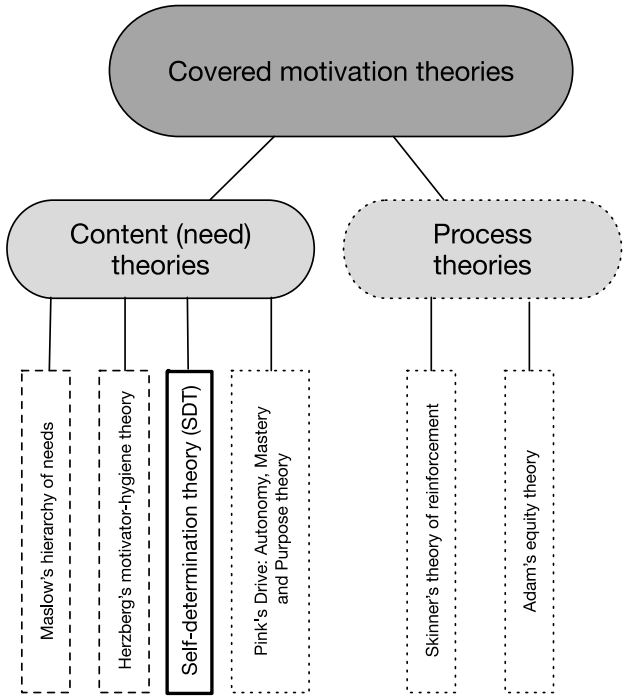


Figure 1 Motivation theories mentioned in this chapter

Self-determination theory (SDT)

The popular theory in open source studies is Edward L. Deci and Richard M. Ryan's self-determination theory (SDT). It says that we have three universal needs that when satisfied allow individuals to function and grow optimally (Deci, 2012; R. M. Ryan & Deci, 2000):

- **Autonomy**
- **Competence**
- **Relatedness**

Figure 2 SDT and universal needs (source: Wikipedia)

Autonomy is the need to control the course of our lives ourselves and to act in agreement with our feeling of self. **Competence** is the need to be able to deal with our environment and the tasks we get. It is the desire to know what actions to take and what the results of those actions will be. **Relatedness** is the need to interact with, connect to and foster close relationships with other people.

Deci and Ryan's self-determination theory distinguishes between types and quality of motivations based on what gives rise to action. From Frederick Herzberg's famous two-factors (motivator-hygiene[8]) theory they inherit the notion of intrinsic and extrinsic work. If the reason for doing work is that it is inherently interesting, enjoyable or something that you value then the motivation is of **intrinsic** nature. If the reason for doing work is because of some favourable outcome, then the work is of **extrinsic** nature (R. Ryan & Deci, 2000).

In terms of Abraham Maslow's famous hierarchy of needs theory[9], **extrinsic motivators** are generally those addressing the basic 3 categories of needs (physiological/basic survival needs, safety/security needs and the need for belonging and love). **Intrinsic motivators** are generally those addressing the top 2 categories of needs (self-esteem and self-actualization)

Intrinsic motivation is the truest form of **autonomous motivation**. An individual performs a task of his/her own choice because he/she finds it interesting/challenging/fun and he/she values what needs to be done. Unlike **externally controlled motivation**, autonomous motivation is **authentic** in the sense that it is self-authored/endorsed. Autonomous motivation promote creativity, problem solving, persistence, positive performance and physical and psychical health (Deci, 2012; R. M. Ryan & Deci, 2000).

The concept of **internalization** describes how motivation can range from being unwilling (not motivated) to compliance and to being truly motivated. Increasing internalization means increasing motivation and commitment leading to increased persistence and engagement.

---

[8] While the classification of intrinsic/extrinsic motivation is the same as Herzberg's, SDT has no concept of motivation versus hygiene factors. These are interesting concepts but has failed to be validated or invalidated by studies (Christina M. Stello, 2014).

[9] Maslow's popular theory is mentioned here to put extrinsic/intrinsic motivation into perspective only. Empiric research has failed to support Maslow's theory with clear and consistent support (César, 2014)

Table 4 shows a summary of the types of motivation in SDT, how the person affected experiences them and (in my own terms) both their internalization strength and their corresponding long-term effectiveness as a motivator. For an explanation of the different types of extrinsic motivation refer to Appendix E.

Types of motivation	Experienced as	Internalization	Long term effectiveness
A) Extrinsic motivation by			
- *External regulation*	Controlled	Lowest	Most ineffective
- *Introjection*	Semi-controlled	Low-medium	Ineffective
- *Identification*	Semi-autonomous	Medium	Semi-effective
- *Integration*	Autonomous	Medium-high	Effective
B) Intrinsic motivation	Autonomous	Highest	Most effective

Table 4 Types of motivation in SDT and their effectiveness

Fulfilment of the universal needs for Autonomy, Relatedness and Competence promotes internalization. In particular, autonomy plays a vital role. Without autonomy, regulations can not be integrated but at most just introjected (R. Ryan & Deci, 2000).

One cannot simply motivate other people, only provide **autonomy support** that may help with self-motivation: "Don't ask how you can motivate others! Ask how you can create the conditions within which others will motivate themselves". (Deci, 2012, sec. 13:20).

A way to promote autonomy is to take the perspective of the individual, providing people with a choice and engaging them, encourage people to initiate action and provide a meaningful rationale for decisions, so that individuals can adopt a value/belief. Also useful is facilitating communications and feedback that promote feelings of competence during work without forfeiting self-determination (R. M. Ryan & Deci, 2000)

Extrinsic incentives and rewards have positive effects only for manual routine-work (**quantity-typed tasks**) that has low complexity and requires little cognitive investment. For any kind of creative/conceptual work (**quality-typed tasks**) like those performed by knowledge workers or software developers, extrinsic incentives and rewards do not work. Even worse, extrinsic rewards can in many cases undermine intrinsic motivation[10] and negatively affect both quality and quantity of work (Cerasoli, Nicklin, & Ford, 2014; R. Ryan & Deci, 2000)

Deci and Ryan's self-determination theory have critics but their core ideas seem to be based on a substantial amount of supporting research with according to the authors "over 100 studies that confirmed and extended their findings" (Karen McCally, 2010). However, many details, like the exact relationship between intrinsic and extrinsic motivations, are still being researched and debated (Cerasoli et al., 2014). Personally, I think that SDT's focus on only Autonomy, Relatedness and Competence as universal needs is bit simplistic and I have yet to

[10] On the topic of monetary compensation and inner motivation, Daniel H. Pink argues that higher payment than what is needed to "take the issue of money off the table" will lead to poorer internal motivation and hence worse performance (Pink, 2010)

read a convincing argument for why there should not be other significant needs in play as well.

A notable and well-communicated variant of SDT comes from the popular book ""Drive: The Surprising Truth About What Motivates us" by Daniel H. Pink. The author has the same notion of intrinsic and extrinsic motivation but diverges from SDT by designating **Autonomy**, **Mastery** and **Purpose** as the 3 universal needs. In Drive, purpose is a common goal that individuals can identify with and strive for in order to be part of a larger cause beyond themselves. Similar to competence in SDT, Mastery is the urge to improve skills and understanding, to get better and to do one's best (Hoerr, 2010; Pink, 2010). Pink's book underscores that SDT's exact universal needs can be challenged while still adhering to the overall premise.

In an open source context, the theory suggests that open source projects may be very motivating to the individual developers that participate precisely because of the strength of personal motivations and pure autonomy. In the words of open source veteran Karl Fogel, "people are much more successful when they have their own motivations for wanting to succeed than when they are merely fulfilling management requests in return for a paycheck" (Fogel, 2015, p. 160).

Process theories

The process theories operate with payment, favourable consequences and other external rewards. They also appear to assume some control or leverage, as present in a typical manager/employee-relationship, which is absent from an (non-firm) open source setting where payment is zero and people are free to decide if they want to contribute or not.

Few aspects of process theories are directly transferable to an open source situation and for that reason I will only briefly mention two process theories that raise concrete points of interest.

One key process theory is B.F. Skinner's **theory of reinforcement**. This theory is essentially the controlled **carrot and stick** approach to motivation, which is in many ways the opposite[11] of what Deci and Ryan's self-determination theory (SDT) says. Reinforcement theory says that behaviour depends on the consequences of past actions. Future behaviour can be formed by applying positive or negative reinforcement like rewards and punishment (Soós et al., 2013).

In an open source context, the sole (and rather obvious) learning point from reinforcement theory is that thanking individual developers for making good contributions to a project is a good idea not only to be polite but also to increase the chance for additional contributions in the future by applying positive reinforcement.

Another process theory is John Stacey's **equity theory**, which states that people are motivated if they perceive to be treated equally and receive fair payment. People compare their contributions (input) and benefits/rewards (output) with others. If they consider their ratio of input/output to differ, they will be motivated to adjust their input (Soós et al., 2013).

[11] According to SDT, (positive) reinforcement makes sense only for quantity-typed tasks like manual routine-work.

In an open source context, the learning point from equity theory is that treating contributors equally is essential for motivation. Thus when evaluating outside contributions or making decisions in an open source project, it should be done in the open and done fairly so there is no suspicion that some contributors are treated better then others or have more say in decisions. According, open source literature stresses the need for transparency and fair voting (Fogel, 2015).

6.4.2. Studies on motivation software developers in general

In their study of motivation of software developers, "What Do We Know about Developer Motivation" (**WDWKADM**) from 2008, researchers Tracy Hall, Helen Sharp, Sarah Beecham, Nathan Baddoo, and Hugh Robinson Web analysed 92 studies on software development from 1980-2006 (Hall et al., 2008) .

The authors suggest that while developers have **characteristics** in common as a professional group, the individual developers vary in their motivations. Both individual personality characteristics and environment/context/demographics affect how strongly the general motivators affect the developer's ultimate motivation. The authors have created a model for how individual motivation is shaped. Their model is shown in Figure 3 (direct copy of their illustration), which shows how general motivators/needs at the bottom of the illustration is influenced by individual characteristics and context at the right, producing an ultimate set of motivating aspects for the individual developer at the top.

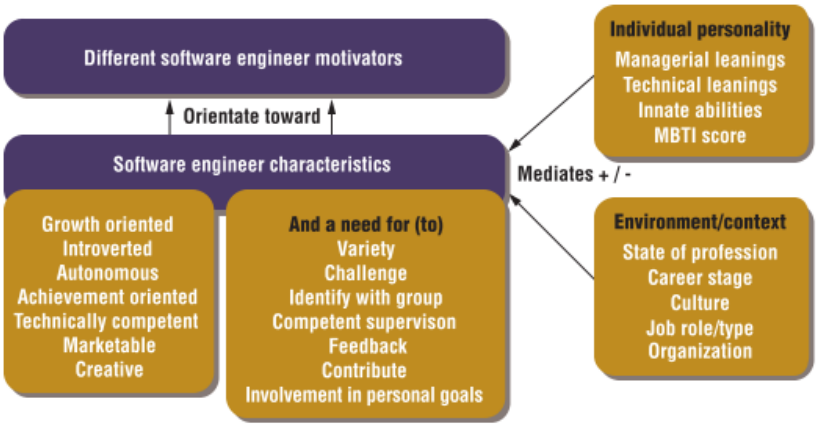


Figure 3 How individual developer motivation is shaped. Source WDWKADM (Hall et al., 2008)

The study proceeds to summarize the **motivators** from an analysis of the 92 different studies. Citing Herzberg's notion of intrinsic and extrinsic motivations, they then arranged reported motivations in these two categories. Figure 4 shows their findings (direct copy of their illustration). The term **SE** is short for **software engineering**. Some motivators are a bit vague. "Development needs addressed "indicates an opportunity to widen skills or specialize. Change means dynamic work. Beneficial means to benefit others or own well-being.

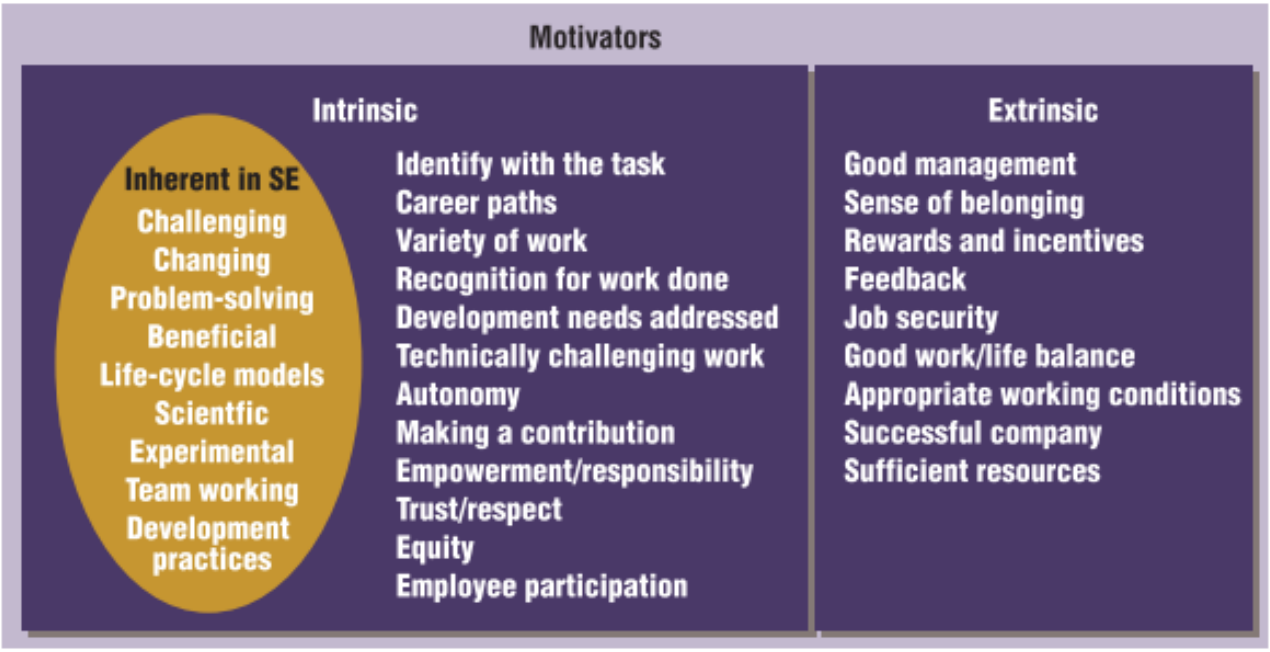


Figure 4 Motivators of software developers. Source WDWKADM (Hall et al., 2008)

For corporate settings where a developer is paid to contribute to an open source project, all the motivators from Figure 4 should be relevant.

For individual volunteer contributors to an open source project there are no bosses, no management, no payment, no job and no company involved, so some motivators from Figure 4 are not relevant. In this case, all intrinsic motivators inherent in SE and most other intrinsic motivators appear to be transferable. Only a few of the extrinsic motivators remain relevant though.

The motivating factors seem to validate SDT's identification of Autonomy, Competence (Change and Development needs addressed) and Relatedness (Team working and sense of belonging). There is no explicit mention of Purpose, though, from Daniel H. Pink's theory of Autonomy, Mastery and Purpose (although providing a benefit might indirectly provide one). The study does make it clear, though, that SDT's and Pink's sole emphasis on 3 universal factors is rather shallow as it only represents a small window from a wider range of motivating factors that influence motivation of software developers.

The **meta-study** of WDWKADM is interesting not only because it nicely accumulates findings from many prior studies, but also because it is one of the few studies I have seen which emphasises that developers have different characteristics and contexts, which strongly influence their motivation.

As an example the authors suggest that "a young developer trying to buy a house or start a family will likely be more motivated by money and less by challenge, whereas a seasoned developer established and secure in his or her job will likely be more motivated by challenge and recognition for quality work" (Hall et al., 2008, p. 93). In other words, a challenging open source project (without pay) is more likely to motivate the established developer than the young developer buying a house or starting a family.- A point which I believe has merit.

However, a key critique of the study that I can make from my own experience in the software business is that the motivating factors are rather incomplete although the study does not say so. For example, quality[12] (and technical debt) is not shown to be a factor. Another example is the size and type of audience for the software (Lerner & Schankerman, 2010; Stürmer, 2005). If the developer is using the software himself, or the software is generally useful to a great many people (incl. other developers), then the motivation ought to increase. If the software is only useful in an isolated setting that is disconnected from the personal world of the developer then the motivation ought to decrease.

6.4.3. Studies specifically on motivation of open source developers

A large number of scientific studies have been made on motivation factors of open source developers, which are based on Deci and Richard M. Ryan's self-determination theory (SDT) and its notions of **intrinsic**, **internalized extrinsic** and **extrinsic motivations** (Krogh et al., 2012).

In the first part of their paper "Carrots and rainbows: Motivation and social practice in open source software development" (**CAR-MASPIOSSD**) from 2012, Georg von Krogh, Stefan Haefliger, Sebastian Spaeth, and Martin W. Wallin performed a broad **meta-analysis** of existing literature, reviewing 40 empirical papers and summarised their findings (Krogh et al., 2012). A direct copy of their table relating motivation factors to studies is presented in Appendix A. A summary of their identified motivation factors is shown in Figure 5 (direct copy of 3rd party presentation (Stürmer, 2015)):

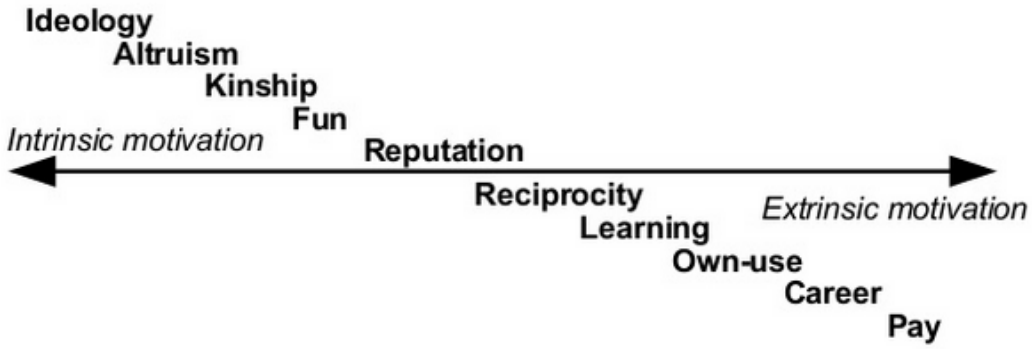


Figure 5 Summary of factors identified by CAR-MASPIOSSD study - source: (Stürmer, 2015)

The findings from CAR-MASPIOSSD vary from study to study suggesting that many factors influence motivation, depending on developer and project. The factors (explained in more detail in the above paper) are listed in Table 5. The table also indicates factors in this study that are new to open source in a sense that cannot be mapped to motivation factors mentioned in the general study of software developers (WDWKADM).

[12] In my experience, the quality aspect makes greenfield development inherently more motivating that maintaining or extending old and buggy legacy software

CAR-MASPIOSSD study motivation factors (Krogh et al., 2012):	OSS specific?	Description
Intrinsic:		
- Ideology	Y	Some developers adhere to the (rather religious) Free/Libra movement and are often driven by ideology. Ex. Believing that "running software that a user cannot inspect, modify, and share is considered immoral." (Krogh et al., 2012, p. 3)
- Altruism		Some developers are motivated by social improvement goals or selfless concern for others
- Kinship		Being part of a community motivates some developers
- Fun	Y	Some developers are motivated by an inherent fun and enjoyment of the work – it is their hobby
Internalized extrinsic:		
- Reputation		Some developers contribute because the status it gives them in the community (peer reputation) and outside the community such as perspective employers (outside reputation).
- Reciprocity	Y	Some developers want give something back to the community in return for the benefits they get (a gift in return for a gift).
- Learning		Improving their skills in the course of contributing motivates some developers. The detailed feedback that some contributions get can be a good learning experience
- Own use	Y	Some developers benefit from their contributions because they are using the software themselves and need a specific project/feature/fix themselves. By participating developers "scratch their itch" (Raymond, 2002)
Extrinsic:		
- Career		Some developers are motivated by the career opportunities their contributions may bring because of their increased experience, status etc. For them participating in open source is a way to signal their talent to prospective employers.
- Pay		Some developers are motivated because some entity is paying them to contribute to open source.

Table 5 Open source motivation factors from CAR-MASPIOSSD study.

According to SDT, when extrinsic motivations are present they ought to **crowd out** intrinsic motivations. However, so far studies (incl. the CAR-MASPIOSSD study) do not support the theory in that regard, with the sole exception that pay has been shown to negatively affect own use motivation (Krogh et al., 2012; Roberts, Hann, & Slaughter, 2006). Contrary, some

studies show that extrinsic motivation can increase intrinsic motivations (possibly as a booster/maintainer of interest) (Roberts et al., 2006).

Other studies stress that the **hybrid factors**, combinations of the motivation factors that reinforce each other, are more influential than individual intrinsic or extrinsic factors (Mair et al., 2015) Hence, motivation should be understood not as influenced either by one stable factor or another but as a "complex continuum of intrinsic, extrinsic, and internalized extrinsic motives … that evolve" as tasks characteristics change over time (Mair et al., 2015).

As shown in Table 5, the motivation factors that are stated as unique to open source are:

- Personal **ideology**
- Having **fun** developing software
- **Giving something back** to the community.
- Opportunities for using the contributions themselves (**own use**)

The study does not address relative strength of each factor and I have seen no overall consensus in the literature about which factors are most important. Instead there is a multitude of contradicting individual studies, each suggesting this or that set of factors to be most important.

Importance of own-use

From a theory standpoint, SDT suggests that of the unique motivation factors for open source, fun and ideology should be the strongest. Compared to those, own-use motivation should be a weak motivation factor. However, I believe that own-use motivation is a strong (not weak) motivation factor from an economics standpoint and based on literature by another kind of source, namely the open source community.

From an economics standpoint, motivation of contributors is derived from own-use. Making a contribution solves a **private need** that adds to the own-use value of the whole project (Lerner & Schankerman, 2010; S. Walli, 2007). Once a part of the public **code base**, future versions is likely to continue to support that private need hereby obviating a costly task of repeated **merging** of a private feature into a constantly evolving code base. As formulated by Stephen Walli: "Individual projects behave as markets from one perspective, and code is currency, the medium of exchange. Just like all economic exchanges, the contributor offers something they value less (a fragment of code solving a particular need) for something they value more ([own-use of] the functioning software package in its entirety). Nobody is working for **free** in an economic sense" (S. Walli, 2007)

Support for own-use is prominent in literature from the open source community:

- "Every good work of software starts by scratching a developer's personal itch." (Raymond, 2002, p. 2). Raymond write "personal itch", but general the "itch" can be both personal and institutional (Fogel, 2015).
- "The essential condition is that the producers of the software have a direct interest in its success, usually because they use it themselves or work directly with people who use it" (Fogel, 2015, p. 23)

Hence, I would argue that own-use (in its broadest sense that include institutional use and use by people in the immediate vicinity of a developer such as family use) is highly likely to play a

key role in an individual's choice of contributing to a project. Being a contributor bears **opportunity costs**, so there is a limit to what a developer has time to do.

Even if another motivation factor is dominant, like for example the wish to learn about new technology, there are plenty of ways to learn about new technology, which do not involve open source development. There are currently 1.5 million projects to choose from (North bridge and Black Duck software, 2015) so the choice of joining a particular project is unlikely to be random.

There must be a reason why a particular project is selected and not another. Hence I postulate that, regardless of other motivations, the reason a contributor chooses to participate in *a specific open source project* is very likely to be influenced by own-use fit and the overall **attractiveness** of that project as further argued in chapters 6.6, 7.3, 8.7 and 10

6.5. Motivation of corporate contributors

The motivations of contributing companies are very different from the motivations of individual developers. Likewise, the theories that can be used to explain their motivations are different too. They are not from motivational theory but are related to economics, business strategies and open innovation. In general, a reasonable premise is that the overall reason that commercial companies contribute to open source is for business reasons such as achieving competitive advantage (Andersen-Gott, Ghinea, & Bygstad, 2012).

Compared to the studies of individual developer motivation, there are far fewer studies on motivations for commercial companies to contribute to open source projects.

Andrea Bonaccorsi and Cristina Rossi classify motivation of both individuals and firms into the 3 main categories of economic motivators, social motivators and technological motivators. In their study, "Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business" (**CMOIPAFTTPITOSM-FCTB**), they compare the motivations of 146 Italian open source firms with individual Linux developers. Somewhat unsurprising they find that while individuals are motivated by a mix of intrinsic and extrinsic motivations in all 3 categories, companies are motivated not by social/idealistic values but by economic and technological motivations (Bonaccorsi & Rossi, 2006). Most significant motivations identified by the study were:

- Economics:
  - Independence of price/license policies of large software companies
- Technology:
  - Community contributions/feedback useful to improve software and fix.
  - Reliability and quality of open source software.

A study of 90 Dutch high-tech firms active in open source products and services offers few direct insights on motivation but suggests that firms with foreign sales and/or focus on product/service development/innovation contribute most to open source projects. They also detail that most firms combine open source and proprietary offerings. (Stam & Joode, 2007)

A small study by Morten Andersen-Gott, Gheorghita Ghinea and Bendik Bygstad of 3 commercial IT service companies, identified the 3 main drivers for contributing to open source as selling of complementary services, greater innovative capability and cost reduction

through outsourcing to the open source community (open sourcing) (Andersen-Gott et al., 2012).

In a study of relationships between open source and companies, Linus Dahlander and Mats G. Magnusson summarize literature findings about the rationale (motivation) for firms to contribute using the same taxonomy as the CMOIPAFTTPITOSM-FCTB study. They identified the following motivations theorized by literature:

- Economics:
  - Pace development and gain competitiveness
  - Using open source business models
  - Cutting cost
- Social
  - Sharing code with community
  - Perception that software want to be free
- Technological
  - Exploiting feedback
  - Diffusion and win adoption
  - Promoting standard

In summary, the limited literature and the few available studies point to a range of economic, technological and to a limited degree also social reasons for companies to contribute to open source. Rational reasons such as cutting costs, maximizing product adoption (West, 2003), selling complementary products, achieving higher degree of innovation, quality improvements etc.

6.6. Attraction of contributors from a social perspective

In the literature it is common to assume a direct link between motivation and the act of making contributions but it appears there are other **external factors** as well that encourage or constrain contributions (Krogh et al., 2012). Factors that are external to the individual developer/firm and outside their influence have an impact whether they decide to make contributions or not. This section will look at those external factors that are suggested by literature with a focus on social factors that a project initiator can influence.

Before looking at external factors it is important to note, as observed by Jason Tsay, that open source software projects and developers "operate with an unprecedented degree of **transparency**" (Tsay, 2015, p. 3). On Github, Sourceforge and similar environments along with their forums, outside users and developers can in detail observe all project **activity**, communication, team arrangement, source code and deliverables. Outsiders can also follow team members and discover who does what and what other projects each contributor is associated with (Tsay, 2015).

Outsiders can easily spot if an open source project has problems either because of a direct observation or because of missing information. An example of a **direct observation** that indicates problems could be the existence of many recent unprocessed bugs in a bug database. Examples of **missing information** include silence or lack of answers in response to questions asked in forums (Corbet, 2010). Another example is missing data on code contributions over a

significant time, which indicates the project has stopped or been abandoned[13]. As noted by Tsay, the transparency of open source projects yields a number of **cues or signals** that outsiders can use to evaluate how **mature** and **healthy** the project is (i.e. how attractive the project is). For example, by looking at the **list of contributors** and their relative activity, an observer can judge if the community is large enough to survive[14] independent of any individual or founding firm (Fogel, 2015). Such input and analyses are likely to influence developers' **decision to contribute** or not (Tsay, 2015).

In particular **activity cues** are important when judging if a project is being maintained. A sign of steady activity (like a **heartbeat**) is ideal. If a project has not been updated for a year, then most people would consider it dead and lose interest (Allen, 2015). The activity should also reflect **frequent releases**: "Regular and frequent software releases are advantageous because they implicate high activity in the community and thus progress" (Stürmer, 2005, p. 68)

It is also easy to observe from forums if a community itself is **dysfunctional** such as its members not answering questions, members being dismissive against newcomers, if the tone is not constructive or the community has "poisonous" members. All of which are **unattractive signs** of an unhealthy community[15] (Corbet, 2010; Turk, 2013). A healthy (and therefore attractive) community maintains good **communication behaviour**, is "honestly friendly" and includes a high **responsiveness** where questions and bugs are addressed quickly (Stürmer, 2005). Right from the start[16], the founding developers need to carefully set the wanted **discussion tone** and establish the right **culture**. By insisting that newcomers follow such rules, the community you get will be like-minded people (Collins-Sussman & Fitzpatrick, 2007)

The only significant literature available on external factors appears to be a narrowly scoped study on ~4500 open source projects from 2006-2008 called "The attraction of contributors in free and open source software projects" by the authors Carlos Santos, George Kuk, Fabio Kon and John Pearson. The authors classify the external factors of influence as "**project attractiveness**" which covers what **actions** a project makes and what **characteristics** a project has, that are aimed at attracting new contributors. They find that users and contributors "gravitate around only a few projects (in each application domain) because of their tendency to select the 'attractive' project" (Santos et al., 2012, p. 15). Hence their concept of project attractiveness is important as it decides which of competing projects win or lose in getting contributors in each product segment (application domain).

Key factors that the above study found to be related to project motivation include project stage (**maturity**), type of **open source licence** and **popularity** (number of hits, downloads and members etc.). Specifically for project stage they found that projects at initial planning

[13] A major concern, considering the high failure rate of open source projects as discussed in section 8.11.
[14] Related to "**Truck Factor**" or "Bus Factor", which is the number of people that can be unexpectedly removed from the project without the project collapsing (Wikipedia, 2016a)
[15] A notable community initiative that promote inclusive and healthy open source communities is the Contributor Covenant (Contributor-covenant.org, 2014)
[16] "Building a supportive Community later in the game is extremely hard - You should start day 1" (Widenius, 2016)

stages (planning, pre-alpha and alpha) impact attractiveness negatively while later stages (beta, production, mature) increasingly enhance attractiveness.

Santos, Kuk, Kon and Pearson argue that, although not examined by them, **community trust in the project and its sponsors** is likely to be key factors for project attractiveness (Santos et al., 2012) Authors Margit Osterloh and Sandra Rota support that trust is important. In their paper "Trust and Community in Open Source Software Production", they argue that new contributors will only be willing to join a trusted project. In open source they related trust[17] to having the existing community adhere to **accepted norms** of open source conduct such as **reciprocity**. **Trustworthiness** of a project requires a sufficient amount of intrinsically motivated members as extrinsically motivated members will only contribute and behave appropriately as long as it is in their own best interest. Potential new contributors can estimate trustworthiness by **observing the behaviour** of a community. For example, they can see if members offer mutual support on forums, help each other and answer questions (Osterloh & Rota, 2004). For more trust and sponsorships refer to chapter 8.7.

Related to trustworthiness, the act of proper **attribution of contributions** is important in growing a community. Project leaders should make sure new contributors to a community are recognizably successful. Then other people would like to join too (Allen, 2015)

A **fun community** that people want to be part of is also an attractive parameter. As further discussed in 7.1.1, Eric Raymond states that in order to attract a community for a project, one needs to provide in advance a “plausible promise” in form of running software (Raymond, 2002). Forrest J. Cavalier writes that the promise is not just technical but also about sociological benefits contributors will gain and about making a case that a sizable community will come to be: “What both (of Raymond’s) projects (Fetchmail and Linux) did have was a handful of enthusiasts and a **plausible promise**. The promise was partly **technical** (this code will be wonderful with a little effort) and **sociological** (if you join our gang, you'll have as much fun as we're having) (Cavalier, 1998).

Finally, some motivational incentives require an **audience**. Some developers will therefore be attracted to work on popular projects with a large number of other programmers and/or users. (Lerner & Schankerman, 2010)

[17] “**Swift trust**” to be precise, because open source participants rarely have the chance to develop “trust” in each other based on long term personal relationships and mutual control. Swift trust can be based on encapsulated interests where it is believed that it is in the best interest of the other party not to deceive. It can also be based on cognitive trust, which is estimated personal characteristics of the other person (Osterloh & Rota, 2004).

7. Open source from a technical perspective

In the technical perspective, open source initiation is about what must be produced before a project is announced (**pre-community**), how technical architecture may facilitate participation and what constitutes an attractive project in technical terms.

7.1. A plausible promise of a product and the MVP

How much software, if any and of which quality, needs to be written **up-front** by the project initiator before efforts to recruit an open source community of co-contributors can be successful? Is **running code** needed (and of a certain **quality**?) or is it enough to state a vision or design that would-be open source contributors can rally around? This is a key question, as the answer has significant impact on the project initiator firm's costs, initial resource needs and the timing on when to go open source with a new project

Unfortunately, I have found no scientific research done on this exact subject. I will therefore examine the question in two ways. First by analysing what the open source community says about this question. Secondly by involving learn start-up literature on a comparable subject, the concept of a **minimal viable product** (**MVP**).

7.1.1. Views from the open source community on up-front code in initial release

Eric Raymond is of the opinion that the project initiator must have a showcase in form of running software before releasing software as open source. The software does not need to be of particular high quality though. Raymond writes in his famous paper "The Cathedral and the Bazaar" that in order to start building a community for a project one needs to provide in advance a "**plausible promise**", elaborating, "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" (Raymond, 2002, p. 16).

Karl Fogel writes in his recently updated book "Producing Open Source Software - How To Run A Successful Free Software Project" that "There is an on-going debate in the free software world about whether it is necessary to begin with running code, or whether a project can benefit from being announced even during the design/discussion stage." (Fogel, 2015, p. 35). He states that he once thought that running code was critical to attract serious developers but has since changed his mind due to successful cases of Subversion and Mozilla that started with design documents and without running software. Ultimately, however, he does conclude that "Running code is still the best foundation for success, and a good rule of thumb would be to wait until you have it before announcing your project" (Fogel, 2015, p. 35).

Michael Weiss expands in his paper on open source projects performance patterns, on the work of both Raymond and Fogel. In his "Credible Promise" pattern, he emphasises the need to provide a core set of the functionality up-front as developers will otherwise lack incentive to join the project – not too much, though, as contributors should be left with unresolved challenges (Weiss, 2009, pp. A5–3). He concludes with the advice "Build a critical mass of functionality early in your project that demonstrates that the project is doable and has merit" with the stated sole exception that "projects without running code can attract developers when their creators have a high **reputation**" (Weiss, 2009, pp. A5–3). Citing Fogel, Weiss

elaborates “For some projects the biggest attraction for other developers is not the functionality (all that exists may be a specification), but the reputation of the project founders” (Weiss, 2009, pp. A5–1)

Matthias Stürmer quotes in his thesis an interview with Gregor Rothfuss, a core developer from the Xaraya project stating “It’s always better if there is already code available instead of just ideas. In SourceForge there are many ideas that never catch on. If some code is already available it’s much easier to find people because usually nobody has time to study ideas and plans. You can’t submit patches for ideas. There are so many other things to do, so it must be attractive” (Stürmer, 2005, p. 49).

Regarding the quality of the initial version Stürmer quotes an interview with Bernhard Bühlmann, a community member of the Plone project “You should publish a software only when its quality is sufficient” making reference to the case of Magnolia that got into a good start because developers “were immediately amazed by the beauty of the code” (Stürmer, 2005, p. 50). Compared to Raymond, this suggests a much stronger emphasis on quality.

Stürmer refers to the importance of feature strength for the initial release by referring to the success of TYPO3 with the following quote from an interview with Daniel Hinderink a marketing leader from the TYPO3 project “I believe it was the sophisticated user interface and the availability of a large range of functionality like a simple address book and a small shopping system. There were about ten plug-ins that came along with the system. Compared to other PHP projects at that time the code quality was very high. Maybe the worst example is PHPNuke, which started out small but many people joined and then control got lost. [...] TYPO3 was different because Kasper had already prepared a lot before publishing it. It was much more complete than other PHP projects (Stürmer, 2005, p. 50). Same interviewee also emphasised the importance of a stable (high quality) API in initial version (for a software platform).

Jeremy Ruston, Head of Open Source Innovation at BT and founder of the TiddlyWiki open source project, argues that without code you are really trying to form a standards group – not an open source project. Upfront code is absolutely required to get traction. Enough code so people can see and understand the vision of the project, but not so much code that it is too late for other people in the community to influence the project in unforeseen ways. (Ruston, 2007) Similar to Ruston, John Mark, open source ecosystems manager at Red Hat, stresses that outsiders are not going to contribute to projects that are too well polished: “leave some loose ends” (OSSS.io, 2014a, pt. 15:50)

Pascal Finette, formerly director of Mozilla Labs, argues that his most important learning point from the Firefox project was that “Superior Products Matter” and that “Mediocrity is boring and exhausting … people don’t care” (Finette, 2012, pt. 16:48). In the absence of massive amount of marketing (like for Coca-Cola), the only way to make people care and the only way to attract contributors to an open source project is having a great product[18]. The product must inspire people and represent something that people want to be a part of. (Finette, 2012)

[18] Great in the eyes of the community – not simply in the eyes of the inventor.

Table 6 summarizes the different stated opinions/cases regarding the requirements on code and quality of the initial release. The answers vary but most sources agree that some amount of code of undetermined quality is needed.

Summary – Opinions on strength of initial release	No quality requirement explicitly stated	Quality a stated requirement
No running code required (enough with design documents)	Karl Fogel as of 2015 (Mozilla and Subversion cases) Michael Weiss (only if creators have a high reputation)	-
Running code required – promising feature set	Karl Fogel (his previous opinion) Eric Raymond (plausible promise) Stürmer / Rothfuss (Xaraya case) Michael Weiss (critical mass but not more) Jeremy Ruston (enough code but not too much)	Stürmer/ Bühlmann (Plone case)
Running code required with strong feature set	Pascal Finette (Firefox)-	Stürmer/ Hinderink (Typo3 case)

Table 6 Summary of stated OSS opinions on strength of product in initial release

My own opinion is that context determines the need for quality and running code. I view these properties as very important, but ultimately only two of many factors that determine project attractiveness. If the project has other strong and differentiating attractors, then a concrete showcase and/or high quality may be less important than if the project has no other remarkable attractors. My view is thus a generalized form of Weiss's view that the need for running code depends on the reputation of funders (Weiss, 2009, pp. A5–1).

Lean start-up's Minimum Viable Product concept and starting an open source project

In this section I will discuss the popular theoretical concept of a **Minimum Viable Product (MVP)** from Lean Startup and assess if it might be transferred to an open source project initiation context. Can the concept of a MVP help to answer how much work should be done up-front by the project initiator in order to attract a community of contributors?

In his famous and influential book, "The Lean Startup", Eric Ries describes an iterative method that companies can use to learn how to end up with products that reflect what customers really want rather then guessing. Companies set up their products and development activities in a way so they continuously can collect and act on data/feedback from customers during development

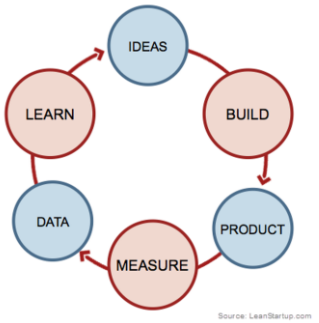


Figure 6 Lean Start-up cycle (source: LeanStartup.com)

(Ries & Hartman, 2011). The **development cycle** he suggests is illustrated in Figure 6.

The first cycle starts with a MVP, which according to Ries is the result of a minimum development effort/time that is required to jumpstart, the **build-measure-learn loop**. The goal of the MVP is to facilitate **learning** as soon as possible and thus avoid making mistakes such as producing products that customers do not want. MVP begins the learning process but does not end it. Specifically, it is the first step in continuous testing of major business hypotheses made by the creator. As such it goes beyond simply answering predetermined technical/design questions from a prototype (Ries & Hartman, 2011).

A key part of a MVP is allowing or facilitating **feedback** from potential customers, which may involve the creation of special instrumentation in the software. Apart from this measuring aspect of experimentation, the MVP should contain just enough features or promise that it makes sense for a real customer and allow that customer to start giving valuable feedback. Therefore, a MVP is not supposed to be perfect in any way nor does it contain all essential features – it does not even have to actually run as working software. Depending on context, MVP's can be simple landing pages, paper prototypes, video demonstrations, an explicit or hidden human replacement for a digital service or even early working prototypes with both features and bugs (Ries, 2011). According to Reis, there is no common rule for deciding how complex a MVP should be (although most people overestimate it): "Deciding exactly how complex an MVP needs to be cannot be done formulaically. It requires judgement" (Ries, 2011, p. 95)

Reis further argue that an MVP should care little about quality, because at the MVP stage little is known about what attributes of a product the customer values and without that knowledge quality cannot be defined: "If we do not know who the customer is, we do not know what quality is" (Ries, 2011, p. 107). He further argues that a low quality MVP does not hinder an eventual high-quality product and if customers perceive the MVP as low quality then this should be considered a learning opportunity to identify customer's quality attributes (Ries, 2011).

Iterative processes and MVP's are difficult to get right. Figure 7 is a copy of a famous conceptualization of development cycles by Henrik Kniberg, which has since gone viral, appearing in numerous places in literature and on the web. Kniberg argues that the MVP to the left, and subsequent results of each development cycle must provide increasing, useful value for the customer. Parts of a car has no value for the customer but a skateboard and various other types of vehicles has value (Kniberg, 2016).

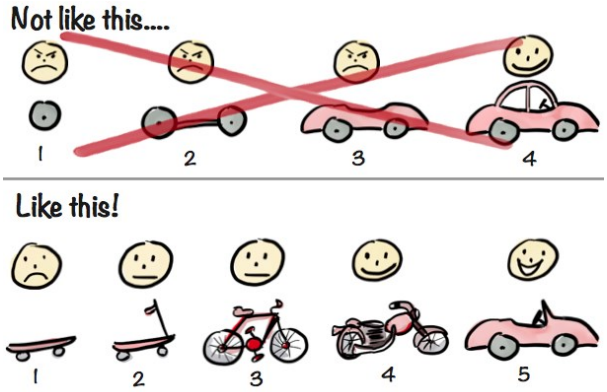


Figure 7 How to start with the right MVP ... or not (source: Henrik Kniberg)

Despite its popularity, Kniberg's illustration has one major flaw, which makes his "Like this!" example (in 2nd row) almost as problematic as the first example and serves to underscore how difficult iterative processes and MVP's can be to get exactly right. The problem is that product and customer segment change at each iteration. The customer that is in the market for a car is not likely the same customer that is in the market for a skateboard. Accordingly, product feedback from a kid using a skateboard MVP is unlikely to be relevant for a future adult making purchasing decisions on a car (Helen Walton, 2015).

Lean start-up and MVP's have attracted a fair amount of criticism which has been summed up by Jan Heitmann (Heitmann, 2014). Criticism includes:

- Essentially no real scientific evidence that the lean start-up methodology is advantageous.
- A focus on engineering over marketing, sales and revenues, which can cause a start-up to run out of money.
- Focus on early adopters that do not represent the whole market.
- A bad first impression from an inferior MVP can be difficult to work around.
- **Game changes** and highly ambitious products like planes or spaceships need to be presented in total in order to be convincing for a start-up: "*Usually, in order to change the world, you need to hit the market with force, at some point. The world doesn't beat a path to your door just because you built a better mousetrap."* (Mougayar, 2013 Attributed to Marc Andreessen).

According to established ux/designer Andy Budd, MVP can largely be seen as an approach to maximizing the interests of people that fund a start-up such as cheaply finding out if a business is viable, if there are customers that will pay money for a product etc. An MVP is not designed to maximize the interest of users/customers. Hence, the lean start-up cycle and MVPs tend to miss or impair design aspects that make products delightful and desirable: "It's really easy in minimal viable products to actually design the delight out of them**"** (Budd & Traynor, 2013)

In our open source context, the "customers" and target of the MVP are potential contributors to the software project that the project initiator needs to convince people to join the project. The price an open source contributor pays is not monetary but the time spent on the project (an **opportunity cost** (Lerner & Schankerman, 2010)). Thus, in an open source project, contributors "pay" their opportunity cost for participating in development but using the resulting product is effectively free. This is different from a typical commercial lean start-up setting, where customers pay for the product/service they receive and the start-up funders and outside investors pay for development.

Also different in the two contexts are the amount of payment and risk. What contributors are asked to risk and "pay" is much higher than what a customer is asked for in the MVP examples mentioned by Reis in his book. Relatively speaking, open source contributors are effectively asked to make a non-trivial and potentially lengthy investment of their time. It therefore makes sense that contributors' need for reassurance (measured in product features and

quality) is higher than for a customer that, based on a MVP, agrees on one occasion to try out a new product or service for a trivial monetary amount.

For all its flaws and dangers, incorporating some aspects of MVP thinking is attractive when resources are scarce such as for a small start-up initiating an open source project. In particular, I think Reis makes a convincing argument that quality is in the eye of the beholder so that optimizing early for quality makes little sense before there are real contributors and users to optimize quality for.

One should keep in mind however that the real goal is to attract contributors, which is not the focus of a MVP. For example, if one is building an open source software equivalent of a smart new car, can a YouTube video MVP or a skateboard MVP serve to persuade rational minded software developers that the project is worthy of their time and does such a MVP help instil confidence that the project is likely to succeed (and they thus get a return on their investment[19] of time and effort)?

7.2. Technical participation architecture of communities

Carliss Y.Baldwin and Kim B.Clark argue that **software architecture** of an open source project is largely decided by the initial designers of a software solution. Designers can use software architecture to make **participation** easier and therefore affect developers' incentives to contribute to a project. Specifically, they argue that software architectures that have high software modularity combined with high option value is especially suited for open source projects and will positively affect participation and effort by outsiders (Baldwin & Clark, 2006).

Modularity is about independently designed parts that work together to support a whole (Baldwin & Clark, 2006). Modularity decomposes the software into smaller **modules**, which can be understood and managed independently, along with standards for how modules interoperate. The modularity of software thus enables division of labour, reduces cognitive complexity and acts as an enabler for distributed innovation allowing different people, units or organisations to work independently on different modules of the same software product[20] (Baldwin & Henkel, 2012).

Option value is about being "tolerant of uncertainty" and welcoming to experimentation or more precisely the ability to adapt the software to new or unforeseen requirements. Options are about rights and abilities but not about obligations. Options are generated by modularity (Baldwin & Henkel, 2012, p. 1117).

A common way to provide modularity and option value in software is to support **plug-in modules** (also called **extensions** or **add-ons**) to extend a common application, framework or

[19] From the view of a contributor to an open project, the act of making a contribution can be seen as similar to making a financial **investment** in a risky business: "When developers join they need to make an investment in your project, something they lose if your project fails" (Weiss, 2009, pp. A5–3).

[20] In addition, modularity has a positive effect on quality as shown by a study of 100 open source projects that established a clear relationship between quality and high modularity (Aberdour, 2007).

platform. Plug-ins are similar to Apps in iOS and Android except that Plug-ins are typically less self-contained in functionality and do not necessarily have a user-interface in itself. To support plug-ins the product core/framework must provide a well-defined plug-in **API**, document rules for using the API correctly and provide some way to host, install and run/stop plug-ins. Plug-ins must in turn call into the common plug-in API to provide their features. For plug-ins, the application/platform/framework is in charge and the plug-ins can only do what is permitted. As for Apps, plug-ins can be added and updated at any time by 3rd parties, while changes to the underlying system core/framework/operating-system requires central planning and control.

Kohsuke Kawaguchi, the founder of the successful Jenkins project, describes plug-ins as giving each developer their own "**sandbox** to play with" and argues that plug-ins represent "the single most important requirement for building a developer community" (Kawaguchi, 2012, pt. 13:40). To be more precise, the benefits that positively affect participation according to literature include:

- Lower **barriers to entry** because a plug-in contributor need only to understand how to leverage the relatively small plug-in API – he/she does not need to understand the way the core system is implemented (Stürmer, 2005). In particular he/she does not need to understand other plug-ins. Developers are sheltered from "crappy code" of others (Kawaguchi, 2012, pt. 15:00)
- **Uncontrolled** yet simple and safe **innovation**. Plug-ins "encourages innovations without your making risky bets" (Kawaguchi, 2012, pt. 15:15) and allows simple cooperation (instead of complex collaboration) as mentioned in chapter 6.3.
  A successful example, mentioned by David Eaves, is add-ons to the Firefox browser. Before add-ons in Firefox, every individual feature required negotiation with a module owner and subsequent integration into Firefox's main source code. An elaborate process which was painful. The introduction of the add-on mechanism in Firefox turned a complex collaboration problem into a much simpler cooperation problem. By following a set of simple guidelines, add-on developers could now create and deploy new features without anyone's permission. (Eaves, 2011, pt. 21:05)
- **Facilitating** contributors with different skillsets/knowledge without compromising the quality of the underlying system core. According to Jeremy Ruston, Head of Open Source Innovation at BT and founder of the TiddlyWiki project, projects with **monolithic** architectures (without plug-ins) are often presented with patches (contributions) of low quality that cannot be integrated in a common code base (Ruston, 2007).
- People feel a sense of **ownership**/stake in their plug-in (Kawaguchi, 2012, pt. 15:00)
- Reducing the official code base of the project. Statistics show that as the code base of an open source project increases, the activity and number of contributors decrease (Rich Sands, 2012). In that regard it is therefore beneficial for an open source project, that the main core of the project stays small and code is moved to plug-ins, that do not need to be part of the main project.
- **Economics of scale** in overall software production because a fixed investment in common design decisions regarding architecture, interfaces and standards in the core can be reused across many plug-ins (Vilen, 2013)

While plug-ins constitute a safe, easy and uncontrolled place of innovation suitable for a large number of contributors, the **core software foundation** needs to be of high quality and thus

under more strict control. The core should be maintained by a much smaller set of core contributors that have all the required skills and knowledge (Ruston, 2007).

In a plug-in bases system, the increased technical complexity and governance of the core should not be a major problem, as the core is not going to attract many developers: "It is actually only a very small amount of people who typically innovate at the core... Thinking that thousands of developers are going to come and innovate your core is really not the way open source works. If you can create opportunities for people to add value through API's and through ways to contribute around the fringes (such as plug-ins) that is when you get the big multiplier effect (of open source)" (OSSS.io, 2014b, pt. 14:45 Andi Gutmans, Co-Founder Zend). Daniel Hinderink, a marketing leader from the open source TYPO3 project, voices a similar viewpoint: "[Most contributions are] definitely in the extensions. There were some donations but most of the activity is in the extension area... When … introduced into the TYPO3 project the community actually exploded" (Stürmer, 2005, p. 53).

Figure 8 summarizes how plug-ins can be used to promote participation and innovation with a wider community.

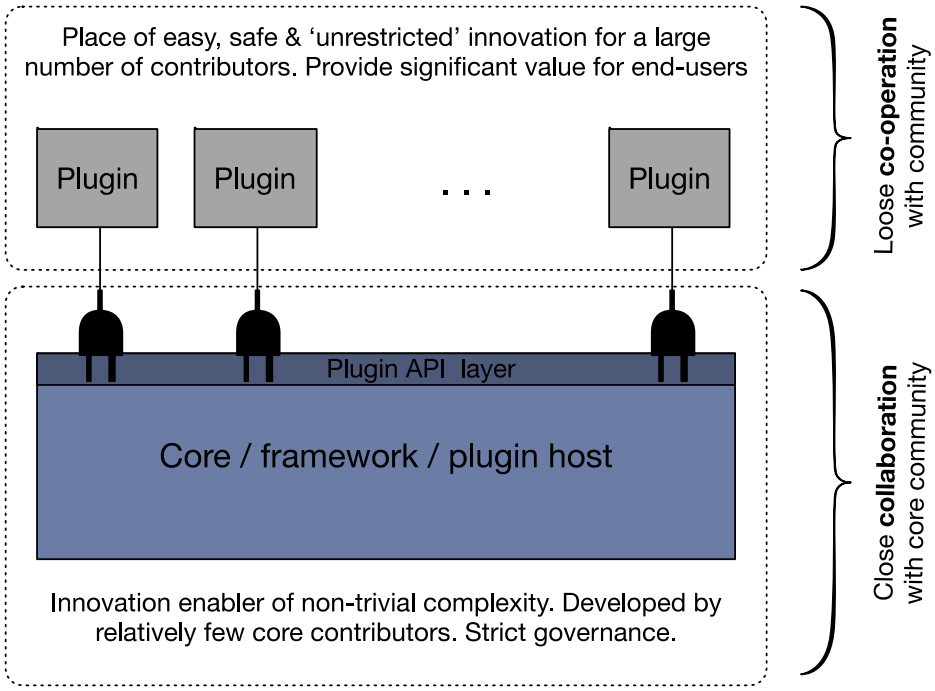


Figure 8 Plugin modularity and innovation in the community

I see one downside of plug-in modularity being that design of the plug-in API is non-trivial requiring a wide range of architecting, design, programming and **security** skills that in my experience, few software developers possess. Also, a plug-in API means that additional work is needed in form of a plug-in repository/store, plug-in update mechanism, documentation and more. I also view it difficult (in some cases even impossible) to properly isolate execution

of plug-in code so that serious bugs, misuse of shared resources and even malicious[21] code does not negatively affect other plug-ins or the core system itself. Without proper **isolation** systems are insecure, difficult to debug and won't scale safely with many plug-ins. I suspect the above downsides explain why many software projects that could benefit from plug-ins may lack plug-in support (at least initially)

For new contributors, plug-ins may also make it almost too easy to add functionality, which may result in some low-quality plug-ins. The community can help here by providing reviews and ratings etc. (Stürmer, 2005). Another potential problem with **plug-in-based ecosystems** is duplicated work and losing track of contributions. Ecosystems with plug-ins thus need a "**center of gravity**" that sustain a coherent project as a whole, enables sharing/reuse and helps attracting new core developers (Kawaguchi, 2012, pt. 15:45)

7.3. Attraction of contributors from a technical perspective

Fundamental in the discussion of the technical factors that encourage or constrain potential contributors in making contributions, is the concept of a "**plausible promise**" discussed in chapter 7.1. In other words, a convincing and working showcase of the **product vision** is part of what attracts contributors.

Unlike the details of what motivates developers, there seem to be a consensus in the literature that the **road to contributing** to a project starts with using (or evaluating) the software first (Aberdour, 2007; Nakakoji, Yamamoto, Nishinaka, Kishida, & Ye, 2002; Riehle, 2014; S. R. Walli, 2013b). As remarked by Stephen Walli, "People don't go trolling across the Internet thinking I need me an open source project to contribute to today" instead people (developers or not) are initially users looking to address a need with a piece of software (S. R. Walli, 2013b, pt. 11:28).

Related to needs, the discussion of motivation in 6.4 notes **own-use** as an important extrinsic motivation factor of an individual. However, own-use as motivation can only come into play if there is a fit between what the project's software does and what the potential contributor **needs**. The degree of fit depends on both the individual's needs and the project's software. It therefore follows that project attractiveness must be influenced by the (partly) external factor of functional (and non-functional) **fit** between the needs/requirements of the contributors and the project's existing software and/or its goals.

In addition, in the event of a fit between need and the project, I believe that the strength of the need (urgency, importance or frequency etc.) is likely to influence the attractiveness of making a contribution. I strongly suspect that open source projects that produce tools that are used daily by developers (or in production) are much more likely to receive contributions than projects with output that are useful on special occasions only. I have no direct references to literature that addresses this point in an open source context, but I do find my claim

[21] An example of security problems with plug-ins is a recent security research paper warning about Firefox add-ons/plug-ins (Buyukkayhan, Onarlioglu, Robertson, & Kirda, 2016). Firefox does not always isolate individual add-ons which is "opening millions of end users to a new type of attack that can surreptitiously execute malicious code and steel sensitive data" (Goodin, 2016)

supported by the fact that all extra large and large open source communities listed in Appendix D are of the type that are likely to be routinely used.

Because successful usage comes before potential contributions, attractiveness of a project must include concepts like **ease of installing**, **configuring**, **building** and **running** the software (S. R. Walli, 2013b). If the user finds these tasks hard it is likely he/she will give up on the project (Harding, 2014; S. R. Walli, 2013b). In a similar way to selling products, it is important to make it "super easy"[22] for people to use one's product by providing **ready to use executables** and **automated installers** that are easy to find (Kawaguchi, 2012; OSSS.io, 2014c; S. R. Walli, 2013b). In addition, user attractiveness is facilitated by **documentation** targeted at users (FAQs and How-to) along with a **communication platform** that users can go to for support and for reporting bugs. (S. R. Walli, 2016). For more coverage of user attraction from a marketing standpoint see chapter 8.8.

A related external factor to ease of using the software is how easy the project makes it for potential contributors to **experiment** with the software and to develop and deliver an actual contribution to a project. Here it is important that the **benefit** outweighs the **opportunity cost** of making a contribution: "Donators are more willing to contribute if the private opportunity costs are not too high" (Osterloh & Rota, 2004, p. 15).

In terms of construction support, the low hanging fruits of attractiveness of contributing are about **publishing** the complete **source code** in an **accessible** way and to **automate** building and testing for developers. In terms of community support, low hanging fruits of attractiveness are a clear **mission**[23] **statement**, concrete **contribution guidelines** and a useful **communication platform** for discussions, **code management** etc. (S. R. Walli, 2016). Finally, good **documentation** that supports a **gradual learning curve** has been shown to be responsible for attracting developers (Aberdour, 2007).

In relation to construction support, Kohsuke Kawaguchi argues that developers like to work in their "own **sandbox**" without having to look at the code of others (Kawaguchi, 2012). As discussed in chapter 7.2, this implies a **modular architecture** with **API's** and **plug-ins**.

Overall **quality of code** is also indicated to influence project attractiveness. On general adoption of open source, a recent survey by Forrester found better **quality software** to be the most important factor of all (user) **adoption drivers** (Hammond, 2014a). In support Stürmer quotes in an interview with Bernhard Bühlmann, a community member of the Plone project, Bühlmann stating that the Magnolia project got into a good start because outside developers "were immediately amazed by the beauty (quality) of the code" (Stürmer, 2005, p. 50). Finally, Michael Widenius, the founder of MySQL, argues that good quality creates the **confidence** in the product for people to contribute and therefore every release (even alpha versions) should be of high enough quality for people to use it in production: "People will first use your product

[22] Same as the old rule that commercial software must be easy to use to be successful: "In the early days of PC-based software (in the early '90s) there was the 5-Minute Rule (or 10-Minute Rule depending on who you consulted) that said if the software didn't install and do something useful [very] quickly it became shelfware" (S. R. Walli, 2013a)

[23] The mission statement should be carefully scoped. If it is too broad the wrong contributors will be attracted, if it is too narrow there will be little or no community interest. Non-goals are also a good way to clarify the mission (Collins-Sussman & Fitzpatrick, 2007).

and only start extending when they believe in it and it's easier to extend it for their own needs than move to something else." (Widenius, 2016)

Developers **cluster** around specific technologies/techniques/tools, often with strong feelings for/against using certain **technologies**, **tools** and **techniques** (Stackoverflow Survey, 2016). Most developers are unlikely to switch to a new programming language or technology platform just because an otherwise interesting project is using it. As remarked by Paul Glen: "Technologists are more **loyal** to their technology than they are to their industry, enterprise, manager, or group. They are attracted first and foremost to technique." (Glen, 2003, p. 20). Hence, in terms of project attractiveness, I believe technology and infrastructure choices are strong influencers.

The project must choose a suitable **project infrastructure** that can support the **social and development needs** of the community. Projects should not try to force **tools**, especially proprietary tools on the community. Instead "go where they are already" (OSSS.io, 2014a, pt. 21:18). The project initiator must "learn, live and love" what the larger community already uses (OSSS.io, 2014a, pt. 21:10). A key infrastructure choice area is **project hosting** where github.com completely dominates as the most popular hosting solution for new open source projects (Rich Sands, 2012). For communication mechanism, a **multi-channel approach** is needed. Given a varied, distributed community in different time zones "most communities require like 5 different (communication) tools" (OSSS.io, 2014a, pt. 20:50)

On the topic of technology choice, Stürmer quotes von Krogh stating "some computer languages are widespread and can attract a large number of potential contributors, while others are known by few, and thus raise contribution barriers" (Stürmer, 2005, p. 44). However, while he is supported by a majority of projects having been written in either Java or in the C/C++ family of programming languages (and increasingly also JavaScript), the usage of programming languages is increasingly **fragmented**[24] among different groups of developers (Berkholz, 2014; Rich Sands, 2012). It should be noted that programming language choice is closely associated with development platform choice.

Other technology related design choices are what tools and code libraries the project depends on. Costly proprietary **dependencies** or just a high number of complex dependencies can make it most difficult for potential contributors to develop for the project. (Corbet, 2010)

[24] In other words, some open source developers prefer **niche programming languages** compared to mainstream languages. According to my own personal experience, I find that developers that use niche languages tend to have above average skills and/or a passion for their choice that makes them willing to go to great lengths to use the language in practice (which tends to be in their free time only). Hence, I speculate that the niche language usage may represent a considerable opportunity to attract early adopter types of contributors (while being a hindrance for scaling the community of contributors later on).

8. Open source from a business perspective

8.1. The commercial market and adoption of open source

Market penetration of open source software has reached 86% of all companies in non-technical industries and probable 100% in technical industries (Volpi, 2014). Consequently, 100% pure **proprietary software** is essentially non-existing, as "all software categories use (open source) or have dependencies on it" (North bridge and Black Duck software, 2015). In particular, developers building new types of applications adopt open source to a high degree. 93% of cloud developers use open source, 92% of mobile developers and 78% of big data developers use open source (Hammond, 2014b).

With the advent of open source becoming mainstream, there is a shift towards software developers being **decision makers** in regard to software adoption in the enterprise according to Forrester: "The path from developer to customer is getting shorter…. More than ever: Developers can block – or significantly aid the adoption of software" (Hammond, 2009).

As illustrated in Figure 9, according to Forrester, the adoption of open source software by developers in the enterprise is driven by a combination of lower overall cost, **adoption speed** and integration of new capabilities. In particular, **fast acquisition** is perceived as a major benefit in enterprises since developers can use open source without prolonged contact with a procurement[25] department (Hammond, 2010). Forrester furthermore sees unusual potential for open source software to expand all three elements of the iron triangle in comparison to traditional situations where one must sacrifice either cost, schedule or capability to expand the other elements (Hammond, 2010).

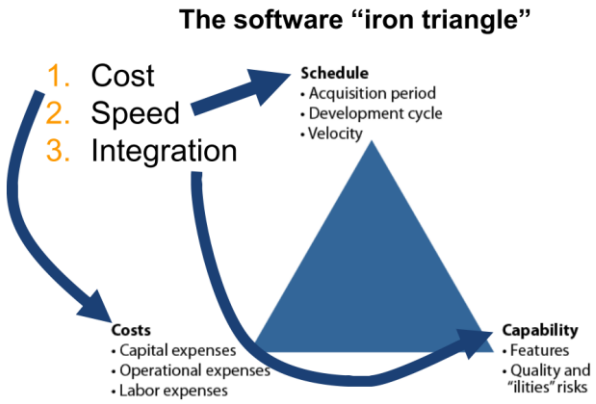


Figure 9 Why developers drive adoption of open source software – source (Hammond, 2010, 2014a)

8.2. Innovation and open source

There are two principal models of encouraging innovation in organizational science: A **collective action** model and a **private investment** model. (Hippel & Krogh, 2002; Stürmer, 2009)

In the collective-action innovation model, the output is non-rival, non-exclusive **public goods** that can be used by anyone and with the related knowledge released to the public. Public goods introduce the **free riding** problem, because 3rd-parties have the option of not

[25] According to Jeffrey Hammond from Forrester, enterprise developers often find it so troublesome and time-consuming to deal with their procurement departments that they first try open source and only go for commercial products if they can prove that all open source alternatives do not work (Hammond, 2014a).

participating in the work yet still benefit from the result of all the work of others. Therefore governments often fund public goods themselves or motivate basic research by subsidising it. Another motivator to overcome free riding, is increased reputation of contributors or a culture of **reciprocity** and knowledge (Hippel & Krogh, 2002).

In a typical private investment model, companies generate **economic rents** from their innovations through exclusive **private goods** protected by secrecy or intellectual property rights like copyright, patents, licences etc. (Hippel & Krogh, 2002; Stürmer, 2009). For a commercial software company, the 'secret source' of their business is the source code for their digital products and services. In the software business, this model is also called **closed-source** as revealing/sharing source code makes little sense when operating under a private investment model, as it challenges exclusivity and hence the basis for generating significant returns from investments.

Neither of the established innovation models above explains why a significant amount of contributions to open source comes from commercial companies (or individuals with a profit goal). Hence, the paper, "Open Source Software and the 'Private-Collective' Innovation Model: Issues for Organization Science" (Hippel & Krogh, 2002) introduces the new **private-collective** compound innovation model .

The private-collective model explains the creation of public goods, such as open source, funded by private means. The model assumes that both the public good and the creation process itself produce benefits. By participating in the innovation process, the contributors will gain **tacit knowledge/expertise**. Hence a contributor, by the very process of contributing, will benefit more than a free rider. (Stürmer, 2009)

The doctoral dissertation "How firms make friends: Communities in private-collective innovation" concludes that the success of a private-collective model requires a thriving community (Stürmer, 2009). To create a prospering community, a company must both reveal their knowledge, such as source code, and demonstrate a significant and credible long-term commitment to working with outsiders in a community. In return a company may gain an **interorganisational competitive advantage**, a "realm of friends", created by long-term network relationships with other firms and individuals that depend on mutual understanding and trust. This is an advantage, which cannot be bought and is difficult to imitate, hence it constitutes a valuable **relational asset** (Stürmer, 2009, p. 10).

Stürmer's concept of a relational asset uses a new **relational view** of competitive advantage as defined by Dyer and Singh (Stürmer, 2009) . A relational view is an alternative to strategizing about industry selection and positioning in Porter's industry-level structure view and also an alternative to the **resource-based view** where competitive advantage derives from efficient application of tangible/intangible company assets that are at best rare, valuable and inimitable. (G. Johnson, Whittington, & Scholes, 2011; Stürmer, 2009)

According to Forrester, industry innovation is shifting to collaborative developer **collectives** (aka communities) around open source. A shift where collectives act as "centres of gravity for development going forward into the next decade" (Hammond, 2014b, pt. 19:00). Enterprises can either be consumed by the generational changes that open source is part of, or restructure and learn how to work with open source communities, build on top of their work and attract talent from those communities. (Hammond, 2014b)

8.3. Copyrights and licensing of open source

Ownership of **copyrights** and the choice of open source **licence** are paramount for a company that is planning to start an open source project. They influence **community participation** and determine which business strategies and which business models are possible (The451group, 2008). After engagement with the community has started and outside contributions have been integrated into the code base, copyright law, the difficulty in contacting all involved developers and the perceptions of the community make it hard to change the license. Hence, a project initiator should "consider the choice of the open source license very carefully because later on it becomes difficult to change without losses" (Stürmer, 2005, p. 46)

Open source licenses can be divided into **permissive licences** and various degrees of **restrictive licences** (Harris, 2015). Permissive licences allow the licensor to use, modify and distribute the software in almost any way without any obligations that are significant from a business standpoint. Permissive licences are non-copyleft and allow private modifications to the code. Prominent examples of permissive licences are **MIT**, **BSD** and **Apache**.

Restrictive licences are also known as **reciprocal**- or **copy-left** licences. They use copyright law to force the licensor to make modifications of the code available under similar terms as the original. Restrictive licences can further be divided into strong copy-left licences (**GPL**, **AGPL**) and weak copy-left license (**LGPL** and **Mozilla**). In simplistic but slightly misleading terms, GPL/AGPL has been called a **viral license** because it requires the licensor to open up all their code that touches the GPL/AGPL code (Meeker, 2015). A viral requirement, which some corporate users find highly unattractive/dangerous and which in turn can constitute both a problem and leverage for the vendor.

Overall, the most popular licence is GPL when including older projects. However, its relative share has been steadily declining over the recent years as projects are increasingly permissively licenced (Aslett, 2012). On the market leading open source project site github.com (Rich Sands, 2012), which contains most new projects, the MIT license is the most common licence (Todorović, 2015).

Figure 10 shows a scale of how permissive/restrictive the 7 most popular licences are (Harris, 2015). A more detailed illustration that points out the possibilities and obligations associated with each model is provided in Appendix C.

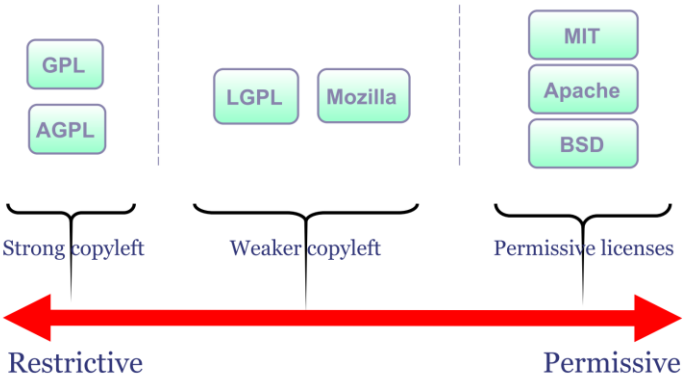


Figure 10 Open source licence spectrum – source: (Harris, 2015)

Generally, permissive/non-copyleft licences is about giving up control and instead maximize participation, innovation and the possibility for monetization by any members of the community. Starting a project under a permissive licence works well for companies that are selling complements such as hardware companies (Germain, 2015; The451group, 2008). In other commercial cases, permissive licences may be considered too much of a gift to competitors and free riders (The451group, 2008).

In opposition, strong copyleft is used to maximize control. The free software communities use strong copyleft to ensure that the project and all its deviations remain open and free. The strong copyleft also works well for commercial vendors that use strong copyleft in combination with **copyrights**[26] to make sure only they can close-source and effectively monetize the project (Aslett, 2012; Germain, 2015; The451group, 2008). For example, such a vendor can sell commercial closed-source extensions to an open source core that it, as the copyright holder, has exclusive rights to make (**open core licensing**). Another example is leveraging the fear of a viral license to sell a product under an additional and more business friendly license (**dual-licensing**). Finally, strong copyleft gives a vendor that is the copyright holder the ability to "dictate the terms by which the code can be productized by potential competitors" (The451group, 2008, p. 32).

In reality, I find it rare for a project to have all its code under a single license. Most projects use code from other open source projects, resulting in a mix of different licences or different versions of the same licence. Unfortunately, this leads to legal complexity because copyleft licences can be combined one-way with certain permissive licences but not the other way around. In addition some licences are entirely incompatible. For example, the GPLv2 licence used by Linux cannot be used together with the Apache Licence or even with the more recent GPLv3 (Fogel, 2015; Meeker, 2015).

Finally, the overall type of licence plays an important role in how attractive a project is for contributors and for the type of contributors that the project can attract. First and foremost, in order to be generally accepted as open source, the licence must be approved by the **Open Source Initiative** (**OSI**). All the licences mentioned in this section (and this thesis) are OSI approved.

Some vendors create special non-OSI approved licences, which restrict use, access or distribution of the source. The result is **gated open source,** which open source developers tend to strongly dislike. It is very rare that hobbyist contributors will contribute to gated projects, leaving need-driven developers as the only contributors to such projects (Shah, 2006)

8.4. Naming and associated intellectual property rights

Besides licensing there are other forms of **intellectual property** (**IP**) that a firm must manage or acquire during project initiation. First and foremost the firm must chose a good name for the project that can help with adoption. A good name gives a good indication of what the

[26] Copyright is here assured by having requiring contributors to sign **Contributor Licence Agreement** (**CLA**) or by not accepting outside contributions at all. In both cases, outside participation suffers (Fogel, 2015)

project does and should be a name that people can remember. In addition the name must not be taken already by another project/entity at various online services such as **social media** (Fogel, 2015).

After choosing a name, the firm should register the name at the chosen open source project host platform (e.g. github), register the associated **domain name** and register h**andles/accounts** on social media and on other relevant online services such as Twitter, Facebook, IRC etc. (Fogel, 2015).

In addition the company should consider acquiring **trademarks** for the name either for defence or for making it difficult for competitors to offer services using the name of the product.

After going public with the project, acquiring these names and rights might be too late, so it is important to do so at the project initiation phase. Note that without first acquiring names and rights, most of the marketing activities in 8.8 make little sense.

8.5. The business side of participation architecture for communities

The business side of participation architecture is mostly about **intellectual property** (**IP**) and is decided by a combination of ownership and openness by licensing. In **addition software modularity** plays a new role.

Transparency and **accessibility**, the key components of openness that determine the level of participation by outsiders, are respectively about "rights to use code and access source code" and "ability to reuse and recombine code in the creation of derived code" (West & O'mahony, 2008, fig. Table 2). **Ownership** is about copyrights to projects and potentially also derived subprojects (West & O'mahony, 2008).

In regard to **openness**, vendor sponsored projects (that are truly open source) are much like autonomous projects. However, in regard to ownership vendor sponsored projects are unlike autonomous projects in that the vendor tends to keep ownership of the core project (West & O'mahony, 2008). Dirk Riehle, states that this is an IP rights imperative: "Ensure that this firm and only this firm has the relicensing rights for the software." (Riehle, 2009).

8.5.1. Revisiting software modularity from a business standpoint

Modularity of software was discussed in chapter 7.2 as a technical way to promote participation. There are, however, business reasons for looking beyond technical aspects of software modularity and for defining module boundaries in ways that are different or suboptimal in regard to the traditional technical perspective of software architecture. These business reasons are related to strategic positioning against competitors and to IP rights.

Firstly, software modularity can make it easier for third parties to **imitate** or **substitute** individual modules which can affect a firm's competitive situation (Baldwin & Henkel, 2012). As an example, API backed modularity was exploited in the early days of MySQL to win over the incumbent mSQL database that had a position of strength because it was a part of the standard Python programming language distribution. MySQL decided to emulate mSQL's exact API used by Python. With MySQL's additional benefit of being more stable, MySQL could

then win over mSQL's users (OSSS.io, 2014c Mårten Gustaf Mickos, CEO at MySQL 2001-2008).

Secondly, software modularity can also be used to protect IP by keeping some parts open and some parts closed. **IP modularity** is when technical module boundaries are aligned with IP rights. Code that is differently open/closed or differently licenced are placed in different modules even if it is suboptimal from a technical or community standpoint. For example SugarCRM uses IP modularity to support two different configurations of their product: "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" (Waltl, Henkel, & Baldwin, 2012)

Since software architecture is difficult to change for existing software, a project initiator should thus consider IP and defensive needs when designing the software architecture. Hence, software architecture is too important to leave solely to developers.

8.6. Open source, monetisation and business models

On the importance of **business models** in an open source context, Bruno Lowagi reminds developers that "Good (open source) engineers build great technology; great engineers also create a sustainable business model." (Lowagie, 2014).

There is plenty of literature about the concept "**open source business model**" but I take the view of Mårten Mickos, Jim Whitehurst, Stephen Wallli and Matthew Aslett that there is in fact no such thing. Open source is a production-, licensing- and distribution model. It is not a business model. As exemplified by Mickos, using robots for manufacturing is a production issue that is certainly not about how value is delivered or captured by an organisation.

Monetisation of open source is essentially about **complements**. For software vendors it is not feasible to earn money from something that has been given away for free, but it is possible to earn money on something extra, a complementary good or service that, when combined with the free software, adds value to the combined offering. Hence, to monetize from open source software, something must be added to it (Aslett, 2010; Germain, 2015). Frequent examples of **complementary offerings** include additional software, additional hardware and support services.

As illustrated in Figure 11, Matthew Aslett, analyst at the451group, has created an interesting visual framework for analysing strategies of open source software vendors, which he has applied to strategies of 300 software vendors (Aslett, 2010, 2011). Detailed definitions of the terms used in his model are provided in Appendix B.

While not being as complete in coverage of elements of a business model as Alexander Osterwalder's canvas, Aslett's model does cover the open source aspects of business models very well and in much more detail then Alexander or any alternatives that I could find in academic literature.

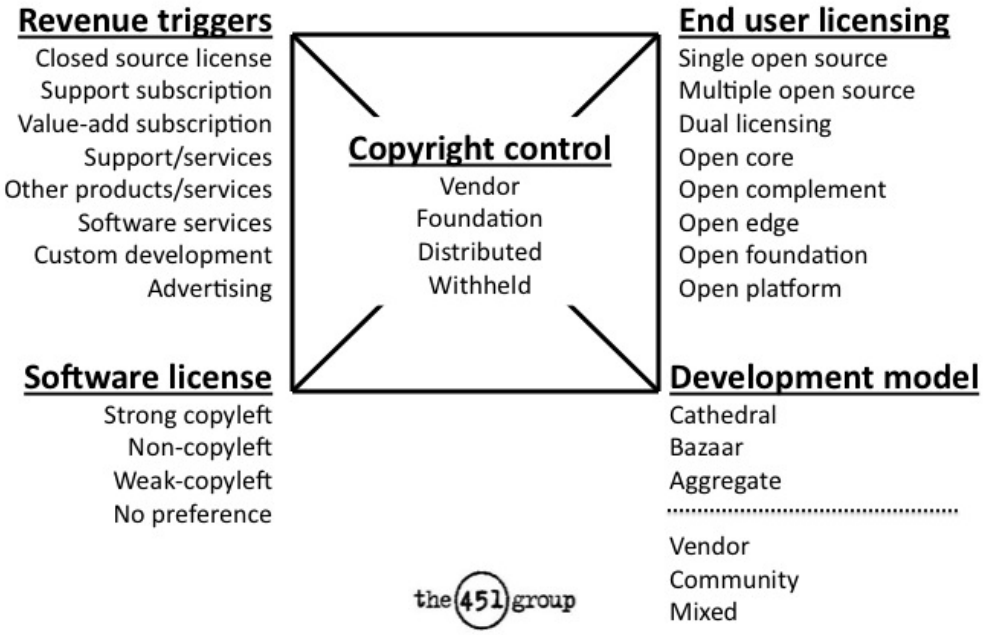


Figure 11 Elements of an open source business strategy – source (Aslett, 2011)

In Aslett's model, the upper triangle *(Revenue triggers - Copyright control - End user licensing)* drives **customer relationships** and the lower triangle *(Software license - Copyright control - Development model)* drives **community relationships**. According to the topic of this section, I will focus on the upper triangle.

Revenue triggers are the causes for "for users to hand over money in return for goods and services not available with the open source code itself" (The451group, 2008, p. 8) and include for example support subscriptions, a paid digital service built on the open source software or complementary products (other products/services).

Copyright control specifies who owns the full copyright of the code, which is important as ownership specifies who can make licensing decisions, which again are the key to any monetization strategy. Examples include vendor owned copyrights and copyrights distributed among the individual developer.

Notable examples of **end user licensing** from the model are the typical open source licence (single), **dual licensing** (typical GPL + an business-friendly license) and **open core licensing** where the core project itself is open source but plug-ins or additional features are available under a separate closed-source license.

Different configurations of selected elements in each of the 5 areas in Aslett's model translates to different business strategies as illustrated in Figure 12.

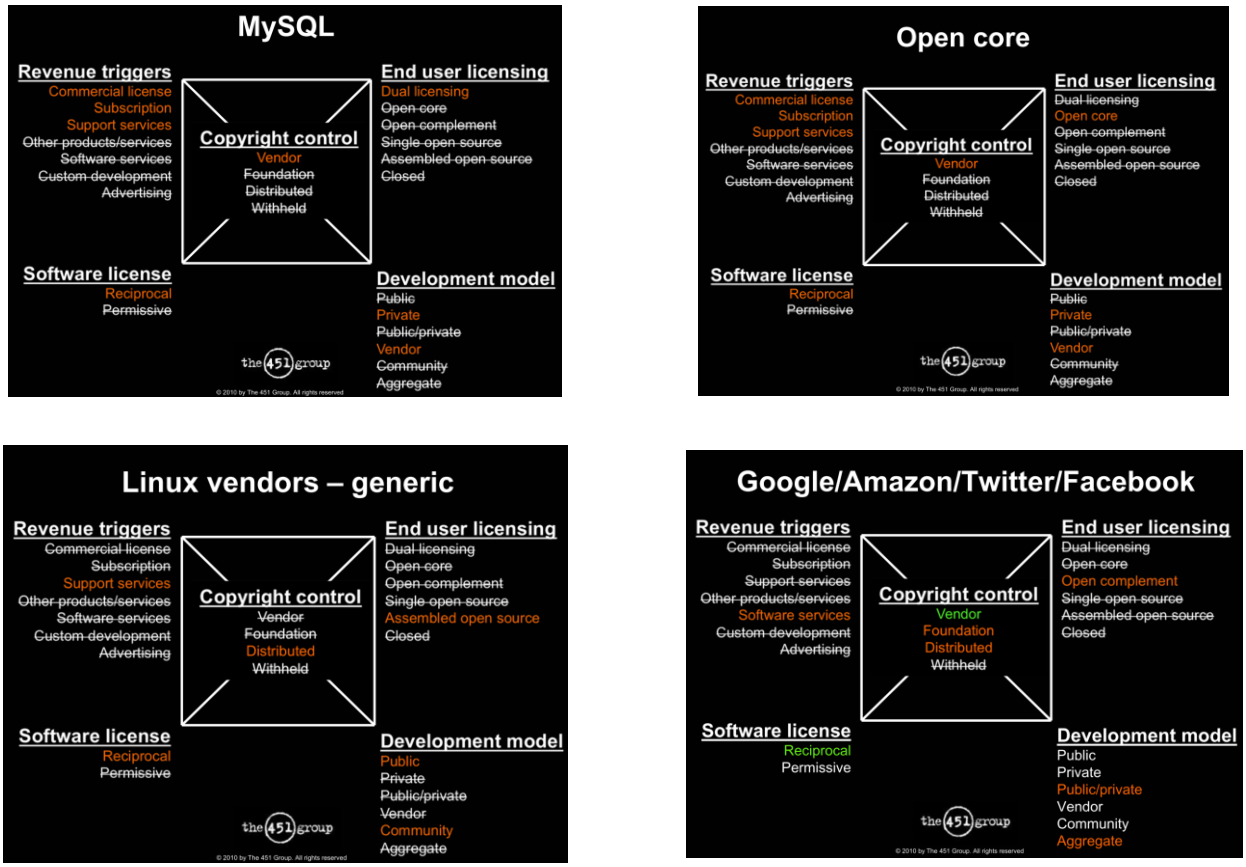


Figure 12 Examples of open source business models – source: (Aslett, 2010)

Of the business strategies shown in Figure 12, I find that the dual licensing strategy (MySQL) and the open core strategy has the potential[27] to be commercially attractive because they provide unique value to customers that competitors are unable to provide. With the right product, the right overall business model and the right execution, these strategies should be able to yield attractive economic rents (assuming they can attract the community they need).

My opinion is shared by industry veterans such as Mike Olsen who says "We have to have reasons for (customers) to buy our products uniquely" (Olson, 2013) or Mårten Mickos who is of the opinion that "most companies have concluded by now that you must have some features that only the paying customers can obtain" (Mickos, 2011, pt. 01:22) or Mike Volpi who argues that "The tenet of open source has always been to give away the "open core" for free, and then charge for additional features." (Volpi, 2014)

By contrast, the generic Linux vendor support strategy is commercially unattractive as it suffers from minimal **product differentiation**, which in line with Porters business strategy thinking of "a race to the bottom" limits companies' bargaining power, negatively impacts their competitive situation and hence ability to generate significant and stable revenue. Hence, the open source support strategy generates only a tiny fraction of the revenue of

[27] To make sure would require a broader analysis that takes into account the concrete product, resources, competitors etc. As a starting point, Alexander Osterwalder's business canvas could be used.

licensing models used by commercial companies like Microsoft or Oracle. Consequently the support business model has largely been a failure beyond Red Hat and even though Red Hat is a successful company, its share of the success of Linux is disproportion small and its market cap is very small compared to proprietary alternatives. (Levine, 2014; Olson, 2013)

8.7. Attracting contributors from a business perspective

This section discusses the business factors (from an open source vendor standpoint) that encourage or constrain potential contributors in making contributions.

Ownership and **governance** models have also been found to play a significant role in literature. Projects owned and governed by independent foundations are most successful in attracting the highest possible amount of contributors, while single-vendor sponsored projects are limited in growth. A study by Henrik Ingo, detailed in Appendix D, shows that all of the largest open source projects with 1000+ contributing developers are owned and governed by an **independent foundation**. Such extra large projects are roughly an order of magnitude times larger than any of the large vendor sponsored communities (Ingo, 2011).

For governance in practice, the **transparency** about how discussions and decisions are made is particularly important for project attractiveness. Weiss notes that transparency is a way to build essential **trust** with the community (Weiss, 2009). Karl Fogel makes an even stronger argument, effectively saying that **openness** and transparency are absolute requirements: "Making important decisions in private is like spraying contributor repellent on your project. No serious contributor would stick around for long in an environment where a secret council makes all the big decisions." (Fogel, 2015, p. 27) Openness includes for example asking the community for feedback before major development decisions[28].

As for impact of **licence**, the above study found that GPL restrictions decrease attractiveness (Santos et al., 2012) but other studies dispute this (Sen, Subramaniam, & Nelson, 2009; Stewart, Ammeter, & Maruping, 2006). My own understanding is that the answer depends on whom you ask. For example it makes sense that corporate contributors will dislike GPL restrictions because it restricts their ability to monetise from the software (Aslett, 2012; Germain, 2015; The451group, 2008).

Undisputed in literature is that the licence should be truly open source. This is best assured by selecting a proven **OSI compliant licence** (Collins-Sussman & Fitzpatrick, 2007). Projects with licences that appear to be open source only on the surface, but are restricted in a way that protects private ownership and control (gated open source), have a hard time attracting a community (Krogh et al., 2012). For example, restricting source code access or making source code hard to get or hard to modify will reduce participation (Corbet, 2010).

Open core approaches, where some components are **proprietary**, means that most of the benefits of open source disappear from a community standpoint. It is also regarded as not truly open source. Open core will therefore significantly reduce the community to a small number that do not need the commercial extensions (Widenius, 2016). Similar approaches

[28] As a side benefit, such open discussions generate activity, which in turn reassure the community that the project is progressing and hence positively affect project attractiveness (Allen, 2015).

such as keeping some **proprietary rights** and doing **commercial re-licensing** schemes is also likely to negatively affect motivation (Fogel, 2015; Krogh et al., 2012).

The literature also points to the avoidance of "large amounts of **legalese**" (Corbet, 2010) and that the selected license should be clearly stated on project website and in the code (Fogel, 2015).

Trust is important for attracting contributors. **Trust** can be built slowly over time by adherence to open source norms such as reciprocity (Widenius, 2016). Another way to gain trust is the backing of a well-known and respected **sponsor**. A developer **rock-star** (thought-leader) is likely to positively influence project attractiveness (Yu, Yin, Wang, & Wang, 2014). Indeed, "For some projects the biggest attraction for other developers is not the functionality (all that exists may be a specification), but the reputation of the project founders" (Weiss, 2009, p. 1). It should be noted that not all sponsors are viewed alike and that **commercial sponsorships** (with profit motives) in some cases reduce attractiveness for some developers where **non-market sponsorships** do not (Stewart et al., 2006). Indeed for commercially sponsored projects, a survey of contributors to Nokia's Maemo project, attests that perceived corporate **creditability** (expertise and trustworthiness) is a dominant factor in affecting motivation of contributors (Stürmer, 2009).

Finally and all but ignored by academic studies I am aware of, I presume to state that it is obvious that marketing and presentation of a project must influence its attractiveness. That it is important that the target audience knows that the product exists[29] and it is important that the project presents itself in an appealing manner. As Karl Fogel writes about the presentation of open source projects: "appearances matter … people cannot stop themselves from forming an immediate first impression" (Fogel, 2015, pp. 11–12). Marketing of open source projects is examined in more detail in 8.8.

8.8. Open source community marketing

"Solving an important project with useful code is only half the battle…. It's equally important—and sometimes more so—'to convince a significant number of people that your project is the best solution to their problem'" (Asay, 2014). **Marketing** is just as useful and necessary for open source projects as for commercial businesses, services and products (Erway & Ruff, 2013).

The importance of marketing (and sponsorship) is supported by a comprehensive study of liveliness (health) of open source projects by the company Black Duck, which concluded that new projects are most likely to succeed if they "Have big backers and marketing behind them" (Rich Sands, 2012). Although important, marketing is however a lot of work that can be significantly more demanding and time-consuming then the development work (Marz, 2014; Posted, 2014)

In the project initiation stage, marketing is the key activity that is designed to end initiation and give life to the project. Unlike marketing towards potential customers, the aim with this kind of marketing is to attract and grow a community. Hence the term **community marketing** is sometimes used in literature.

[29] "If no one knows you are there, then there you are" (Erway & Ruff, 2013, pt. 02:39)

Rules and tools of traditional marketing such as for example strategic planning, (web) presence, market research, segmentation, channels, positioning, branding, alliances and networking all apply to open source marketing (Erway & Ruff, 2013). Some models may need a little adjustment though. As exemplified by Imed Hammouda, Timo Aaltonen and Petri Sirkkala the classic marketing model, **7P's of marketing mix**[30] (Product, Promotion, Price, Place, People, Process, Physical Evidence), need only trivial changes such as product ⇒ project, price[31] ⇒ license and physical evidence ⇒ perception (of the projects mission and potential) (Hammouda, Aaltonen, & Sirkkala, 2008).

Marketers should keep in mind, however, that "code is king", that the audience is fact focused and that marketing alone cannot drive acceptance (Erway & Ruff, 2013; Glen, 2003). All claims should be demonstrably true: "with open source activities, there is an unusually high quantity of people with the expertise to verify claims" (Fogel, 2015, p. 101). Marketers should also note that communities are not like markets for commercial products. Communities are more like citizens that want to influence the project, have a stake in it and ultimately identify themselves as being a part of the project. (Finette, 2012)

As discussed in chapter 6.6, the broader literature consistently points out that the path that leads to being a contributor starts with being a **user**. Hence there are 3 major objectives related to community growth that marketing must help with:

- Creating **awareness** among potential users.
- Expanding the **user base**.
- Encourage and **convert**[32, 34] developers that use the product into contributors.

The three major objectives of community marketing can be seen in the **community funnel**[33] illustrated in Figure 13. The concrete illustration originates from a presentation by Lars Kurth, but I found several other similar funnels in open source literature. Like a sales funnel, people travel through the community funnel from left to right, with most dropping out on the way. In their journey, people assume a number of roles and perform a number of activities on the way, which must be supported in order to keep people progressing towards contributing to the project. (Lars Kurth, 2010).

[30] A widely used model from 1960 that is now a bit dated, as it suffers from **push mentality**, unlike modern marketing approaches that are more about engagement. (Chaffey, 2016)
[31] I do not agree with the authors that open software has no cost. The time spent on making a contribution is a cost. Hence, I think required development time or opportunity cost (possibly in combination with license) would be a better translation for price.
[32] According to Stephen Walli, there is anecdotal evidence from several large and small projects that the **conversion rate** of users into contributors is about 1/1000: "for every 1000 user, you might see 100 bug reports, out of which 10 will send code that purports to fix said bug, out of which 1 read your coding guidelines and really did fix the bug" (S. R. Walli, 2013a)
[33] According to Lars Kurth, the funnel model is well understood by decisions makers (and most other people). The funnel is thus highly useful for pitching funds and other activities beyond sales/marketing of open source projects (Lars Kurth, 2010)

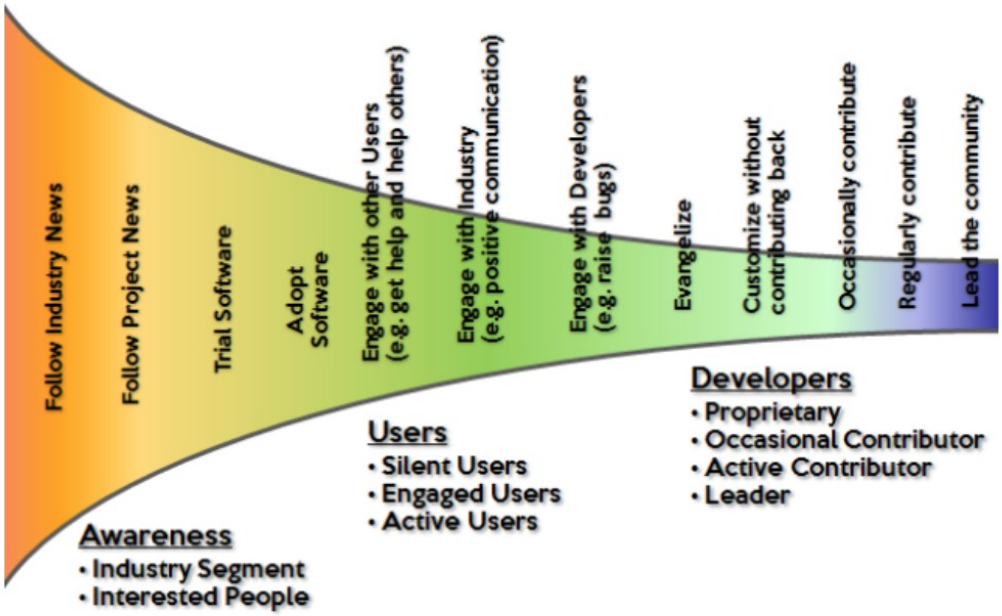


Figure 13 Communities as funnels – source: (Lars Kurth, 2010)

Marketing must identify the concrete **activities** that people go through and the concrete actions that the project can take to support those activates according to the project goals and resources. For example at the **awareness stage**, people can be supported by social media buzz or by project presence at industry conferences while documentation and usability can support people at the user stage. Outcomes of support actions can be measured and the individual **metrics** (like for example media coverage, downloads or answered/unanswered forum questions) can be combined to communicate health of the community and the effectiveness of the marketing approach (Lars Kurth, 2010).

The specific marketing activities depend on the project, but they involve improving the factors that make the project attractive as discussed in chapter 6.6. Generally, an early marketing activity in a project's initiation phase is the creation of informative and visually appealing **documentation** of the project: "The (project) site's appearance signals whether care was taken in organizing the project's presentation… The mere presence of certain standard (documentation) offerings, in expected places, reassures users and developers who are deciding whether they want to get involved. By giving off this aura of preparedness, the project sends out a message: 'Your time will not be wasted if you get involved,' which is exactly what people need to hear" (Fogel, 2015, pp. 11–12). Importantly, the documentation should include a brief coherent project pitch and the concrete steps to "getting started" using the project (Holman, 2011)

The written documentation is **linkable**, **indexable** and **tweetable** and therefore a highly useful basis for further marketing activities (Holman, 2011), including key and inexpensive activities such as:

- Ensuring **searchability** by making sure search engines can find the project and its appropriate landing page. Here open source projects has an advantage in open source directories which can help decidedly with ranking (Chalef & Mickos, 2008)
- Postings on blogs, social media, twitter etc.

People do not passively pass through the funnel. They are also looking for **interaction**. Users and potential contributors use the response from the community to assess it: "the very first touch that a person has on a project; that's them testing you… They want to see how you respond" (Allen, 2015, pt. 13:10). All questions – however trivial - signal interest in a project. That very person that asked the question may end as a contributor or leader of the project. Hence project members should prioritise to answer questions and should be welcoming to newcomers. Some newcomers may even come in angry, frustrated annoyed because of bugs. A project manager should not be upset or respond in the same way but give outlets, channel the newcomer's passion, turn it into activity and challenge the newcomer to do something productive. (Allen, 2015)

The conversion of users to contributors is far from being just about support from marketers. Conversion affects a wide range of areas that include project management, project releases and documentation. For instance, the act of delegating and substituting people is not just about "getting individual tasks done; they're also about drawing people into a closer commitment to the project" (Fogel, 2015, p. 160). Similarly, making releases are not just about adherence to existing plans but also about promoting recruitment by reprioritizing the inclusion of bug fixes provided by newcomers (Stürmer, 2005).

8.9. Benefits of open sourcing for a commercial vendor

In general, the literature points to open source as having many strategic benefits that makes it difficult for propitiatory development to compete. As stated by business professional Mike Olson: "Open source innovates faster, spreads faster, does great work faster then any single company can" (Olson, 2013).

Dirk Riehle has named a number of **business function** associated benefits for a vendor sponsored project that derive from the engagement of a user community. His benefits are listed verbatim in Table 7 (Riehle, 2012).

Riehle argues that by actively using **customer-side champions**, open source can be used as an effective sales and marketing strategy by firms. In the sales funnel for commercial open source, leads come from the existing community and the traditional "pre-sales-to-sale" activities are replaced with a "user-to-customer" conversion[34] process. Footholds by early adopters (champions) that use the free software drive customer acquisition cost down and reduce the risk of a later purchase in the eyes of the customer. (Riehle, 2012)

Business function	Benefit for vendor
Sales	More and easier sales due to customer-side champions
Marketing	More believable and cheaper marketing through engaged community
Product management	Superior product thanks to broad and deep user innovation.

[34] Actually, it is more a process of selling to customers that have the right mind set from the start. "Some people spend time to save money, some spend money to save time". The customers are in the second category. Trying to convert people in the first category can be a costly mistake. (Germain, 2015 Attributed to Mårten Mickos, PARC Forum Talk, 2010)

Engineering	Superior product that is developed faster thanks to fast and immediate community feedback.
Support	Lower support costs thanks to self-supporting user community

Table 7 Benefits per business function for single vendor commercial open source – source: (Riehle, 2012)

For the sales and marketing functions, other sources generally state a increased **technology diffusion** as a benefit of open source but without providing concrete benefits like Riehle (Stürmer & Myrach, 2015; The451group, 2008; West, 2003).

In addition to Riehle's customer-side champions, open source contributors in the community play a key role too. As contributors they have **vested investment** in the project and are likely to **advocate** for the project or at least create awareness and buzz in blogs, social media etc. As noted by Kevin Efrusy, a venture capitalist, "The (open source) developers are the people who bring you in; they're your salespeople" (Chalef & Mickos, 2008). Finally, open source users also provide marketing support as mentioned by Mårten Mickos from MySQL who refers to the user community as its best marketers (Chalef & Mickos, 2008)

As for benefits in innovation and product management, there are many sources that point to the **crowd** being able to **out-innovate** individual firms. One documented example is the group of diverse Foldit gamers that collectively solved the structure of an extremely difficult protein in 3 weeks. A problem that scientists had been unable to solve in more then 10 years (Akst, 2011)

As for the benefits in engineering, the literature points to increase **quality** and more **cost-effective** production (Monty Widenius & Nyman, 2014) here exemplified by two quotes:

- "It is often cheaper and more effective to recruit self-selected volunteers from the Internet than it is to manage buildings full of people who would rather be doing something else" (Raymond, 2002, p. 22)
- "Linus's law: Given enough **eyeballs**, all bugs are shallow.'' (Raymond, 2002, p. 6) meaning that giving enough developers and testers just about any problem will be found and fixed quickly[35].

8.10. Challenges of open source development

A mistaken view of open source is that it is an easy way to cut development costs. Instead of spending time and money on software development yourself, you just need to release an initial version on the Internet as open source and then your work is done. Once on the Internet legions of developers will soon find your code, like it, maintain it, fix your bugs and extend the code for you with great new features (that you will all appreciate).

Instead, the literature suggests that going open source is a complex endeavour that will cost more in the short term than doing the corresponding work in-house with your own resources (Fogel, 2015). Going open source is a "huge **commitment** to a process that takes a lot of cultivation and investment of time and resources" (OSSS.io, 2014a, pt. 11:40 Diane Tate, Program Manager at Mozilla). Just **community support** alone can be a massive undertaking.

[35] The problem however is that most projects are too small to have enough eyeballs (see chapter 8.11. Linux's law may work for Linux but not for open source projects in general.

For example, Michael Widenius, a founder of MySQL, "personally wrote 30,000+ emails during the first 5 years to help people with using MySQL" (Widenius, 2016)

Small companies that do not have the money to work full time on their open source offerings may have a hard time competing with proprietary alternatives (Widenius, 2016). If the product is good but the pace is too slow there is always a risk that another entity decides to create a **fork** (copy) of the project, possibly for competitive reasons (Levine, 2014). "The complexities of defining and controlling a stable roadmap versus innovating quickly enough to prevent a fork is vicious and complex for small organizations." (Levine, 2014).

For an organisation, the changes of the **software development process** from internal and proprietary to transparent and distributed open source ways of doing things are difficult and time consuming to implement: "The change from proprietary to open is about on the same order as going from waterfall to agile… Same degree of change in philosophy for developers." (OSSS.io, 2014a, pt. 15:55 John Mark, Open Source Ecosystems Manager at Redhat)

8.11. Open source project success rates

Failure to attract contributors has been a persistent challenge for most projects. In 2002 Sandeep Krishnamurthy researched the top 100 mature projects on then popular Sourceforge and found that "most OSS programs (were) developed by individuals, rather than communities." with a median number of 4 contributors per project (Krishnamurthy, 2002). In 2013, Donnie Berkholz, examined yearly contributions to 50.000 active[36] open source projects and found that 51% had only 1 contributors and 87% had 5 or fewer contributors (Berkholz, 2013). Controversially, of all active projects the percentages of successful projects with a high number of contributors were merely 1% for 50+ contributors and 0.1% for 200+ contributors (Berkholz, 2013). Finally, even for projects with multiple contributors it is common that most contributors make only one contribution, leaving the bulk of the work to just a few contributors. (Stewart et al., 2006)

Another consideration is the high rate that open source projects fail (i.e. are abandoned) which is "probably on the order of 90– 95%" (Fogel, 2015, p. 1). More specifically, a recent study of projects hosted on github.com found that 98% of projects where unmodified after a year (i.e. essentially abandoned) (Berkholz, 2014). Similar numbers were reported by a study of 550,000 projects on Ohloh, which found that more than 95% of projects were unmodified after a year (Rich Sands, 2012).

All these statistics indicate that initiating and running a successful open source project with an active community of significant size is very difficult. Something I can attest to from personal experience.

One should keep in mind, however, that developers might leverage open source infrastructure for other reasons than (seriously) attempting to launch an open source project[37]. Creating projects on open forges like in SourceForge or GitHub offer benefits like free version control and source code backup that a developer may appreciate for his/her personal home-grown

[36] I.e. excluding abandoned projects.
[37] Which might help explain why most projects released on github have no license at all (McAllister, 2013)

projects or experiments. Such projects are often not intended as serious open source projects yet affect most statistics.

In addition, the above dismal statistics should not be confused with the chance of *'winning the lottery'*. Unlike a random lottery, the project initiator has influence over outcomes and can significantly improve the chance of success (e.g. by working hard, by doing the right things at the right time and by not making costly mistakes).

8.12. The central dilemma of commercial open source

Stürmer writes that "managing a largely independent open source community is a challenging balancing act between exertion of **control** to appropriate value creation, and **openness** in order to gain and preserve credibility and motivate external contributions" (Stürmer, 2009, p. 4).

By his remarks, Stürmer introduces the central **dilemma** of commercial open source. Resolving the conflicts between the goal of making an innovation successful, for which a striving community is needed, and the goal of profiting from the success of the innovation (West, 2003). For a commercial firm success must be a combination of both. Failed innovation that could have been monetized is just as unattractive for a firm as successful innovation without the ability to monetize from it.

From an open source community viewpoint, I find that the literature points to 4 overall reasons why some individual members of a community may dislike a vendor sponsored open source project:

- The entire project may not be not entirely open source (**open core** models): The existence of **proprietary extensions** to the open source project, without which the open source version, might not be truly useful, removes much of the benefits of open source (Widenius, 2016)
- **Asymmetry** in rights to monetise (dual licenses models etc.): "only one party has that right, and other participants in the project are thus being asked to contribute to an asymmetric result" (Fogel, 2015, p. 197)
- **Closed governance** model dominated by the vendor. A majority of vendor-sponsored projects are not open enough in terms of transparency and accessibility as detailed in chapter 6.3 (Ingo, 2011).
- Clash of **ideology**: Some community members are opposed to most kinds of commercialisation with the possible exception of selling branded T-shirts, distribution of CD's, support and similar (undefendable) low-income models. An **archetype** of such ideological driver members is Richard Stallman, the influential head of the free software foundation. Stallman believes that commercialisation of software "is a crime", has contempt for closed software, is strongly against modern cloud/SaaS solutions (a trap) and strives for "truth, beauty, or justice" over personal success (Wikiquote, 2016).

Bruno Lowagie, the founder of the IText project, has a different perspective on the dilemma. He is of the opinion that "Open source can be used to create value, but that will only work if

you also sustain that value". He writes that the projects must have viable business models[38]. Viable business models that ensure jobs and the ability for the project to assign the necessary resources for continued innovation and quality control. According to Lowagie, the infamous HeartBleed security bug in the OpenSSL project was due to the lack of an effective business model that could fund the project. To avoid situations like this he recommends that users and developers "stay away from any open source project that isn't supported by a healthy business model" (Lowagie, 2014).

I think it is impossible to appease all types of community members, especially the ideological ones. That said, community must still come first[39]. Governance and the chosen business model must be acceptable to a large amount of potential contributors. I suggest an open core approach should be careful not to cripple the open source version but only add proprietary features that paying enterprises need, but the contributors do not need. I also suggest a dual-license approach should consider decreasing the asymmetric relationship with its community by sharing opportunities for monetisation with its community. Yet, another possibility would be to choose a model that is not based on licensing but SaaS (Software As A Service).

Finally, I also suggest that proper **communication** of Bruno Lowagie's key point about healthy projects needing income is a vital part of resolving the dilemma. Commercial vendors should explain to users and developers of a project upfront that there is a valid reason for creating revenue from the project, that it does not weaken the open source bits and that they will ultimately benefit in form of additional innovation and support (as opposed to a project running out of steam).

[38] Based on restrictive copy-left open source licences, unlike for example the Apache license, which is permissive. "Based on my 15 years of experience in open source, I know that it's almost impossible to create a sustainable business model based on the (permissive) Apache Software License." (Lowagie, 2014).

[39] "Foster, develop, invest in your community like it is the difference between success and failure… because it is…. If you don't grow your community, if your community isn't thriving, whatever your product is, you are going to suffer" (OSSS.io, 2014c, pt. 16:50 Mark Brewer, CEO Typesafe/Lightbend)

9. Open digital platforms, ecosystems and open source

There are two major types of **platforms** used commercially: Internal (proprietary) platforms that are used as a common foundation for a line of derived products in a single company and external (open) platforms which act as a common foundation for an industry or group of firms that can provide complementary products, extensions or services on top of the platform (Gawer, A., Cusumano, Gawer, & Cusumano, 2012). In the rest of this section (and in this thesis as a whole), I discuss **open platforms**. Such open platforms are used in many different industries but the focus here is **digital platforms** (aka software platforms) used in the software industry.

Whereas products are best described by their features, platforms are best described by their **communities** (M. W. Van Alstyne, Parker, & Choudary, 2016). An open platform is based on open standards, is openly documented and designed so the platform can be leveraged by anyone. In the software world, an open digital platform is a software platform with published API's that 3rd parties can use to extend the platform with new functionality. As noted by Forrester, API's are central to any platform: "Developers first inclination is to look for a service they can call or an API they can use" (Hammond, 2014b)

9.1.1. Platform strategy

In their book "Strategy Rules - five timeless lessons from Bill Gates, Andry Grove And Steve Jobs" (2015), authors David Yoffie and Michael Cusumano argue, using the cases of Microsoft, Intel and Apple, that singular products cannot provide a sustainable completive advantage. Instead companies should think in platforms that make it easier to sell add-on products to a *captured audience* as well as allow an **ecosystem** with 3rd parties that can contribute with their unique value. The lesson is "Build platforms & ecosystems, not just products - No firm is an 'island,' especially in technology driven markets" (Yoffie & Cusumano, 2015).

In support, Marshall Van Alstyne argues that "The Product Business Model is Broken" (M. Van Alstyne, 2015, p. 12). Digital platforms and their ecosystems with 3rd party developers are in combination able to put significant more resource into development then any product companies. Singular products maintained by individual software companies cannot compete on innovation, scale or costs. As a result the platform companies are increasingly dominant[40] in the economy. Similar to the giants of the industrial age, **economics of scale** gives the big platforms the advantage. The key difference though is that the old industrial age giants used "**supply side economics of scale**" to gain competitive advantage, while the modern open platforms use "**demand side economics of scale**" (M. Van Alstyne, 2015)

A key goal of a platform is to promote economic **network effects**, where the value of the resulting ecosystem increases with the number of participants (VisionMobile, 2013). As more and more 3rd parties contributors build solutions on top of the platform (such as plug-ins), customers will be attracted by the increasing value of using the platform. More customers increase the attractiveness of the market which in turn attracts more 3rd party contributors

[40] Measured in market capitalisation, the top 3 companies in the US (Apple, Google and Microsoft) are all platform companies (M. Van Alstyne, 2015).

yielding "a virtuous circle of developers and users that fuel **exponential growth**" (VisionMobile, 2013)

Commercially successful platforms are both **open** and **closed**. They have open API's that make it easy to contribute solutions (complements) on top of the platform and they are closed around core business of their owner in a way that allow the platform owner to capture value (VisionMobile, 2013). For example, Apple is open in regards to App developers but closed towards distribution and the devices it sells (VisionMobile, 2013).

From a traditional **industry selection and positioning view**, the external forces in Porters 5 forcers are considered as having negative effects, so competitive advantage is about raising barriers against competitors and reducing the bargaining power of customers and suppliers etc. Correspondingly, in the traditional strategic **resource view** completive advantage is derived from controlling tangible/intangible company assets used by production that are at best rare, valuable and inimitable (G. Johnson et al., 2011).

Platforms change the rules of business strategy. Instead of considering customers and suppliers in terms of their (negative) bargaining power, they are seen in terms of the **value they add** to the platform. In a similar change, many vendors can be seen as potential providers of complementary value instead of a threat. The strategy is now about **facilitating** external interactions, **orchestrating** external resources and providing **governance** that encourages **external innovation** yet does not threaten the platform owner. Platform companies do not have to own rare or inimitable resources – they can have 3rd parties bring them in. Completive advantage is now derived from a (hard to copy) **interorganizational network** of customers and complementors that form an ecosystem together with the platform owner (M. W. Van Alstyne et al., 2016; M. Van Alstyne, 2015; Stürmer, 2009)

Platform companies must allow and promote 3rd party contributions. Rather than trying to solve and capitalize from all needs, they should focus on a few valuable applications (or plug-ins) and let 3rd parties monetise the **long tail** of potential additions to the platform. By actively leaving room for 3rd parties[41] the platform companies can lower costs, spread business risks among 3rd parties and incentivise an increasing amount of innovation. (M. Van Alstyne, 2015)

Launching a new platform suffers from a difficult **chicken and egg** problem, because users require other users for the platform to have value. Typical **multi-sided platforms** with both buyers and sellers won't get sellers before there is a **critical mass** of buyers, but buyers won't show up before there is a critical mass of sellers (Gawer, A., Cusumano et al., 2012; N. L. Johnson, 2014; Parker & Alstyne, 2014). Here the literature suggests **subsidising** one type of user to get other users, **seeding** the platform with initial complementary value (developed internally or sponsored) for the platform to be useful, **piggybacking** on users of

[41] MySpace serves as an interesting warning of the consequences of failing to invite 3rd parties. In 2008, MySpace was the leading social media hub but in three years it lost most of its community to competing platforms. As put by former MySpace founder Chris DeWolfe: "We tried to create every feature in the world and said, 'O.K., we can do it, why should we let a third party do it?' … We should have picked five to ten key features that we totally focused on and let other people innovate on everything else." (Gillette, 2011)

supplementary networks and/or initially **restricting** the platform to a **micro-market** before branching out (Parker & Alstyne, 2014)

9.1.2. Platforms and open source

I view platforms and vendor sponsored open source projects as similar in multiple ways. They both leverage **external resources** that the vendor does not own, they are both difficult to launch due to the need for **external participation** and they both share the same **socio-technical architecture** of participation discussed in chapter 6.3 and 7.2.

Specifically, the architecture of participation is about supporting external development communities for both open source and for platforms. Hence, in terms of their architecture, digital platforms are a perfect fit for open source projects.

Similar to the chicken and egg launch problem in platforms, open source has something called the **mobilization problem,** a term introduced by Michael Weiss after the "penguin problem" from economics. The mobilization problem is about enticement of developers, since developers won't join before there is a community of other developers. He explains, "Hungry penguins are gathered on a floe of ice. However, none of them wants to dive first for fear of being eaten by a predator. No penguin moves until every penguin moves" (Weiss, 2009, pp. A5–3). The mobilization problem is, however, more simple than the chicken and egg problem because the mobilization problem is **single sided** (developers also assume the role of users) and because providing value attracts users. Hence, Weiss suggests that providing a critical mass of **initial value** can solve the mobilization problem. In other words, by providing the **plausible promise** discussed in chapter 7.1 (Weiss, 2009, pp. A5–3).

From a competitive business standpoint, I find that there are two major forces in play for open source platforms: Network effect (demand side economics of scale) and commoditization of software.

Network effects suggest that the bigger platform wins, which tends to favour **first movers**. Because of demand side economics of scale, competition between similar platforms tends to end with **winner-takes-all** scenarios with few exceptions (Parker & Alstyne, 2014)

Commoditization of software by open source suggests that open source products tend to displace **proprietary products** when having similar capabilities (features and quality): "In product category after product category we are starting to see open source displace commercial products (Hammond, 2014a).

In the absence of strong commercial platforms already having gained solid foothold, new platform entrants that are open source benefit from having the advantage of both forces. Subsequent commercial challengers will find it extremely difficult to compete against both forces. As summed up by Mike Olson, co-founder of Cloudera and with a background from multiple open source companies, "No dominant platform-level software infrastructure has emerged in the last ten years in closed-source, proprietary form… You can no longer win with a closed-source platform, and you can't build a successful stand-alone company purely on open source." (Olson, 2013)

10. A holistic view on contribution decisions by the community

In the literature it is common to assume a direct link between motivation and the act of making contributions, but as discussed in chapters 6.6, 7.3 and 8.7 there are many other factors in play that encourage or constrain contributions. **External factors** that the project initiator has significant influence on, unlike potential contributors. For contributors the factors are considered external since they have little or no influence on them (Krogh et al., 2012).

From the previous discussions on attractiveness, a conclusion is that a motivated developer is not enough for the act of making a decision to contribute to a particular project. A project must also be marketed and the project must be generally attractive to contribute to - more so than any competing projects. In these discussions I have argued that project attractiveness is influenced by external factors.

Scholars like Georg Von Krogh et al. describe the sparse academic work in the area of external factors as "recent and difficult to categorise" (Krogh et al., 2012). Nevertheless, I will propose a way to categorise and model various findings on such factors. As a basis I use chapters 6, 7 and 8 (and in particular sections 6.6, 7.3 and 8.7), where I have drawn on a large amount of literature that mentions the subject in passing (in a way that I have interpreted and classified as being related to attractiveness).

The first categorisation I suggest is to divide the external factors into **genuine attractors** (e.g. capabilities) that make a project attractive and factors that acts as **barriers** (or dis-attractors) that make a project difficult to contribute to or decidedly unattractive. For example, running software that is uniquely useful can be categorized as an attractor, while missing source code is best categorized as a barrier. Using negation, some types of factors can be modelled either as an attractor or a barrier. In these cases, I classify factors only as attractors if they are of significance when comparing with alternative open source projects and thus likely to make a real difference for contributors. In the above example of "missing source code" as barrier, it makes no sense to use "having source code" as an attractor instead, as real open source projects do have source code published. Having source code is expected. It is not something that a contributor looks at as a distinguishing capability. However, he/she does look at code as a barrier if the source is missing.

Building on the classification of attractors versus barriers, I suggest that a contributor's decision to contribute (or not) depends on individual motivation, the attractiveness and barriers of contributing to the project and finally any alternatives as illustrated in the force field in Figure 14.

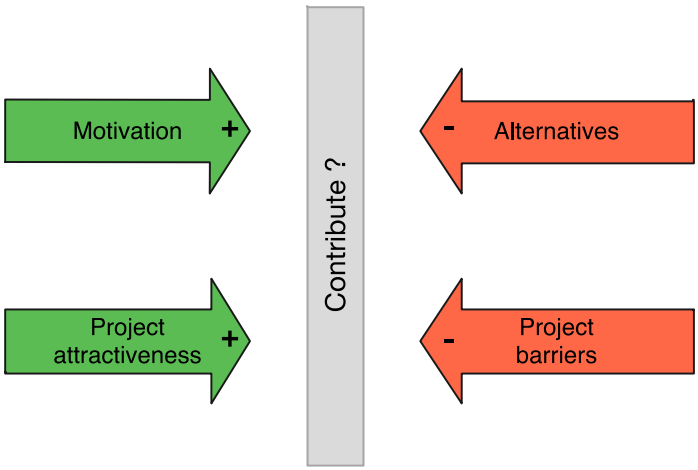


Figure 14 Decision making by a potential contributor

Other categorisations that I suggest are beneficial for looking at external factors are:

- What kind of people/firms that are primarily affected: Users, individual developers or corporate developers.
- The area or perspective the factors most relate to as seen from the vendor: Business, Community and Technical.

Using the above categorisations, I have created a **holistic model** containing a condensed set of all the external factors identified by this thesis that affect project attractiveness. An illustration of my model is shown in Figure 15. A project that is a good fit for open source has most of the attractors and few of the barriers mentioned in the model.

A few factors are difficult to categorize and are thus located at boundaries in the model. In particular, any single choice of programming language & platform can be considered attractive or a barrier depending on the unique viewpoint of each individual open source developer. The only way for this choice to be a near-universal attractor is to support multiple languages and platforms (which is usually difficult and extra demanding in terms of time and resources).

Note, that it is not implied that a project should realistically have all specified attractors and no barriers. There can be valid reasons for some barriers to exist and for the lack of specific attractors (e.g. strategic or economic). However, too few attractors or too many barriers are likely to make the project unsuccessful in attracting a community.

Although representing just a theory at this stage, I believe the model is useful as a summary of what a wide range of literature (plus a bit of common sense of my own) says about what makes an open source project attractive for potential contributors (and what does not). The model is also useful as a kind of checklist when initiating a new project. As such, I believe the model constitute a new addition to the open source literature that is useful in both its current state and as a basis for further study.

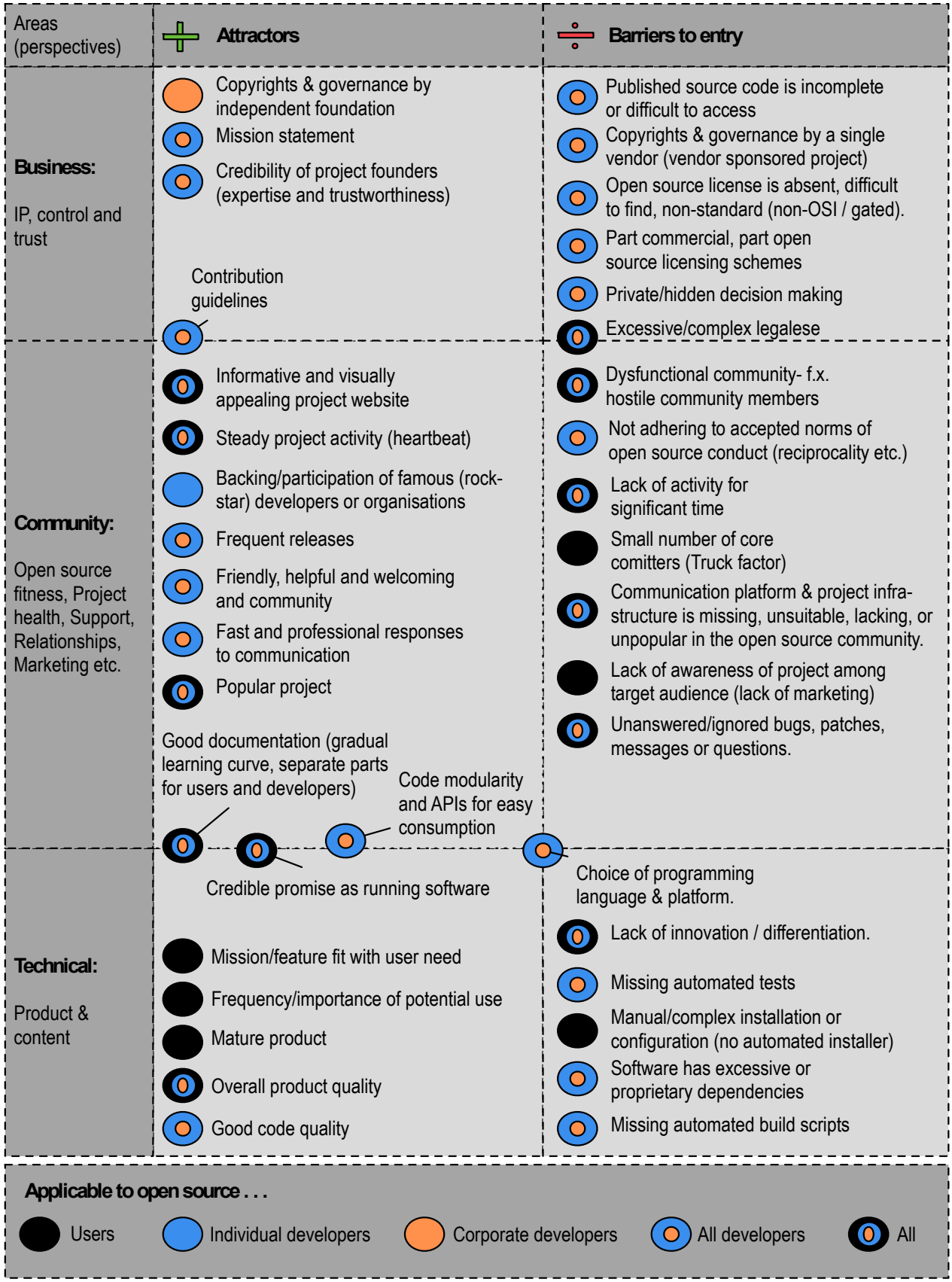


Figure 15 Model of external influencers of attractiveness for contributors

11. Revisiting the digital platform case

This chapter is confidential and has been removed from the public version. Contact the author for access to the content (may require signing a NDA).

12. Conclusion

The objective of this thesis is to answer the following question:

- **What are the general concerns for a firm when initiating an open source digital platform project?**

A firm should be aware that actions and decisions at the **initiation stage** define the project, influence how likely the project is to be **successful** in terms of **attracting a community** and lay the foundation for the ability of the business to generate **economic rents** in the future.

My answer to the research question is presented in Figure 16 in form of a **holistic view** of the general **concerns**. The illustration should be read bottom up.

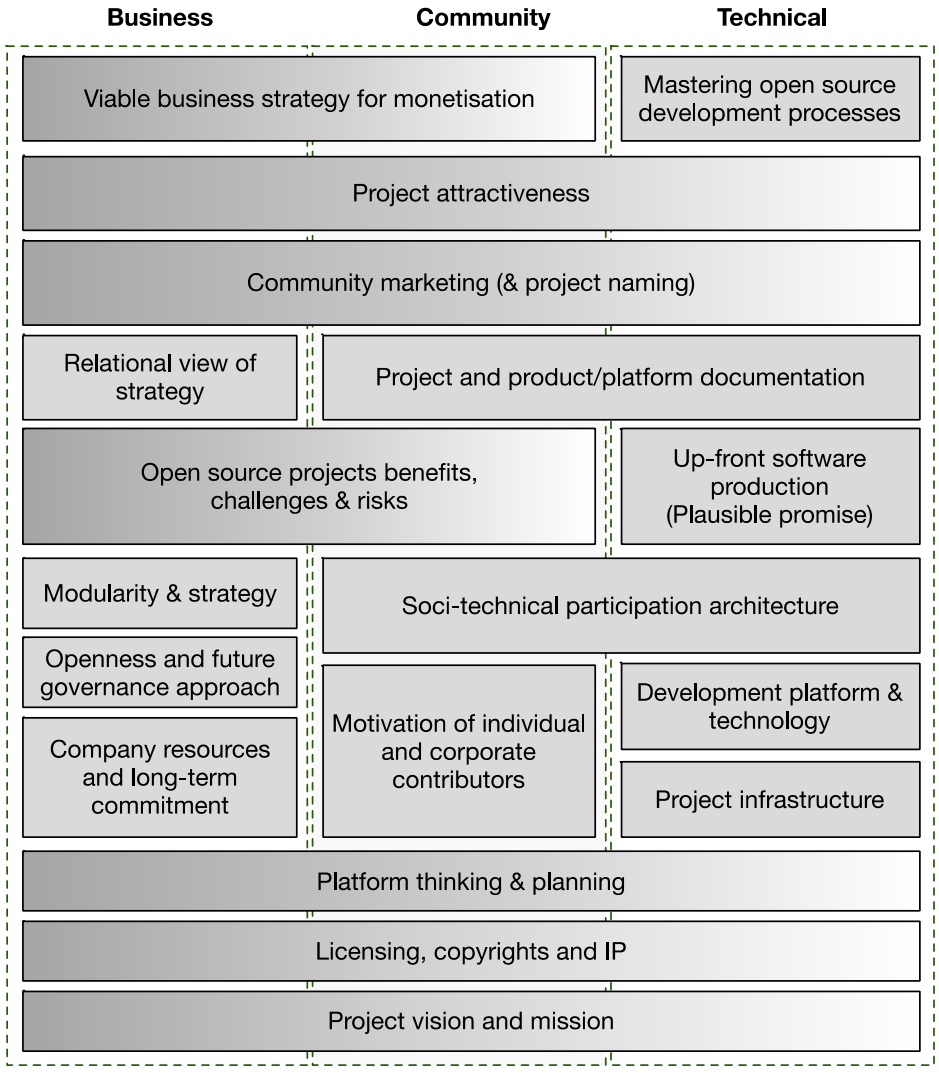


Figure 16 Vendor concerns related to initiating an open source digital platform

As illustrated in Figure 16, a firm should start with formulating a public product **vision** and a project **mission** for both internal use and as a **rallying point** by the community as discussed

in 7.3. Both should be broad enough to attract contributors and narrow enough to avoid attracting the wrong contributors.

The firm should then consider if **copyrights** should be **owned** or **distributed** and which licences to use according to community **participation needs**, business **strategy** and **competitive** considerations as discussed in chapter 8.3 and 8.12 etc. Generally for **vendor-sponsored projects**, **copyright** (and hence re-licensing rights) of key software components should belong to the firm (see chapter 8.5). In any case, firms should be aware that copyrights and licensing are expensive or outright impossible to change after initiation (see chapter 8.3).

At the project initiation stage of a platform the firm should consider the eventual business needs in regard to the future **ecosystem** of the platform. What should be **open** sourced, what should be **closed** (in order to secure revenue streams) and strategies to attract a **critical mass** of **users** to the platform (see chapter 9). All of which can influence the project in regard to priorities, features etc. As discussed, digital platforms are a perfect fit for open source, as the same **socio-technical** design that makes open source work is ideally suited for open platforms as well (see 9.1.2). I also see open source as a way to **subsidise** some platforms during launch as discussed in chapters 9 and 11.

From a business perspective, a primary concern is if the firm has the **collective will**, the **resources** and long-term **commitment** for a serious open source project. As discussed in chapter 8.10, going open source is a **significant endeavour**. In the short-term it will not save anything or produce benefits compared to developing a product with internal resources. When resources and commitment are secured, the firm should then consider how the **software modularity** affects their strategy (see chapter 8.5.1) and how the choice of a level of **openness** (and a future method of **governance**) can support both the business interests of the firm and the need of the community (see chapter 8.5, 8.12 and Appendix F).

A firm must understand **motivation of potential contributors** in order to have a chance to make the project attractive for the community and later to successfully lead a project. As discussed in chapters 6.4, effective motivation of individual developers (of all kinds) is about **inner** and **internalised motivation** that respects developers' personal need for **autonomy**, **competence** and **relatedness** to other people. Unlike a commercial setting that allows managers to exercise control through **carrot and stick** approaches to motivation, such (highly ineffective and short-term) approaches are not possible in open source projects[42]. Individual open source developers share most of the same **intrinsic** and **extrinsic motivations** with developers in general, but the **distinguishing motivations** that are unique to individual open source developers are **ideology**, **having fun**, **giving something back** to the community and **own-use** opportunities (see chapters 6.4 and 6.5).

A firm must set up suitable **project infrastructure** such as a code repository, bug tracking, forums etc. Also, before any software can be written, **technology choices** must be made such as a development platform. Importantly, the choice of infrastructure, as well as all technology

[42] The **self-determined methods** of motivating independent open source developers can be viewed as an ideal that managers in traditional commercial settings should strive for, when optimizing long-term motivation of their employed developers in order get the resulting benefits of high performance, quality and engagement.

choices, must first and foremost suit the needs and expectations of the external community – not just the needs of the vendor (see chapter 7.3)

Software architecture in an open source project is not just about technicalities. A firm should design the **software architecture** of the solution not only from a technical view of engineering but also in regard to the needs of **community participation**, business needs and in regard to what should be opened/closed in the platform as discussed in chapters 6.3, 7.2, 8.5 etc. In particular, **plug-in architecture** is suitable for wide participation (and for support of platform ecosystems).

A firm should generally be aware of special benefits, drawbacks and challenges of open innovation in form of an open source project as described in chapters 8.9, 8.10 and 8.12 etc. A notable technical challenge is for internal developers unused to open source to master the **open source development processes** and **open source governance** approaches (in particular **transparency**). This is as big a change for proprietary developers as going from waterfall to agile development.

During or before project initiation, the firm generally needs to develop a preliminary version of the intended product that provides enough value to attract users and to persuade contributors to invest their time by providing a **plausible promise** that the project will attain its goals (see chapter 7.1). Depending on how overall attractive the project is, this pre-community software may need to have more or less strength in features and quality. A simple **MVP** may not be enough to convince developers (see chapter 7.1).

A firm should carefully **document** all aspects of the project such as community governance (for example contribution guidelines), the product (platform), source code organisation etc. Documentation must support different **audiences** such as users of the product, contributors and project site visitors. The documentation must be **informative**, **indexed** (**searchable**), presented in a **visually appealing** way and written so that it provides a **gradual learning curve**. Good documentation is a vital component of marketing (see chapter 8.8) and an important factor in project attractiveness (see chapter 10).

A firm should allocate considerable resources to deal with **marketing** of the project towards users and contributors. **Community marketing** is both similar and different from *normal* marketing of products as mentioned in chapter 8.8. Most importantly, potential contributors are initially seen as **users** with a need. Hence, marketing to contributors is initially about getting **visitors**, attracting users and helping some of these users to become contributors.

A firm should arrange the project in a way that is as **attractive** to a community as possible and limits **barriers** to those that are absolutely necessary for business reasons. The model in Figure 15 (chapter 10) can be used for that purpose. For example, just like a famous lead actor may positively affect ticket sales for a Hollywood movie production, a firm can improve project attractiveness by hiring or partnering with a **famous** and well-respected open source developer or open source organisation.

Finally, a firm should consider **monetisation** and business strategy. Strategy should include a **relational view** of competitiveness and platform thinking. As discussed in chapters 8.2 and 9, traditional **strategic positioning** and internal strategic **resource views** of competitive advantages are not directly applicable in an open source or digital platform context.

12.1. Contributions to existing literature

I believe this thesis makes two significant additions to the existing academic body of knowledge.

12.1.1. New holistic view of open source project initiation

While there are vast amounts of (academic) literature about the social, technical or business side of commercial open source projects, there is precious little literature, which looks at open source **holistically** from multiple perspectives. There is literature that looks at things from a commercial business view and also mentions community aspects (or the opposite), but equal/holistic treatment is rare. Specifically, I am aware of no other existing literature that clearly and broadly look at commercial open source project initiation holistically from all of my chosen 3 perspectives: business, social and technical. In addition, I have found few **cross-references** between business literature, academic literature and the open source community's own writings. In both of these areas, I believe this thesis contributes with new knowledge by combining different views.

The **multi-perspective** approach I have used is important, as decisions made at business, community or technical level impact other areas as illustrated in Figure 17. E.g. business decisions affect the ability for a project to grow a community and software architecture can positively or negatively affect both community size and a firm's competitive position.

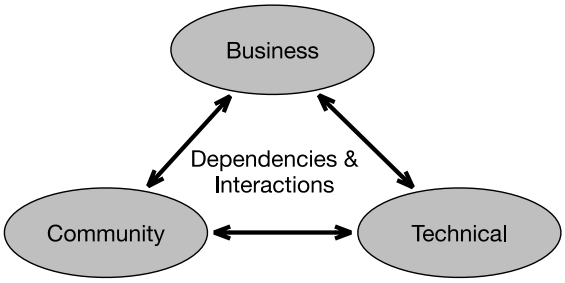


Figure 17 Cross-perspective interactions

12.1.2. New and comprehensive model of open source project attractiveness

While there is an abundance of literature that mentions related subjects such as success factors, patterns and good practices of open source, there are very few sources that deal with the concept of open source **project attractiveness** and recognize it as such. Apart from a single (narrow-scoped) study expressly about project attractiveness the only other sources I could find was nuggets of information scattered between many references. I did find literature about **barriers** (unfortunately too late for inclusion) but I found no literature surveys and no other models about project attractiveness apart from the single study mentioned above.

This thesis contributes to academic knowledge by organising the many extracts about open source project attractiveness into a single holistic model in chapter 10. The model is yet untested but I believe it is a useful. As an overview it is unmatched in detail and completeness.

12.2. Limitations and critique of this thesis

This thesis bases its analysis on existing literature with no **empirical** data of its own. As such the strength of its conclusions can be no better then the underlying literature. Apart from the subject of motivation, most of the literature bases its conclusions on small surveys and other **qualitative research**, game theory based on uncertain assumptions, specific project cases, personal experiences and anecdotal evidence. All of which can't be **generalized**.

While I have looked extensively for suitable literature and searched for specific answers, in the end I have for mundane reasons only read and used a fraction of what is available. Hence I may have overlooked important literature.

In addition, I am particularly aware of unprocessed yet relevant literature regarding participation barriers and maturity/health models of open source. Literature that I suspect could be beneficial for a greater understanding (of project attractiveness in particular) but which I could not include due to lack of time and space.

The most researched area in open source literature appears to be motivation of individuals, but I find even that area lacking. As can be seen in Appendix A, individual studies of motivation point to different intrinsic or extrinsic factors as being more or less important (no consensus)[43]. For researching individual motivation, my use of output from **meta-studies** means that many nuances disappear and those specific aspects of motivation that apply to my specific context may be missing. On the subject of motivation of firms, the existing literature is quite lacking and consequently so is the associated chapter 6.5 of this thesis.

On the related subject of attractiveness, the model that I have constructed (Figure 15) is currently no more than a postulate. Firstly, most sources of external factors in the literature are based on individual experiences that can't be generalized. Secondly, I have had to reinterpret most sources as the literature rarely classifies something to be about external factors of attractiveness. My reinterpretation may be wrong. Thirdly, my classification of external factors into attractors or barriers is based on my judgement, which may be in error too (as well as subject to my personal bias).

12.3. Opportunities for further research

There are many subjects covered in this thesis that would benefit from additional research. E.g. Developer motivation, project attractiveness and how the community reacts to various monetisation schemes in the specific context of vendor sponsored open source digital platforms. Such subjects could benefit from additional research of further literature, by **qualitative** case studies of projects similar to my case and by introducing **quantitative** research based on empirical data etc.

In particular, a next step for the model of attractiveness from chapter 10, would be to set up empirical studies to see if the various stated attractors and barriers can be supported or not. I suspect that a suitable approach could be a semi-automatic **data/content analysis** of a significant amount of open source projects in open source **project repositories** like Github.

From the derived **statistics** I believe it should be possible to see if the successful and not so successful projects have technical, social and business characteristics that support or reject various aspect of the model. Such a study could also be used to provide some indication of the relative strength of various factors.

[43] I suspect this is due to studies asking questions on what motivates developers without enough context. A very exaggerated analogue is to ask random people on the Internet 'Is it sunny?' and then trying to extract meaning from the many responses without taking both geographical location and time of the observation into account.

13. Literature

Aberdour, M. (2007). Achieving quality in open source software. *IEEE Software*, (September), 58–64. http://doi.org/10.1016/j.ejrad.2010.05.004

Akst, J. (2011). Public Solves Protein Structure | The Scientist Magazine®. Retrieved February 1, 2016, from http://www.the-scientist.com/?articles.view/articleNo/31155/title/Public-Solves-Protein-Structure/

Allen, D. (2015). Everything You Need to Know to Grow Open Source. Retrieved from https://www.youtube.com/watch?v=a_vqg-go8XI

Alstyne, M. Van. (2015). Platform Shift : How New Biz Models Are Changing the Shape of Industry. InfoEcon.

Alstyne, M. W. Van, Parker, G. G., & Choudary, S. P. (2016). Pipelines, Platforms, and the New Rules of Strategy. Retrieved April 22, 2016, from https://hbr.org/2016/04/pipelines-platforms-and-the-new-rules-of-strategy

Andersen-Gott, M., Ghinea, G., & Bygstad, B. (2012). Why do commercial companies contribute to open source software? *International Journal of Information Management*, *32*(2), 106–117. http://doi.org/10.1016/j.ijinfomgt.2011.10.003

Asay, M. (2014). Open-Source Projects Need More Than Good Code—They Need Marketing - ReadWrite. Retrieved January 25, 2016, from http://readwrite.com/2014/10/10/open-source-marketing-apache-storm-nathan-merz

Asghar, I., & Usman, M. (2013). Motivational and de-motivational factors for software engineers: An empirical investigation. *Proceedings - 11th International Conference on Frontiers of Information Technology, FIT 2013*, 66–71. http://doi.org/10.1109/FIT.2013.20

Aslett, M. (2010). From support services to software services The evolution of open source business strategies. the451group.

Aslett, M. (2011). Updated open source business strategy framework. Retrieved April 12, 2016, from https://blogs.the451group.com/opensource/2011/01/06/updated-open-source-business-strategy-framework/

Aslett, M. (2012). On the rise and fall of the GNU GPL. Retrieved April 12, 2016, from https://blogs.the451group.com/opensource/2012/12/19/on-the-rise-and-fall-of-the-gnu-gpl/

Baldwin, C. Y., & Clark, K. B. (2006). The Architecture of Participation: Does Code Architecture Mitigate Free Riding in the Open Source Development Model? *Management Science*, *52*(7), 1116–1127. http://doi.org/10.1287/mnsc.1060.0546

Baldwin, C. Y., & Henkel, J. (2012). The Impact of Modularity on Intellectual Property and Value Appropriation The Impact of Modularity on Intellectual Property and Value Appropriation.

Berkholz, D. (2013). The size of open-source communities and its impact upon activity, licensing, and hosting. Retrieved March 5, 2016, from http://redmonk.com/dberkholz/2013/04/22/the-size-of-open-source-communities-and-its-impact-upon-activity-licensing-and-hosting/

Berkholz, D. (2014). GitHub language trends and the fragmenting landscape – Donnie Berkholz's Story of Data. Retrieved February 27, 2016, from http://redmonk.com/dberkholz/2014/05/02/github-language-trends-and-the-fragmenting-landscape/#ixzz30wEgsUif

Boel, S. K., & Cecez-Kecmanovic, D. (2010). Literature reviews and the hermeneutic circle.

Auatralian Academic & Research Libraries, *41*(2), 129–144. http://doi.org/10.1080/00048623.2010.10721450

Bonaccorsi, A., & Rossi, C. (2006). Comparing motivations of individual programmers and firms to take part in the open source movement: From community to business. *Knowledge, Technology & Policy*, *6*(4), 1–6. http://doi.org/10.1007/s12130-006-1003-9

Brodkin, J. (2010). Microsoft: We love open source. Retrieved April 11, 2016, from http://www.networkworld.com/article/2216878/windows/microsoft---we-love-open-source-.html

Budd, A., & Traynor, D. (2013). An Interview with Andy Budd - Inside Intercom. Retrieved April 4, 2016, from https://blog.intercom.io/an-interview-with-andy-budd/

Buyukkayhan, A. S., Onarlioglu, K., Robertson, W., & Kirda, E. (2016). CrossFire: An Analysis of Firefox Extension-Reuse Vulnerabilities. *Ndss*, (February), 21–24.

Cavalier, F. J. (1998). Some Implications of Bazaar Size. Retrieved December 28, 2015, from http://www.mibsoftware.com/bazdev/0003.htm

Cerasoli, C. P., Nicklin, J. M., & Ford, M. T. (2014). Intrinsic Motivation and Extrinsic Incentives Jointly Predict Performance: A 40-Year Meta-Analysis. *Psychological Bulletin*, *140*(4), 980–1008. http://doi.org/10.1037/a0035661

César, A. (2014). *A Theory of Motivation and Satisfaction of Software Engineers*. Universidade Federal de Pernambuco.

Chaffey, D. (2016). Marketing models that have stood the test of time. Retrieved May 5, 2016, from http://www.smartinsights.com/digital-marketing-strategy/online-business-revenue-models/marketing-models/

Chalef, D., & Mickos, M. (2008). Open-source software as guerrilla marketing strategy, 14–16.

Christensen, M. M., & Jensen, J. E. (2015). Strategiudvikling med open source.

Christina M. Stello. (2014). An Integrative Literature Review. *Herzberg's Two-Factor Theory of Job Satisfaction*, 32.

Collins-Sussman, B., & Fitzpatrick, B. (2007). What's In It for Me? Benefits from Open Sourcing Code. GoogleTechTalks. Retrieved from https://www.youtube.com/watch?v=ZtYJoatnHb8

Columbus, L. (2015). Roundup Of Cloud Computing Forecasts And Market Estimates, 2015 - Forbes. Retrieved April 17, 2016, from http://www.forbes.com/sites/louiscolumbus/2015/01/24/roundup-of-cloud-computing-forecasts-and-market-estimates-2015/#4e7a5473740c

Contributor-covenant.org. (2014). Contributor Covenant: A Code of Conduct for Open Source Projects. Retrieved May 5, 2016, from http://contributor-covenant.org/

Corbet, J. (2010). LCA: How to destroy your community. Retrieved May 2, 2016, from http://lwn.net/Articles/370157/

Deci, E. L. (2012). Promoting Motivation, Health, and Excellence. Retrieved from https://www.youtube.com/watch?v=VGrcets0E6I

Eaves, D. (2011). Open Source Community Building. Djangocon. Retrieved from https://www.youtube.com/watch?v=SzGi1DfbZMI

Economist, T. (2014). Platforms - Something to stand on. Retrieved April 22, 2016, from http://www.economist.com/news/special-report/21593583-proliferating-digital-platforms-will-be-heart-tomorrows-economy-and-even

Ehls, D. (2013). Open Source Participation Behavior-A Review and Introduction of a Participation Lifecycle Model. *35th DRUID Celebration Conference*. Retrieved from http://druid8.sit.aau.dk/acc_papers/8tfya9e35eitx6godspbo767ars3.pdf

Erway, T., & Ruff, N. (2013). Can You Market an Open Source Project ? Embedded Linux Conference.

Finette, P. (2012). Changing the world with Open Source - Pascal Finette. Retrieved May 6, 2016, from https://www.youtube.com/watch?v=n6e5S80PRBQ

Fingas, R. (2016). App Store reached estimated $6.4 billion of Apple's revenue during 2015. Retrieved February 1, 2016, from http://appleinsider.com/articles/16/01/07/app-store-reached-estimated-64-billion-of-apple-revenue-during-2015

Fogel, K. (2015). *Producing Open Source Software - How To Run A Successful Free Software Project*. Self published. Retrieved from http://producingoss.com/

Gawer, A., Cusumano, M. A., Gawer, A., & Cusumano, M. A. (2012). Industry Platforms and Ecosystem Innovation. In *Druid* (Vol. 31, pp. 417–433.). http://doi.org/10.1111/jpim.12105

Geer, D. (2005). Java IDE. *Computer*, *38*(7), 16–18. http://doi.org/10.1109/MC.2005.228

Germain, J. M. (2015). HP's Marten Mickos: Open Source Is Not a Business Model. Retrieved April 12, 2016, from http://www.linuxinsider.com/story/81732.html

Ghazawne, A. (2015). Digital Platforms & Ecosystems - ITU Presentation.

Gillette, F. (2011). The Rise and Inglorious Fall of Myspace. Retrieved May 7, 2016, from http://www.bloomberg.com/news/articles/2011-06-22/the-rise-and-inglorious-fall-of-myspace

Glen, P. (2003). Leading Technical People, 19–24.

Goodin, D. (2016). NoScript and other popular Firefox add-ons open millions to new attack. Retrieved April 6, 2016, from http://arstechnica.com/security/2016/04/noscript-and-other-popular-firefox-add-ons-open-millions-to-new-attack/

Hall, T. (2008). Motivating Software Developers. Retrieved from http://www.uio.no/studier/emner/matnat/ifi/INF5700/h08/undervisningsmateriale/Motivating software developers.ppt

Hall, T., Sharp, H., Beecham, S., Baddoo, N., & Robinson, H. (2008). What do we know about developer motivation? *IEEE Software*, *25*(4), 92–94. http://doi.org/10.1109/MS.2008.105

Hammond, J. (2009). Open Source and its role in a new IT ecosystem. Forrester. Retrieved from http://www.slideshare.net/Brunovonrotz/open-source-and-its-role-in-a-new-it-ecosystem

Hammond, J. (2010). OSS Adoption Patterns In Enterprise IT. Forrester.

Hammond, J. (2014a). Open Source By The Numbers. Forrester. Retrieved from https://www.youtube.com/watch?v=GTFM39h5m5g

Hammond, J. (2014b). Open Source: A Key Component of Modern Applications (keynote). Forrester.

Hammouda, I., Aaltonen, T., & Sirkkala, P. (2008). Exploiting social software to build open source communities. *Automated Software Engineering - Workshops, 2008. ASE Workshops 2008. 23rd IEEE/ACM International Conference on DOI - 10.1109/ASEW.2008.4686309*, 42–45. http://doi.org/10.1109/ASEW.2008.4686309

Harding, R. (2014). Growing an Open Source Project.

Harris, T. (2015). *Software licences*.

Heitmann, J. (2014). The Lean Startup - A pragmatic view on its Flaws and Pitfalls, (November).

Helen Walton. (2015). Lean Start-Up, and How It Almost Killed Our Company. Retrieved January 14, 2016, from http://www.infoq.com/articles/lean-startup-killed

Hertel, G., Niedner, S., & Herrmann, S. (2003). Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research Policy*, *32*(7), 1159–1177. http://doi.org/10.1016/S0048-7333(03)00047-7

Hippel, E. von, & Krogh, G. von. (2002). Open Source Software and the "Private-Collective"

Innovation Model: Issues for Organization Science, *14*(2003), 208–223.
Hoerr, A. (2010). *Drive - The Surprising Truth About What Motivates Us*.
Holman, Z. (2011). Open Source Doesn't Just Market Itself. Retrieved January 25, 2016, from http://zachholman.com/posts/open-source-marketing/
Ingo, H. (2011). How to grow your open source project 10x and revenues 5x. *OSCON*. Retrieved from http://openlife.cc/blogs/2010/november/how-grow-your-open-source-project-10x-and-revenues-5x
Johnson, G., Whittington, R., & Scholes, K. (2011). *Exploring Strategy, 9th ed*.
Johnson, N. L. (2014). Platform or Perish: An Introduction to Platform Economics. Retrieved April 22, 2016, from http://www.applicoinc.com/blog/platform-or-perish-an-introduction-to-platform-economics/
Karen McCally. (2010). Rochester Review :: University of Rochester. Retrieved March 15, 2016, from http://www.rochester.edu/pr/Review/V72N6/0401_feature1.html
Kawaguchi, K. (2012). Creating a Developer Community. Monki Gras. Retrieved from https://www.youtube.com/watch?v=zfMdaHS3rYs
Ke, W. (2010). The Effects of Extrinsic Motivations and Satisfaction in Open Source Software Development. *Journal of the Association for Information Systems*, *11*(12), 784–808.
Kniberg, H. (2016). Making sense of MVP (Minimum Viable Product) – and why I prefer Earliest Testable/Usable/Lovable. Retrieved April 4, 2016, from http://blog.crisp.se/2016/01/25/henrikkniberg/making-sense-of-mvp
Krishnamurthy, S. (2002). Cave or Community, *7*(6), 1–10.
Kroah-Hartman, G., Corbet, J., & McPherson, A. (2009). *Linux Kernel Development - How Fast it is Going, Who is Doing It, What They are Doing, and Who is Sponsoring It*. Retrieved from http://www.linuxfoundation.org/sites/main/files/publications/whowriteslinux.pdf
Krogh, G. Von, Haefliger, S., Spaeth, S., & Wallin, M. W. (2012). CARROTS AND RAINBOWS: MOTIVATION AND SOCIAL PRACTICE IN OPEN SOURCE SOFTWARE DEVELOPMENT, *36*(2), 649–676.
Lars Kurth. (2010). Viewing Communities as Funnels | Tales From The Community on WordPress.com. Retrieved January 17, 2016, from https://talesfromthecommunity.wordpress.com/2012/06/16/viewing-communities-as-funnels/#more-361
Lerner, J., & Schankerman, M. (2010). *The Comingled Code: Open Source and Economic Development*. *MIT Press Books* (Vol. 1). The MIT Press. Retrieved from http://ideas.repec.org/b/mtp/titles/0262014632.html
Levine, P. (2014). Why There Will Never Be Another RedHat - The Economics Of Open Source, 1–31. Retrieved from http://techcrunch.com/2014/02/13/please-dont-tell-me-you-want-to-be-the-next-red-hat/
Lokhman, A., Mikkonen, T., Hammouda, I., Kazman, R., & Chen, H.-M. (2013). A Core-Periphery-Legality Architectural Style for Open Source System Development. *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, 3148–3157. http://doi.org/10.1109/HICSS.2013.34
Lowagie, B. (2014). Heartbleed, an ASL business model failure? *Online Gazette*. Retrieved from http://lowagie.com/heartbleed_asl_bumodel
Mair, P., Hofmann, E., Gruber, K., Hatzinger, R., Zeileis, A., & Hornik, K. (2015). What Drives Package Authors to Participate in the R Project for Statistical Computing? Exploring Motivation, Values, andWork Design Patrick. *Proceedings of the National Academy of Sciences of the United States of America*.
Marz, N. (2014). History of Apache Storm and lessons learned - thoughts from the red planet - thoughts from the red planet. Retrieved January 25, 2016, from

http://nathanmarz.com/blog/history-of-apache-storm-and-lessons-learned.html
McAllister, N. (2013). Most projects on GitHub not open source licensed. Retrieved April 26, 2016, from http://www.theregister.co.uk/2013/04/18/github_licensing_study/
Meeker, H. (2015). *Open Source for Business - A practical Guide to Open Source Software Licensing*. Flemming Editorial.
Merrill, S. (2011). Open Source is an Ecosystem, not a Zero Sum Game. Retrieved January 17, 2016, from http://techcrunch.com/2011/08/21/linuxcon-open-source-is-an-ecosystem-not-a-zero-sum-game/
Mickos, M. G. (2011). Commercialization of Open Source. Retrieved from http://ecorner.stanford.edu/videos/2832/Commercialization-of-Open-Source
Monty Widenius, M., & Nyman, L. (2014). The Business of Open Source Software: A Primer. *Technology Innovation Management Review*, (January), 4–11. Retrieved from http://search.ebscohost.com/login.aspx?direct=true&db=ent&AN=94361297&site=ehost-live
Mougayar, W. (2013). Startup Management » Don't Let Lean Startup Become a Crutch. Retrieved April 5, 2016, from http://startupmanagement.org/2013/07/25/dont-let-lean-startup-become-a-crutch/
Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., & Ye, Y. (2002). Evolution patterns of open-source software systems and communities. *Proceedings of the International Workshop on Principles of Software Evolution - IWPSE '02*, (January), 76 – 85. http://doi.org/10.1145/512035.512055
North bridge and Black Duck software. (2015). *2015 - the future of OPEN source*.
Olson, M. (2013). The Cloudera Model. Retrieved January 1, 2016, from https://www.linkedin.com/pulse/20131003190011-29380071-the-cloudera-model
Open Source Initiative. (2007). The Open Source Definition. Retrieved April 12, 2016, from https://opensource.org/osd
OSSS.io. (2014a). Building an Open Source Community. Retrieved from https://www.youtube.com/watch?v=nxiiNRNTscU
OSSS.io. (2014b). Founding Open Source Companies. Retrieved from https://www.youtube.com/watch?v=r2obF8MQDvA
OSSS.io. (2014c). Growing Open Source Companies. Retrieved from https://www.youtube.com/watch?v=9qPnqkuxLGw
Osterloh, M., & Rota, S. (2004). Trust and Community in Open Source Software Production. *Analyse & Kritik*, *26*(1), 279–301.
Osterwalder, A., & Pigneur, Y. (2010). *Business Model Generation*. *Journal of Chemical Information and Modeling* (Vol. 53). Wiley.
Parker, G., & Alstyne, M. W. Van. (2014). Platform Strategy Survey, (1967), 14. http://doi.org/10.2139/ssrn.2439323
Pink, D. H. (2010). RSA Animate - Drive The surprising truth about what motivates us. Retrieved from http://www.danpink.com/video/
Posted, J. S. (2014). Treat Open Source Like a Startup ★ Mozilla Hacks – the Web developer blog. Retrieved January 25, 2016, from https://hacks.mozilla.org/2014/05/open-source-marketing-with-velocityjs/
Raymond, E. (2002). The cathedral and the bazaar, (version 3).
Rich Sands. (2012). By the Numbers. *Black Duck Software*. Retrieved from http://www.slideshare.net/blackducksoftware/open-source-by-the-numbers
Riehle, D. (2009). The Business Model of Commercial Open Source Software. *Business*.
Riehle, D. (2012). The single-vendor commercial open course business model. *Information Systems and E-Business Management*, *10*(1), 5–17. http://doi.org/10.1007/s10257-010-

0149-x
Riehle, D. (2014). The Five Stages of Open Source Volunteering, (March).
Ries, E. (2011). *The Lean Startup*. Penguin.
Ries, E., & Hartman, K. (2011). Lean Startup Book Summary. http://doi.org/23
Roberts, J. a., Hann, I.-H., & Slaughter, S. a. (2006). Understanding the Motivations, Participation, and Performance of Open Source Software Developers: A Longitudinal Study of the Apache Projects. *Management Science*, *52*(7), 984–999. http://doi.org/10.1287/mnsc.1060.0554
Ruston, J. (2007). How to start an open source project. Retrieved from https://vimeo.com/856110
Ryan, R., & Deci, E. (2000). Intrinsic and Extrinsic Motivations: Classic Definitions and New Directions. *Contemporary Educational Psychology*, *25*(1), 54–67. http://doi.org/10.1006/ceps.1999.1020
Ryan, R. M., & Deci, E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *The American Psychologist*, *55*(1), 68–78. http://doi.org/10.1037/0003-066X.55.1.68
Santos, C., Kuk, G., Kon, F., & Pearson, J. (2012). The attraction of contributors in free and open source software projects. *The Journal of Strategic Information Systems*, *22*(1), 26–45. http://doi.org/10.1016/j.jsis.2012.07.004
Sen, R., Subramaniam, C., & Nelson, M. L. (2009). Determinants of the Choice of Open Source Software License. *Journal of Management Information Systems*, *25*(3), 207–240. http://doi.org/10.2753/MIS0742-1222250306
Shah, S. K. (2006). Motivation, Governance, and the Viability of Hybrid Forms in Open Source Software Development. *Management Science*, *52*(7), 1000–1014. http://doi.org/10.1287/mnsc.1060.0553
Soós, J. K., Takács, I., Krasz, K. G., & Villám, O. (2013). Psychology - Motivation theories. Retrieved from http://www.tankonyvtar.hu/hu/tartalom/tamop412A/2011-0023_Psychology/030300.scorml
Stackoverflow Survey. (2016). Developer Survey Results. Retrieved from http://stackoverflow.com/research/developer-survey-2016
Stam, W., & Joode, R. van W. de. (2007). Analyzing Firm Participation in Open Source Communities.
Stewart, K. J., Ammeter, A. P., & Maruping, L. M. (2006). Impacts of license choice and organizational sponsorship on user interest and development activity in open source software projects. *Information Systems Research*, *17*(2), 126–144. http://doi.org/10.1287/isre.1060.0082
Stürmer, M. (2005). *Open source community building*. *Licentiate, University of Bern*. Bern. Retrieved from http://www.opensource.ch/fileadmin/user_upload/opensource.ch/knowhow/2005_OpenSourceCommunityBuilding.pdf
Stürmer, M. (2009). *How firms make friends: Communities in private-collective innovation*. *ETH Zürich, Doctoral Dissertation No. 18630*.
Stürmer, M. (2015). Digital sustainability of open source communities Open Source Software : Community, (February).
Stürmer, M., & Myrach, D. T. (2015). Research on open source software , management and communities : Introduction to research on open source software , selection of papers by students. Retrieved from http://www.slideshare.net/nice/introduction-to-research-on-open-source-software
The451group. (2008). *Open Source – Is not a business model*.

Todorović, A. (2015). Open source licensing at Github. Retrieved April 12, 2016, from https://opensource.com/life/15/7/interview-ben-balter-github

Tsay, J. (2015). *Thesis Proposal Software Developers Using Signals in Transparent Environments*. Carnegie Mellon University.

Turk, M. J. (2013). How to Scale a Code in the Human Dimension, 1–9. http://doi.org/10.1145/2484762.2484782

Vilen, P. (2013). *Publishers, turnkeys, clubs and boutiques. A business model taxonomy in the context of free open source software extensions. Case WordPress.*

VisionMobile. (2013). *The M2M Ecosystem Recipe - How Telcos can win the M2M game by playing by ecosystem rules*.

Volpi, M. (2014). A "Perfect Storm" Moment for Multibillion-Dollar Open Source Companies. Retrieved from http://recode.net/2014/03/25/a-perfect-storm-moment-for-multibillion-dollar-open-source-companies/

von Krogh, G., & von Hippel, E. (2006). The promise of research on open source software. *Management Science*, *52*(7), 975–983. http://doi.org/10.1287/mnsc.1060.0560

Walli, S. (2007). Free and Open Source Software Developers Working for Free (Economics 101). Retrieved April 29, 2016, from http://stephesblog.blogs.com/my_weblog/2007/09/free-and-open-1.html

Walli, S. R. (2013a). Once More unto the Breach: Patterns and Practices for Open Source Software Success. Retrieved April 30, 2016, from http://stephesblog.blogs.com/my_weblog/2013/07/patterns-and-practices-for-open-source-software-success.html

Walli, S. R. (2013b). Patterns for Open Source Succes Talk. OuterConf. Retrieved from https://www.youtube.com/watch?v=iPjvLnJSn7U

Walli, S. R. (2016). Patterns and Practices for Open Source Project Success.

Waltl, J., Henkel, J., & Baldwin, C. Y. (2012). IP modularity in software ecosystems: How SugarCRM's IP and business model shape its product architecture. *Lecture Notes in Business Information Processing*, *114 LNBIP*, 94–106. http://doi.org/10.1007/978-3-642-30746-1_8

Watson, B. R. T., & Boudreau, M. (2015). The Business of OPEN SOURCE - Missing Patterns, *51*(4), 41–46.

Weiss, M. (2009). Performance of Open Source Projects. *EuroPLoP*, (11), 1–15. Retrieved from http://ceur-ws.org/Vol-566/A5_PerfOpenSource.pdf?origin=publication_detail\nhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Performance+of+Open+Source+Projects.#1

West, J. (2003). How open is open enough? *Research Policy*, *32*(7), 1259–1285. http://doi.org/10.1016/S0048-7333(03)00052-0

West, J., & O'mahony, S. (2008). The Role of Participation Architecture in Growing Sponsored Open Source Communities. *Industry & Innovation*, *15*(2), 145–168. http://doi.org/10.1080/13662710801970142

Widenius, M. (2016). How to create a successful (in business and development) open source project. Open Ocean - Open source days.

Wikipedia. (2016a). Bus Factor. Retrieved from https://en.wikipedia.org/wiki/Bus_factor

Wikipedia. (2016b). Plug-in (computing) - Wikipedia, the free encyclopedia. Retrieved April 16, 2016, from https://en.wikipedia.org/wiki/Plug-in_%28computing%29

Wikiquote. (2016). Richard Stallman. Retrieved May 6, 2016, from https://en.wikiquote.org/wiki/Richard_Stallman

Yoffie, D., & Cusumano, M. (2015). Strategy Rules - five timeless lessons from Bill Gates, Andry Grove And Steve Jobs. Retrieved from http://hbswk.hbs.edu/item/the-5-strategy-rules-

of-bill-gates-andy-grove-and-steve-jobs
Young, E. and. (2011). *Open Source Software*.
Yu, Y., Yin, G., Wang, H., & Wang, T. (2014). Exploring the Patterns of Social Behavior in GitHub. *Proceedings of the 1st International Workshop on Crowd-Based Software Development Methods and Technologies*, 31–36. http://doi.org/10.1145/2666539.2666571

Appendix A. CAR-MASPIOSSD study results

The figure below is a direct copy of the results of the meta-analysis of 40 empirical papers on open source motivation from “Carrots and rainbows: Motivation and social practice in open source software development" referred to as CAR-MASPIOSSD in chapter 6.4.3 (Krogh et al., 2012)

	Intrinsic				Internalized Extrinsic				Extrinsic	
	Ideology	Altruism	Kinship	Fun	Reputation	Reciprocity	Learning	Own-use	Career	Pay
Alexy and Leitner 2007										✓
Baldwin and Clark 2006										
Benkler 2002				✓						✓
Berquist and Ljungberg 2001						✓				
Bitzer et al. 2007		✓						✓		
David and Shapiro 2008	✓		✓			✓	✓	✓		
David et al. 2003	✓					✓	✓	✓		
Fershtman and Gandal 2004										
Ghosh 2005	✓	✓			✓		✓	✓	✓	✓
Hars and Ou 2002		✓	✓		✓		✓	✓	✓	
Haruvy et al. 2003		✓								
Hemetsberger 2004	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Hertel et al. 2003	✓		✓	✓	✓			✓	✓	✓
Ke and Zhang 2008		✓								
Lakhani and von Hippel 2003	✓			✓	✓	✓		✓		
Lakhani and Wolf 2005	✓		✓	✓	✓	✓	✓	✓	✓	✓
Lattemann and Stieglitz 2005					✓			✓		✓
Lee and Cole 2003										
Lerner and Tirole 2002					✓				✓	
Luthiger and Jungwirth 2007				✓						✓
Markus 2007										
Okoli and Oh 2007					✓					
O’Mahony and Ferraro 2007										
Oreg and Nov 2008		✓			✓		✓			
Osterloh and Rota 2007		✓						✓		
Riehle 2007									✓	
Roberts et al. 2006				✓	✓		✓	✓	✓	✓
Rullani 2007										
Schofield and Cooper 2006		✓	✓					✓		
Shah 2006	✓			✓		✓	✓	✓		
Spaeth et al. 2008					✓		✓			
Stewart et al. 2006										
Stewart and Gosain 2006	✓	✓			✓		✓			
von Hippel and von Krogh 2003				✓			✓	✓		
von Krogh et al. 2003										
Wu et al. 2007		✓					✓	✓	✓	
Xu et al. 2009	✓			✓	✓		✓	✓		
Ye and Kishida 2003							✓			
Yu et al. 2007	✓	✓			✓	✓	✓		✓	
Zeitlyn 2003			✓							

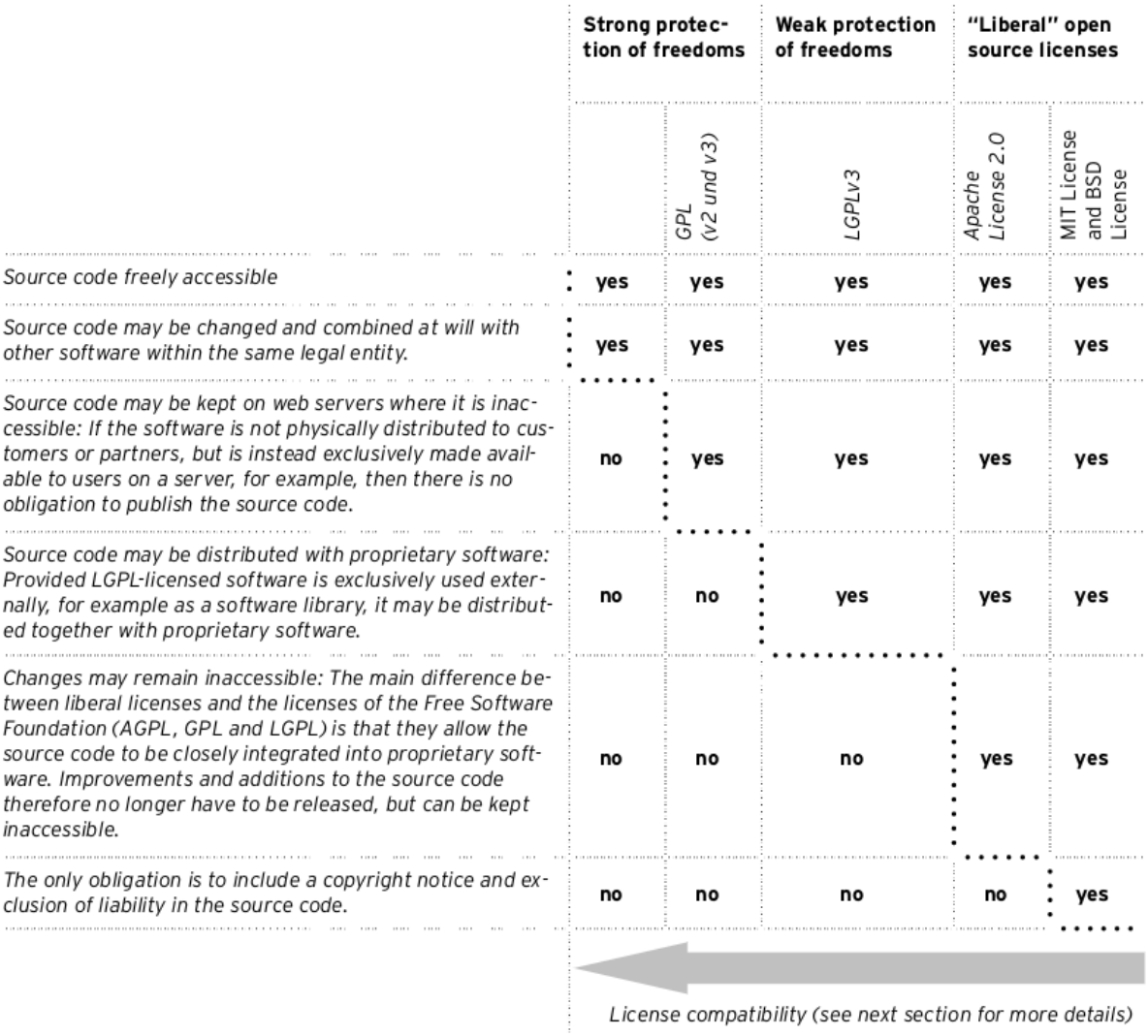
Appendix B. Term definitions for Matthew Aslett's framework

Below is a verbatim copy of Matthew Aslett's exact definitions of the terms used in his open source business strategy framework (Aslett, 2011) but rearranged into a table.

Terms as defined by Matthew Aslett's in his elements of open source business strategy framework.	
Software license	
Strong copyleft	Reciprocal licenses that ensure redistributed modifications and derived works based on or including the code must be made available under the same license. For example the GNU GPL and the Affero GPL
Weak-copyleft	Reciprocal licenses that enable integration with closed source software without the entire derived work having to be made available under the same license. For example the GNU Lesser GPL, the Eclipse Public License, the Mozilla Public License, the Common Development and Distribution License (CDDL)
Non-copyleft	Permissive licenses that do not place restrictions on code usage, enabling it to be integrated with closed source software and the combined code to be distributed under a closed source license. For example BSD licenses, the X11/MIT license, the Apache License
No preference	The vendor commercializes software that combines or utilizes multiple open source software licenses and has no discernible preference
Development model	
The Cathedral	The source code is available with each software release, but is developed privately by an exclusive group of developers
The Bazaar	The code is developed in public, with builds and updates constantly made available on a public forge available to anyone
Aggregate	The vendor commercializes software that utilizes a combination of publicly and privately developed software and has no discernible preference
Community	The software is predominantly developed by a community
Vendor	The software is predominantly developed by a vendor
Mixed	The vendor commercializes software that utilizes a combination of community- and vendor-developed software and has no discernible preference
Copyright control	
Vendor	The copyright is owned by a single vendor.
Foundation	The copyright is owned by a foundation.
Distributed	Copyright ownership is distributed across the individual developers.
Withheld	The copyright is owned by another vendor.
End user licensing	
Single open source	The software and all associated features are available under a single open source license
Multiple open source	The software and all associated features are available using a combination of open source licenses
Dual licensing	The software is available using an open source license, or a closed source license
Open core	The core project is open source, but a version with additional functionality is available using a closed source license
Open complement	Complementary products and services are available using a closed source license
Open edge	The core product is closed source, but extensions and complementary features are open source.
Open foundation	The core product is closed source, but is built on open source software
Open platform	Open source software has been used to create a platform for the delivery of software services and Web applications
Revenue triggers	
Closed source license	Either for a version of the full project, or a larger software package or hardware appliance based on the project, or for extensions to the open source core
Support subscription	An annual, repeatable support and service agreement
Value-add subscription	An annual, repeatable support and service agreement with additional features/functionality delivered as a service
Service/support	Ad hoc support calls, service, training and consulting contracts
Software services	Users pay to access and use the software via hosted or cloud services
Advertising	The software is free to use and is funded by associated advertising
Custom Development	Customers pay for the software to be customized to meet their specific requirements
Other Products and Services	The open source software is not used to directly generate revenue. Complementary products provide the revenue

Appendix C. Obligations of open source licences

The illustration below gives an overview of open source license characteristics and is a direct copy from a report by Ernst and Young (Young, 2011). Permissive licenses are shown at the right, while restrictive licences are shown at the right. The left-most column with the most restrictive license is erroneously without a label in the source report. The correct label here appears to be “AGPL”.



Appendix D. Large open source projects

The illustration below provides an overview of large successful open source projects and their ownership (or governance). It is an direct copy from the presentation “How to grow your open source project 10x and revenues 5x” (Ingo, 2011):

XtraLarge 1000+ devs 100+ commits/day	Linux, KDE, Apache, Drupal, Eclipse, Perl+CPAN, Mozilla+Addons, Gnome		
Large 20-200 devs 50-100 commits	GCC, Python, Samba	MySQL, Qt, OpenOffice, Mono, JBoss, OpenJDK	PHP+PEAR
Medium	GIMP	Subversion, GhostScript, Wordpress	phpMyAdmin
Missing data	Xorg, GNU system tools		
	Foundation	**Vendor**	**"Just a project"**

Appendix E. Detailing extrinsic motivations

In self-determination theory (SDT) discussed in chapter 6.4.1, **extrinsic motivation** can have several forms which are listed and explained below (R. Ryan & Deci, 2000):

- **External regulation**
- **Introjection**
- **Identification**
- **Integration**

One type of extrinsic motivation, based on **external regulation**, is experienced as **controlled**, which is the carrot and stick approach by reward or punishment. It does not work very well. People that are in a controlled motivation situation tend to take short-paths that may negatively affect quality/ethics of their work. It also has negative long-term consequences for performance and well-being (Deci, 2012; R. M. Ryan & Deci, 2000)

Another type of extrinsic motivation, based on **introjection**, is experienced as pressure to perform actions to avoid anxiety and guilt or to gain pride, ego-enhancements, self-esteem or feeling of being of value. Ultimately, resulting behaviour is experienced as controlled and is thus not an effective motivator. (R. Ryan & Deci, 2000)

Another type of extrinsic motivation, based on **identification**, is when a person accepts and identifies with the importance of an external impressed behaviour. In this weakly **autonomous** form of extrinsic motivation, the individuals agree to do the work because of his/her own choice and because of its instrumental value that he/she agrees is worthwhile (so not because of compliance). (R. Ryan & Deci, 2000)

A final type of extrinsic motivation, based on **integration**, is when a person not just accepts and identifies with the importance of an external impressed behaviour but also deeply assimilates the regulation in his/her own system of values and needs. Consequently, that person will have difficulty in not doing the impressed behaviour because it is something that is now a part of that person's self. Integrated motivation is autonomous and free from conflict but still external as the resulting behaviour is done for its instrumental value in achieving some outcome. (R. Ryan & Deci, 2000)

Appendix F. Governance of open source projects

Vendor driven open source projects are typically using a closed **governance model** where the vendor dominates, while community driven open source projects are almost exclusively open (Ingo, 2011) . For open community driven projects, there are two major forms of governance: Benevolent dictator model and meritocracy.

The **benevolent dictator** model is essentially a situation where final decision-making authority sits with a judge/arbitrator approved by the community. Commonly, a wise dictator influences discussions, defers to specialists of individual modules and makes decisions only when no consensus can be reached. In reality, the power of the benevolent dictator is much restrained by contributors' freedom to leave or fork (copy) a project (Fogel, 2015). Linux is an example of a project governed by a benevolent dictator, Linus. (Hertel, Niedner, & Herrmann, 2003)

Meritocracy is a form of democracy based on merit: "The more work you have done, the more you are allowed to do". For a meritocracy to work promotions are from internal ranks only and are based on past contributions. The Apache project is an example of meritocracy in action (Roberts et al., 2006, p. 2 Attributed to Roy Fielding from the Apache project). Community led open source projects tend to evolve into democratic models, which can better survive different dictators or absence of one and is thus more "evolutionary stable" (Fogel, 2015, p. 74).

For open source projects **enforcement** of agreements can be done internally or by third-party enforcement enabled by signed contracts. Third-party enforcement is problematic, as it will negatively affect participation, in particular of individuals.

Research shows that **self-governance** mechanisms tend to be more effective than 3rd-party enforcement (Stürmer, 2009). Here enforcement can be assured if a sufficient number of intrinsic motivated people feel obligated to punish rule-breakers. In open source communities, individuals are sanctioned by "violently blaming individuals on the Internet or refusing to respond to communicate with said individuals. Such sanctions are reported to be quite effective (Osterloh & Rota, 2004).

Another benefit of self-governance mechanisms it that they are more likely to represent a sustainable competitive advantage than signed contracts because trust and reputation components of self-enforcement is harder to copy (Stürmer, 2009)

Finally, governance mechanisms should ensure that the community members in opposition to decisions are treated fairly and respectfully. Respecting other viewpoints and recognizing that they have rational reason for what they do is something to keep in mind when governing an open source project. Universally when decisions are made, when you state or do something publicly, one can generally assume that 70% won't care, 20% will support you and 10% will be against. The 70% that do not care, do care however that the detracting 10% are treated fairly and respectfully (Mickos, 2011 attributed to Jonathan Ian Schwartz, former CEO of Sun Microsystems)